ChoiceMetrics Pty Ltd

# Ngene 1.2
# USER MANUAL & REFERENCE GUIDE

The Cutting Edge in Experimental Design

Version: 30-Jan-18

Email: contact@choice-metrics.com
Website: www.choice-metrics.com

## End-User License Agreement

This is a contract between you and ChoiceMetrics. The software product refers to the computer software and documentation as well as any upgrades, modified versions, copies or supplements supplied by ChoiceMetrics. By installing, downloading, accessing or otherwise using the software product, you agree to be bound by the terms and conditions of this agreement.

### Copyright and Intellectual Property

This software product is copyrighted by, and all rights are reserved by ChoiceMetrics. No part of this software product, either the software or the documentation, may be reproduced, distributed, downloaded, stored in a retrieval system, transmitted in any form or by any means, sold or transferred without prior permission of ChoiceMetrics. You may not modify, adapt, translate, or change the software product. You may not reverse engineer, decompile, dissemble, or otherwise attempt to discover the source code of the software product. The software product is licensed, not sold. Your possession, installation and use of the software product does not transfer to you any title and intellectual property rights, nor does this license grant you any rights in connection with software product trademarks.

### Use of the Software Product

You have only the non-exclusive right to use this software product. A single computer license is registered to one specific computer, and is not intended for installation on a network or in a computer laboratory. For a multi-computer site license, the specific terms of the site license agreement apply for scope of use and installation.

### Limited Warranty

ChoiceMetrics warrants that the software product will perform substantially in accordance with the documentation for a period of ninety (90) days from the date of the original purchase. To make a warrnaty claim, you must notify ChoiceMetrics in writing within ninety (90) days from the date of the original purchase and return the defective software to ChoiceMetrics. If the software does not perform substantially in accordance with the documentation, the entire liability and your exclusive remedy shall be limited to, at ChoiceMetric's option, the replacement of the software product or refund of the license fee paid to ChoiceMetrics for the software product. Proof of purchase from an authorized source is required. This limited warranty is void if failure of the software product has resulted from accident, abuse, or misapplication. Some states and jurisdictions do not allow limits on the duration of an implied warranty, so the above limitation may not apply to you. To the extent permissible, any implied warranties on the software product are limited to ninety (90) days.

ChoiceMetrics does not warrant the performance or results you may obtain by using the software product. To the maximum extent permitted by applicable law, ChoiceMetrics disclaims all other warranties and conditions, either express or implied, including, but not limited to, implied warranties of merchantability, fitness for a particular purpose, title, and non-infringement with respect to the software product. This limited warranty gives you specific legal rights. You may have others, which vary from state to state and jurisdiction to jurisdiction.

### Limitation of Liability

Under no circumstances will ChoiceMetrics be liable to you or any other person for any indirect, special, incidental, or consequential damages whatsoever (including, without limitation, damages for loss of business profit, business interruption, computer failure or malfunction, loss of business information, or any other pecuniary loss) arising out of the use or inability to use the software product, even if ChoiceMetrics has been advised of the possibility of such damages. In any case, ChoiceMetrics's entire liability under any provision of this agreement shall not exceed the amount paid to ChoiceMetrics for the software product. Some states or jurisdictions do not allow the exclusion or limitation of liability for incidental or consequential damages, so the above limitation may not apply to you.

# Table of Contents

# Chapter 1

# Introduction

# 1 Introduction

This user manual and reference guide describes how to use the Ngene software and also briefly introduces the underlying methodology.

## 1.1 What is Ngene?

Ngene is software for generating experimental designs that are used in stated choice experiments for the purpose of estimating choice models, particularly of the logit type.

Ngene is distributed by ChoiceMetrics (www.choice-metrics.com). The syntax used in Ngene is similar to that used in Nlogit/Limdep.

## 1.2 About Version 1

Ngene 1 is the first commercial release of this software. It has a modern graphical interface and state-of-the-art methods for generating a wide range of experimental designs. Ngene allows for the generation of orthogonal designs, optimal orthogonal designs and efficient stated choice designs. Ngene 1 supports orthogonal main effects only designs and for efficient designs, supports main effects and interaction effects for MNL, MMNL panel and cross sectional and EC panel and cross sectional models. Ngene also allows for constraints and nesting of attributes for different types of designs. Ngene allows the user to open and read existing data, for example to evaluate designs that may have been generated elsewhere.

Ngene 1 also allows the user to build the HTML code for any design generated. The user may take an existing design (even one generated using other software) and build step by step the HTML code for presenting that design. The user will have to write their own code to capture data using the design, however for those wishing to show clients what the experiment might look like in practice, this feature will allow for a quick solution without having to first write the complete survey themselves.

Point releases are released periodically as a free upgrade, and add minor functionality and fix bugs. The current release is Ngene 1.1.1. Check the website to see if a more recent version of Ngene is available.

## 1.3 Feature overview

Ngene is designed to be the single source of stated choice (SC) experimental designs. As such, it has an extensive range of features and outputs.

With Ngene you can:

**Specify designs with great flexibility:**

- Generate designs with any number of choice situations, alternatives, attributes and attribute levels.
- Maintain attribute level balance, or specify that an attribute must occur an exact number of times or between a minimum and maximum number of times.
- Dummy and effects code attributes.

- Specify logical rules to limit what attribute levels can coexist in a choice situation.
- Interrogate design level correlations as calculated using a range of correlation formulas (Pearson product moment, G index, J index, Spearman rank, Point biserial, CP coefficient, H index).

**Generate full and fractional factorial designs.**

**Generate orthogonal designs:**

- Maintain orthogonality either across or within alternatives.
- Obtain orthogonal designs for a very large range of design dimensions.
- Add blocking and foldover columns.
- Generate optimal orthogonal in the differences designs.
- Find the most efficient orthogonal design.

**Generate efficient designs:**

- Report and optimize on efficiency measures including d, a, s (sample size), b (utility balance), and wtp (willingness to pay).
- Report and optimize on efficiency measures for multinomial logit (MNL) models, mixed multinomial logit (MMNL) models (panel and cross sectional) and error components (EC) models (panel and cross sectional).
- Account for prior uncertainty with normally and uniformly distributed Bayesian priors.
- Report and optimize on the Bayesian mean, median, minimum, maximum and standard deviation.
- Draw Bayesian and random parameter distributions with random, Halton, Sobol and MLHS draws, as well as Gaussian quadrature.
- Optimize on more than one model and error measure type using model averaging.
- Search for efficient designs using the pair swapping, RSC and modified Federov algorithms.
- Report utilities, probabilities, the Fisher matrix and the covariance matrix for each model type.

**Generate formatted HTML mockups:**

- Format the scenarios by placing design levels, text and radio buttons wherever you like within a table.
- Format or relabel design levels for presentation.
- Apply cascading style sheets (CSS) to instantly modify the appearance of the formatted scenarios (CSS files included, or create you own).
- View the mockups directly within Ngene.

**Interact with a modern user interface that maximizes flexibility:**

- Open and evaluate existing data files and designs.
- Open files independently in the workspace or maintain syntax, data and output files within a project.
- Retain all syntax runs and associated outputs during a session.
- Interrogate any design found during a search.
- Report design properties as needed - no need to decide what to report before the syntax runs, and no calculation of unnecessary properties during the search.
- Easily view any number of user selected design properties in a grid, and copy directly to other applications including Microsoft Excel.

and do much more...

## 1.4 Overview of this manual

This manual is arranged so that the functions you are most likely to use are the ones you will find documented first. We have attempted to be concise, where possible substituting hands-on examples for lengthy prose about particular aspects. Nonetheless, in order to be complete, this manual is necessarily longer than we would have hoped. In spite of this, first time users should take the time to skim the first few paragraphs of each chapter before beginning serious use.

This manual is broken into chapters:

- Chapter 1 "Introduction" gives a brief introduction to the capabilities of Ngene, and the contents of this manual.
- Chapter 2 "Installation and Setup" provides instructions for the installation and setup of the Ngene software.
- Chapter 3 "The Ngene Workspace" explores the graphical operating environment and its various components.
- Chapter 4 "Ngene Syntax" is an introduction to the structure of the syntax that the analyst uses to control Ngene.
- Chapter 5 "Introduction to Experimental Design Theory" provides an introduction to experimental design theory. It is recommended to read this chapter before reading subsequent chapters.
- Chapter 6 "Orthogonal Designs" discusses the theory of orthogonal designs, and guides the user through the construction of various types of orthogonal designs in Ngene.
- Chapter 7 "Efficient Designs" introduces the theory of efficient designs, and demonstrates the basic features of efficient design generation in Ngene.
- Chapter 8 "Advanced Features in Generating Efficient Designs" describes some state-of-the-art design generation techniques that can be utilized in Ngene.
- Chapter 9 "Designs With Continuous Attribute Levels" examines designs that allow some attributes to have continuous attribute levels.
- Chapter 10 "Formatting Experiments" explores the tools that Ngene provides for creating HTML survey mockups using the generated experimental designs.
- Chapter 11 "Syntax Reference" outlines in detail the permissible syntax of each of Ngene's commands and properties.
- Chapter 12 "Endnotes" contains endnotes from the entire manual.
- Chapter 13 "References" lists all references cited in the manual.

**Key concepts.** Some pieces of information are very important. To make them stand out from the rest of the documentation, these 'key concepts' will be presented in a yellow box such as this one.

# Chapter 2

# Installation and Setup

# 2 Installation and Setup

## 2.1 Installing Ngene

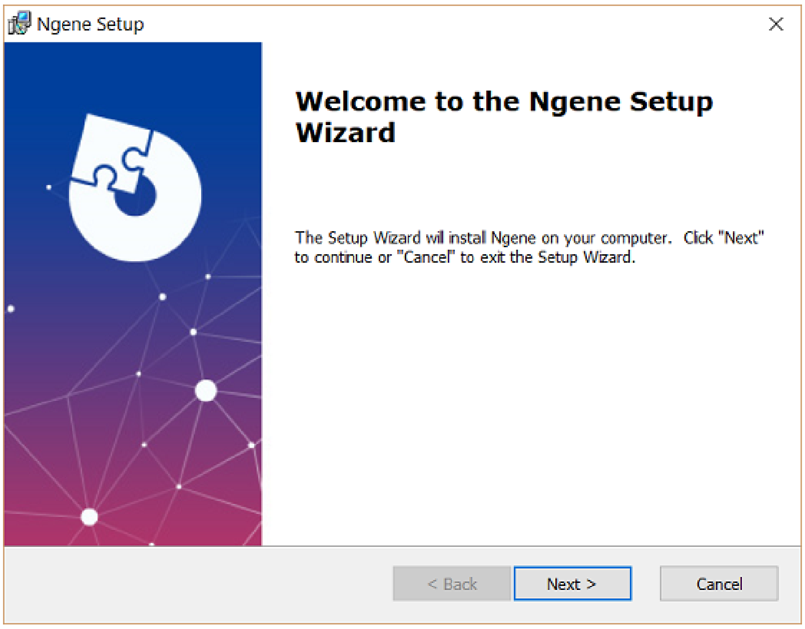
Ngene is a Windows based program (there is no Macintosh version). As of version 1.0.2, Ngene can run on computers installed with 64 bit versions of Windows.

To install Ngene:

**1.Install .NET 4.0, if necessary**
Ngene requires .NET 4.0 to run. If you do not already have .NET 4.0 installed, you can download the latest version from the Microsoft website. If you are uncertain if .NET 4.0 is installed, attempt step 3 - an error message will be shown if .NET 4.0 is not installed.

**2.Obtain the Ngene installer.**
Download the installer EXE file from www.choice-metrics.com/download. The file is moderately large - approximately 100MB.

**3.Navigate to and run the setup program**
Run the program 'Ngene setup.exe'. You can change the installation location if you wish, and install for either all users of the computer, or just yourself.



Welcome to the Ngene Setup Wizard

The Setup Wizard will install Ngene on your computer. Click "Next" to continue or "Cancel" to exit the Setup Wizard.

**One screen of the setup program**

**4.Run Ngene**
A shortcut called 'Ngene' will have been placed in your Start menu. Open this shortcut to run Ngene.

**5.Activate Ngene**
If you purchased your copy of Ngene, refer to the section License activation and management for information on how to activate your copy of Ngene. Otherwise, Ngene will run as an evaluation version.

## 2.2 Evaluation version

Until you activate Ngene, Ngene will run as an evaluation version. Ngene can only be activated if you have purchased the software. If you have purchased the software, refer to the next section, License activation and management, for details on how to activate Ngene.

We have provided the evaluation version to allow you to see how Ngene works, and experience first hand all of the features that it provides. You can pass the software on to others freely. The only limitation in the evaluation version is that all design values will appear as "0", with the real design levels being obfuscated. All other functionality will be complete.

## 2.3 Purchasing Ngene

Ngene can be purchased securely online through PayPal, or by bank transfer. For up to date details on how to pay, including current pricing, visit www.choice-metrics.com/purchase.

## 2.4 License activation and management

Single licenses of Ngene allow the software to be used on at most two computers. We understand that many people want a copy for their desktop computer and for their laptop.

The full version of Ngene is activated using a license ID and password provided by ChoiceMetrics, with the software being locked to a single computer after activation. Moving the software with the license file to another computer will cause Ngene to revert to the evaluation version on that computer.

**Note that purchasing Ngene is not instantaneous.** We will need to check that payment has cleared before we send you your password. Please do not leave the purchase of Ngene to the last minute if you need it for a project.

There are two mechanisms for activating Ngene: online activation and manual activation. We strongly recommend online activation, as it is faster and more convenient. If however you do not have internet access on the computer you wish to activate, you may need to manually activate the software. Both methods are described below.

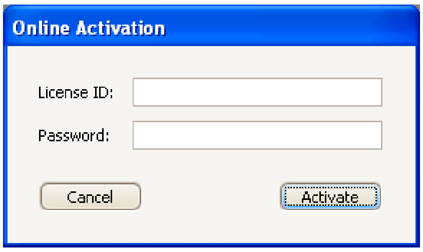
### Online activation

**1.** Purchase Ngene. We will email you a receipt, a license ID, and a password.

**2.** If you have not already done so, download Ngene from the ChoiceMetrics website at www.choice-metrics.com/download and install the software.

**3.** Run Ngene.

**4.** From the Help menu, select ‘Online Activation’. A dialog box will appear, similar to that below.



**The online activation dialog box**

**5.** Enter the license ID and password provided to you when you purchased Ngene. If you have lost these details, email sales@choice-metrics.com and we will send through the details again. Your software should now be activated.

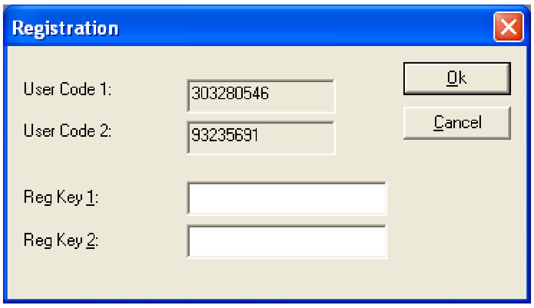
## Manual activation

**1.** Purchase Ngene.

**2.** If you have not already done so, download Ngene from the ChoiceMetrics website at www.choice-metrics.com/download and install the software.

**3.** Run Ngene.

**4.** From the Help menu, select ‘Manual Activation’. A dialog box will appear, similar to that below.



Registration

User Code 1: 303280546

User Code 2: 93235691

Reg Key 1:

Reg Key 2:

**The manual activation dialog box**

**5.** Email sales@choice-metrics.com with ‘User Code 1’ and ‘User Code 2’.

**6.** We will reply with an activation code. Run Ngene. From the 'Help' menu, again select ‘Manual Activation’. Enter the activation code into the field 'Reg Key 1:'. Your copy of Ngene should now be activated.

Note that the user codes will sometimes reset before the user codes can be entered. You will need to email us again with the updated user codes. This issue is outside of our control, and is another reason why online activation is preferable.

### What if I change computers?

If you received a licence ID and password for online activation (i.e. you purchased the software after late 2009), the same details will allow you to activate Ngene after you have upgraded or changed computers. If the activation fails due to insufficient activations being available, email contact@choice-metrics.com. If you purchased Ngene in 2009, you may not have received a license ID and password. Email contact@choice-metrics.com to obtain these details. Alternatively, you can request a manual activation.

If you have merely upgraded a part of your computer, it is unlikely that you will need a new activation, although this is a possibility. For example, updating the operating system or installing a new hard drive is unlikely to cause any problems.

### What if I uninstall Ngene?

If you uninstall Ngene, the license file "Ngene.lf" will be left in the folder in which Ngene was installed. So long as this file is left in place, future installations of Ngene *to the same folder on the same computer* will not need activation. It is strongly recommended that you create a backup of the "Ngene.lf" file.

## 2.5 Updating Ngene

To install a more recent version of Ngene, simply download the installation file for the new version, and run the installer. The old version will be removed and the new version installed. If you have activated Ngene, it will remain activated after the update, so long as you install the new version to the same location as the old version.

You can find the version of the installed copy of Ngene in the About window, accessible from the Help menu.

The best way to check for and download updates is via the Check for Updates feature in the Help menu. You will either be notified that your software is up to date, or given an option to download the latest version, without having to visit the ChoiceMetrics website.

## 2.6 Uninstalling Ngene

Navigate to the Control Panel, and open 'Add or Remove Program'. Select 'Ngene' from the list, and then its associated 'Uninstall' button.

# Chapter 3

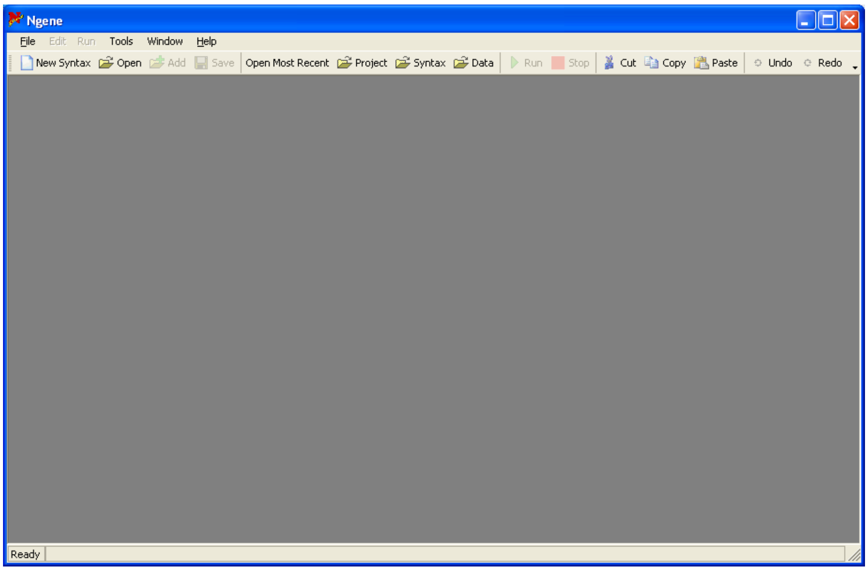
# The Ngene Workspace

# 3 The Ngene Workspace

This chapter will explain how to navigate within and operate the Ngene workspace. Ngene is primarily command driven, where the commands are stored in a syntax file and entered via the user's keyboard. However, the rest of Ngene utilizes a rich graphical user interface (GUI), the components of which are described in this chapter.

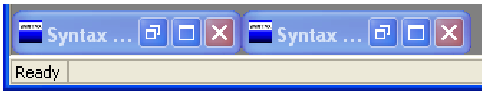
## 3.1 Workspace overview

Upon starting Ngene, a blank workspace will appear as below. Initially, the workspace consists only of a menu bar and a toolbar with buttons. When performing tasks within Ngene, new windows will appear within the workspace.



**The Ngene workspace as it appears on startup**

The windows can be minimized, so that the window appears at the bottom of the workspace as shown below.



**Windows minimized within the workspace**

Various files can be opened and represented within Ngene as windows, including:

- Ngene project files
- Ngene syntax files
- Ngene design files
- Excel files
- Comma separated files

Of these files, only one project can be open at any one time. There is no limit to how many of the remaining file types can be open within the main operating environment.

A key distinction can be made within Ngene between a managed and an unmanaged workspace.

A managed workspace is controlled by an open project. All new files created will automatically be added to the project folder, and files external to the project that are opened will be copied to the project's folder.

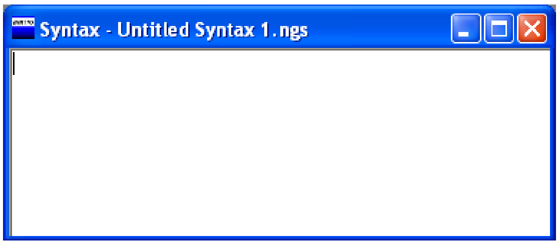
An unmanaged workspace exists when no project is open. All new files will not be stored until they are saved explicitly, and files will be opened from their original location.

The choice of which type of workspace to use will depend on the user's preferences, and the number of files and designs they wish to work with.

The following sections outline the files that can be opened, their purpose, and how they are handled and visualized within Ngene.

## 3.2 Syntax windows and files

Whilst some functionality can be invoked in Ngene via the menus, syntax is the primary method of controlling the program. Syntax is entered as plain text into a syntax window, an example of which is below. To run syntax, the syntax window that contains the relevant syntax must be made active, and the Run menu item selected. The results of the run will be displayed in the Output window.



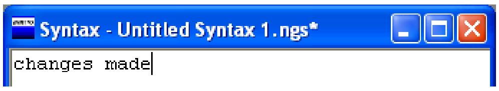
**An empty syntax window**

A description of the structure of Ngene syntax is covered in Ngene Syntax, together with a simple example. The syntax is introduced across several chapters, and the manual also contains a Syntax Reference chapter.

### Syntax files

Ngene syntax is stored in syntax files, which have a .ngs suffix. These are plain text files, and so are portable and can be opened by a wide range of programs. However, Ngene registers these files so that they will open in Ngene by default. The File menu section describes how syntax files can be created, opened and saved.

When a syntax file has not been saved since changes were made, a star will appear next to the file name in that file's syntax window (see below).



```
changes made
```

**A syntax file that has had changes since it was last saved**

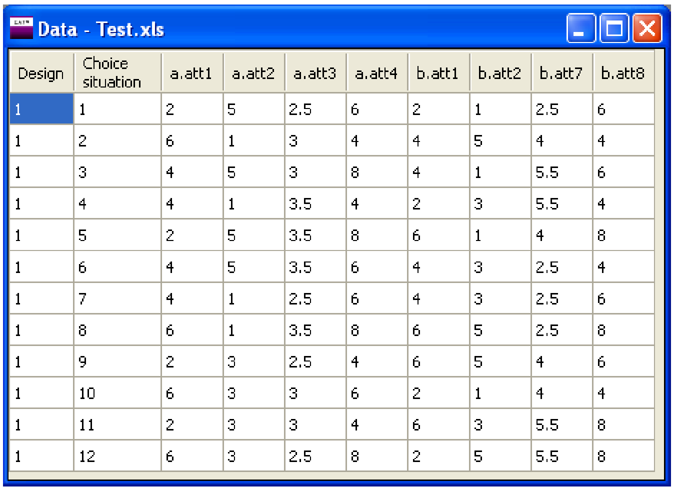
## 3.3 Data windows and files

Various functions in Ngene may require that the user access data files. For example, the analyst may wish to evaluate an existing design stored in an Excel file.

The current release version of Ngene supports the access of Excel files (including the new .xlsx and .xlsm file formats), comma separated (CSV) files, semicolon delimited files, and tab delimited files. Data file access is read-only, so the data can be viewed within Ngene and used by the routines, but may not be modified. Memory permitting, any number of datasets can be opened simultaneously.

The File menu section describes how data files can be opened.

Below is a screenshot of an Excel file containing a design that has been opened with Ngene.

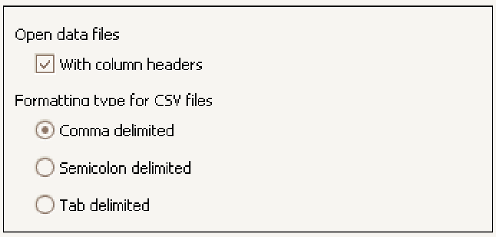


| Design | Choice situation | a.att1 | a.att2 | a.att3 | a.att4 | b.att1 | b.att2 | b.att7 | b.att8 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 5 | 2.5 | 6 | 2 | 1 | 2.5 | 6 |
| 1 | 2 | 6 | 1 | 3 | 4 | 4 | 5 | 4 | 4 |
| 1 | 3 | 4 | 5 | 3 | 8 | 4 | 1 | 5.5 | 6 |
| 1 | 4 | 4 | 1 | 3.5 | 4 | 2 | 3 | 5.5 | 4 |
| 1 | 5 | 2 | 5 | 3.5 | 8 | 6 | 1 | 4 | 8 |
| 1 | 6 | 4 | 5 | 3.5 | 6 | 4 | 3 | 2.5 | 4 |
| 1 | 7 | 4 | 1 | 2.5 | 6 | 4 | 3 | 2.5 | 6 |
| 1 | 8 | 6 | 1 | 3.5 | 8 | 6 | 5 | 2.5 | 8 |
| 1 | 9 | 2 | 3 | 2.5 | 4 | 6 | 5 | 4 | 6 |
| 1 | 10 | 6 | 3 | 3 | 6 | 2 | 1 | 4 | 4 |
| 1 | 11 | 2 | 3 | 3 | 4 | 6 | 3 | 5.5 | 8 |
| 1 | 12 | 6 | 3 | 2.5 | 8 | 2 | 5 | 5.5 | 8 |

**An Excel file displayed within Ngene**

If the data file represents a design, and is to be used with the *eval* or *start* properties, the columns need to exist in a specific order. The first column contains the design number. In most cases, there will just be a single design. In this case, all rows should contain the value 1. If *D* designs are to be evaluated, perhaps for a heterogeneous design, then the designs should appear in order, with the first column containing values 1...*D*. The second column contains the choice situation number. One choice situation should be specified per row, with the values increasing from 1 to *S* in order for every design (where there are *S* choice situations per design). The remaining columns contain the attributes, and should be specified in the same order as the attributes are declared in the syntax that will be used to evaluate the design. Note that constants in a utility expression are not treated as attributes, and should not be stored in the data file. The above example contains a single design with 12 choice situations and eight attributes.

Ngene can either treat the first row in the data file as a header, or the first row of actual data. For the former, the first row should contain names for each column, and the actual data should be specified from the second row. For the later, no column names need to be specified, and the data can begin from the first row. To change this setting, select Session Options or Permanent Options from the Tools menu, and select the General tab (see below for the relevant part of this screen). Check or uncheck the "With column headers" check box. All data files will be opened with this setting.



Open data files

☑ With column headers

Formatting type for CSV files

● Comma delimited

○ Semicolon delimited

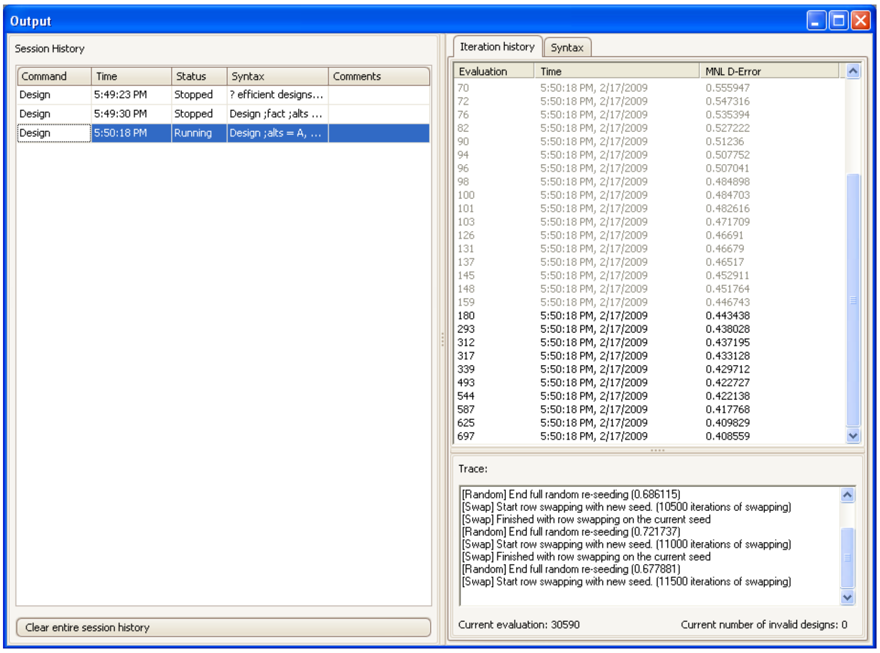
○ Tab delimited

**Preferences for changing how data files are read**

When opening a CSV file, the file can be seperated by commas (the default), semicolons, or tabs. To change this setting, select Session Options or Permanent Options from the Tools menu, and select the General tab (see above for the relevant part of this screen). All data files will be opened with this setting.

## 3.4 Output window

When syntax is run, results are accessed from the Output window, shown below. The Output window is not initially visible, but will automatically open the first time syntax is run in a session.



| Command | Time | Status | Syntax | Comments |
|---|---|---|---|---|
| Design | 5:49:23 PM | Stopped | ? efficient designs… | |
| Design | 5:49:30 PM | Stopped | Design ;fact ;alts … | |
| Design | 5:50:18 PM | Running | Design ;alts = A, … | |

| Evaluation | Time | MNL D-Error |
|---|---|---|
| 70 | 5:50:18 PM, 2/17/2009 | 0.555947 |
| 72 | 5:50:18 PM, 2/17/2009 | 0.547316 |
| 76 | 5:50:18 PM, 2/17/2009 | 0.535394 |
| 82 | 5:50:18 PM, 2/17/2009 | 0.527222 |
| 90 | 5:50:18 PM, 2/17/2009 | 0.51236 |
| 94 | 5:50:18 PM, 2/17/2009 | 0.507752 |
| 96 | 5:50:18 PM, 2/17/2009 | 0.507041 |
| 98 | 5:50:18 PM, 2/17/2009 | 0.484898 |
| 100 | 5:50:18 PM, 2/17/2009 | 0.484703 |
| 101 | 5:50:18 PM, 2/17/2009 | 0.482616 |
| 103 | 5:50:18 PM, 2/17/2009 | 0.471709 |
| 126 | 5:50:18 PM, 2/17/2009 | 0.46691 |
| 131 | 5:50:18 PM, 2/17/2009 | 0.46679 |
| 137 | 5:50:18 PM, 2/17/2009 | 0.46517 |
| 145 | 5:50:18 PM, 2/17/2009 | 0.452911 |
| 148 | 5:50:18 PM, 2/17/2009 | 0.451764 |
| 159 | 5:50:18 PM, 2/17/2009 | 0.446743 |
| 180 | 5:50:18 PM, 2/17/2009 | 0.443438 |
| 293 | 5:50:18 PM, 2/17/2009 | 0.438028 |
| 312 | 5:50:18 PM, 2/17/2009 | 0.437195 |
| 317 | 5:50:18 PM, 2/17/2009 | 0.433128 |
| 339 | 5:50:18 PM, 2/17/2009 | 0.429712 |
| 493 | 5:50:18 PM, 2/17/2009 | 0.422727 |
| 544 | 5:50:18 PM, 2/17/2009 | 0.422138 |
| 587 | 5:50:18 PM, 2/17/2009 | 0.417768 |
| 625 | 5:50:18 PM, 2/17/2009 | 0.409829 |
| 697 | 5:50:18 PM, 2/17/2009 | 0.408559 |

Trace:

```
[Random] End full random re-seeding (0.686115)
[Swap] Start row swapping with new seed. (10500 iterations of swapping)
[Swap] Finished with row swapping on the current seed
[Random] End full random re-seeding (0.721737)
[Swap] Start row swapping with new seed. (11000 iterations of swapping)
[Swap] Finished with row swapping on the current seed
[Random] End full random re-seeding (0.677881)
[Swap] Start row swapping with new seed. (11500 iterations of swapping)
```

Current evaluation: 30590

Current number of invalid designs: 0

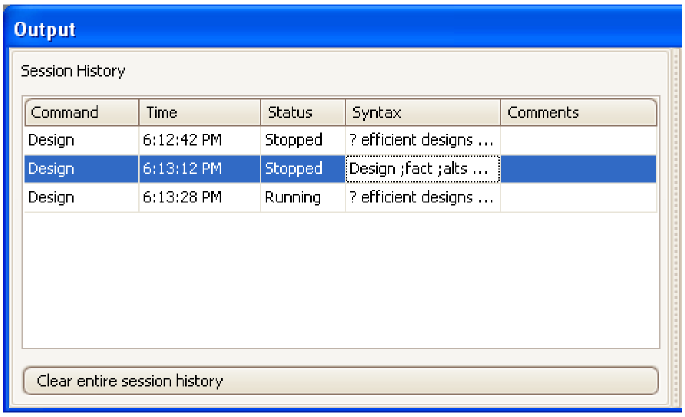
**The Output window**

The Output window consists of several parts, listed below.

### Session History

On the left, the session history is stored. Each time syntax is run, a new row will appear in the Session History list. The row, which represents a single syntax run, contains several fields of information:

- **Command**: the main command that was run.
- **Time**: the time the run commenced.
- **Status**: running, paused, or stopped. Note that only one routine may be run at a time. Hence, the user cannot pause one routine and start a second.
- **Syntax**: the syntax that was run. Placing the curser over a cell in this column will produce a pop-up box that shows the full syntax used for that routine.
- **Comments**: the user may type personal comments here that may be useful for future reference. Also, if an error occurs when the syntax is parsed, Ngene will place the word ‘Error’ here. In doing this, the user will quickly be able to see that the routine for that syntax is not running and hopefully be able to diagnose the problem.

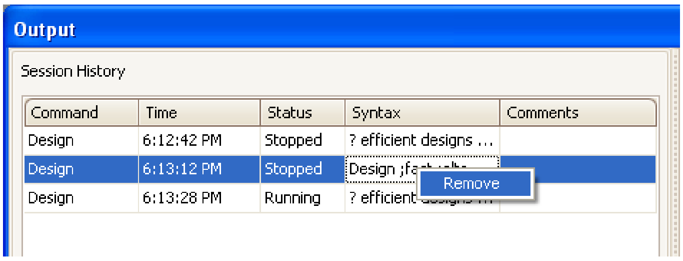


| Command | Time | Status | Syntax | Comments |
|---|---|---|---|---|
| Design | 6:12:42 PM | Stopped | ? efficient designs ... | |
| Design | 6:13:12 PM | Stopped | Design ;fact ;alts ... | |
| Design | 6:13:28 PM | Running | ? efficient designs ... | |

**The Session History**

The user may wish to remove previous syntax runs to free memory. This can be done in one of two ways:

1. The user may press the ‘Clear entire session history’ button located below the Session History list. This will clear all syntax runs that are not running and free the associated memory.
2. The user may remove a single syntax run by right hand clicking on any cell of the corresponding row in the Session History list, and selecting 'Remove' from the popup menu, as shown below.



| Command | Time | Status | Syntax | Comments |
|---|---|---|---|---|
| Design | 6:12:42 PM | Stopped | ? efficient designs ... | |
| Design | 6:13:12 PM | Stopped | Design ;fa | |
| Design | 6:13:28 PM | Running | ? efficient | |

**Removing a single syntax run**

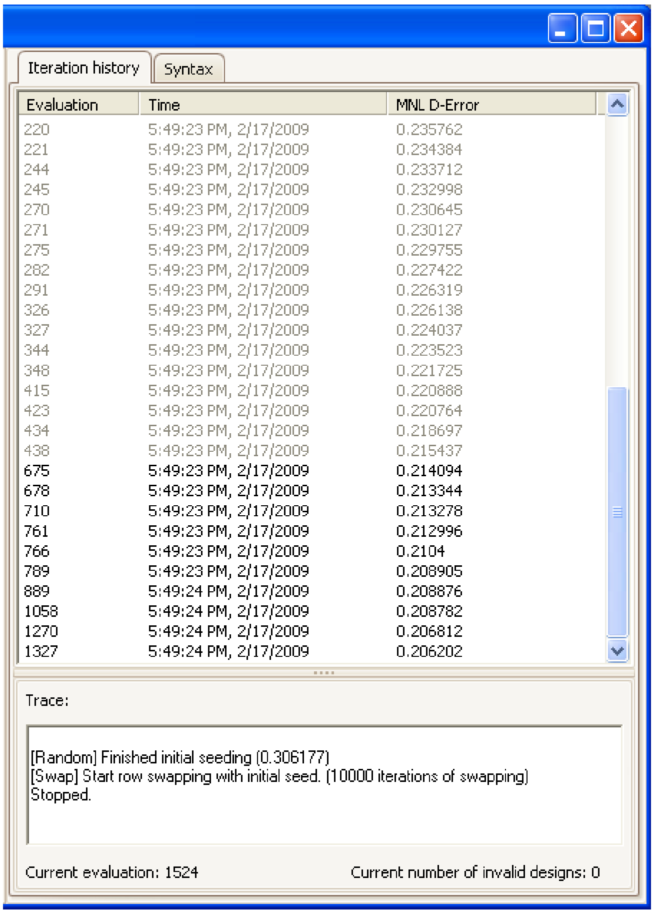
Note that once removed, a syntax run cannot be retrieved. That is, the undo button will not retrieve a removed run. However, any designs that were added to the project or opened in a Design window prior to the removal of the run will still be accessible.

Selecting a syntax run in the Session History load that run's output on the right hand side of the window. Each syntax run is described by two tabs: the Design History tab and the Syntax tab.

## Iteration History tab

The top of the Iteration History tab contains a list of all designs found so far in the syntax run. Sometimes only a single design will be found, at other times there may be very many designs found, as below.

**Very important:** To open a design window and examine the properties of the design, including the design levels, double click on a row in the iteration history.



| Evaluation | Time | MNL D-Error |
|---|---|---|
| 220 | 5:49:23 PM, 2/17/2009 | 0.235762 |
| 221 | 5:49:23 PM, 2/17/2009 | 0.234384 |
| 244 | 5:49:23 PM, 2/17/2009 | 0.233712 |
| 245 | 5:49:23 PM, 2/17/2009 | 0.232998 |
| 270 | 5:49:23 PM, 2/17/2009 | 0.230645 |
| 271 | 5:49:23 PM, 2/17/2009 | 0.230127 |
| 275 | 5:49:23 PM, 2/17/2009 | 0.229755 |
| 282 | 5:49:23 PM, 2/17/2009 | 0.227422 |
| 291 | 5:49:23 PM, 2/17/2009 | 0.226319 |
| 326 | 5:49:23 PM, 2/17/2009 | 0.226138 |
| 327 | 5:49:23 PM, 2/17/2009 | 0.224037 |
| 344 | 5:49:23 PM, 2/17/2009 | 0.223523 |
| 348 | 5:49:23 PM, 2/17/2009 | 0.221725 |
| 415 | 5:49:23 PM, 2/17/2009 | 0.220888 |
| 423 | 5:49:23 PM, 2/17/2009 | 0.220764 |
| 434 | 5:49:23 PM, 2/17/2009 | 0.218697 |
| 438 | 5:49:23 PM, 2/17/2009 | 0.215437 |
| 675 | 5:49:23 PM, 2/17/2009 | 0.214094 |
| 678 | 5:49:23 PM, 2/17/2009 | 0.213344 |
| 710 | 5:49:23 PM, 2/17/2009 | 0.213278 |
| 761 | 5:49:23 PM, 2/17/2009 | 0.212996 |
| 766 | 5:49:23 PM, 2/17/2009 | 0.2104 |
| 789 | 5:49:23 PM, 2/17/2009 | 0.208905 |
| 889 | 5:49:24 PM, 2/17/2009 | 0.208876 |
| 1058 | 5:49:24 PM, 2/17/2009 | 0.208782 |
| 1270 | 5:49:24 PM, 2/17/2009 | 0.206812 |
| 1327 | 5:49:24 PM, 2/17/2009 | 0.206202 |

Trace:

```
[Random] Finished initial seeding (0.306177)
[Swap] Start row swapping with initial seed. (10000 iterations of swapping)
Stopped.
```

Current evaluation: 1524 Current number of invalid designs: 0

**The Iteration history**

Each row in the list represents a single design. Three properties of the design are displayed:

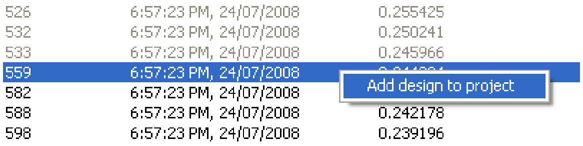
- **Evaluation**: this indicates how many designs have been evaluated in the search to get to this design state.
- **Time**: the time and date that the design was found.
- **A performance measure (optional)**: the actual measure (and so the column heading) will vary depending on the syntax. For example, efficient designs will report the efficiency measure being optimized (e.g. MNL d error in the example above), but orthogonal designs will not report anything for this field.

The information in the row is only a small subset of all the available properties of the design that can be reported. This information is presented in a separate window, the Design window. To open this window and examine this information, double click on the row of the design you wish to examine. See Design windows and files for more information on Design windows.

Some rows may grey out. This means that the design has been deleted and is no longer available. While in Ngene attempts to make as much information available as possible, some designs can consume a large amount of memory, and the deletion of old designs is a strategy for preventing Ngene from running out of memory. The first design is always retained, and then the *N* most recent designs are also retained, where *N* is an integer that can be configured from the Options dialog box, or specified in syntax. Also, all designs can be specified to be retained. The syntax for both these options is:

```
;store = N
;store = all
```

Designs can be added to the current project if it is open. This can either be done from the Design window, from the Add button in the toolbar, or from the design history list. To perform the later, right click on any retained design, and select 'Add design to project' from the popup menu, as shown below.



| | | |
|---|---|---|
| 526 | 6:57:23 PM, 24/07/2008 | 0.255425 |
| 532 | 6:57:23 PM, 24/07/2008 | 0.250241 |
| 533 | 6:57:23 PM, 24/07/2008 | 0.245966 |
| 559 | 6:57:23 PM, 24/07/2008 | [illegible] |
| 582 | 6:57:23 PM, 24/07/2008 | [illegible] |
| 588 | 6:57:23 PM, 24/07/2008 | 0.242178 |
| 598 | 6:57:23 PM, 24/07/2008 | 0.239196 |

**Adding a design to the project from the Iteration History list**

The bottom of the Iteration History tab contains a scrolling text area called 'Trace'. This text area is used by Ngene to provide a variety of feedback to the user, including, but not limited to:

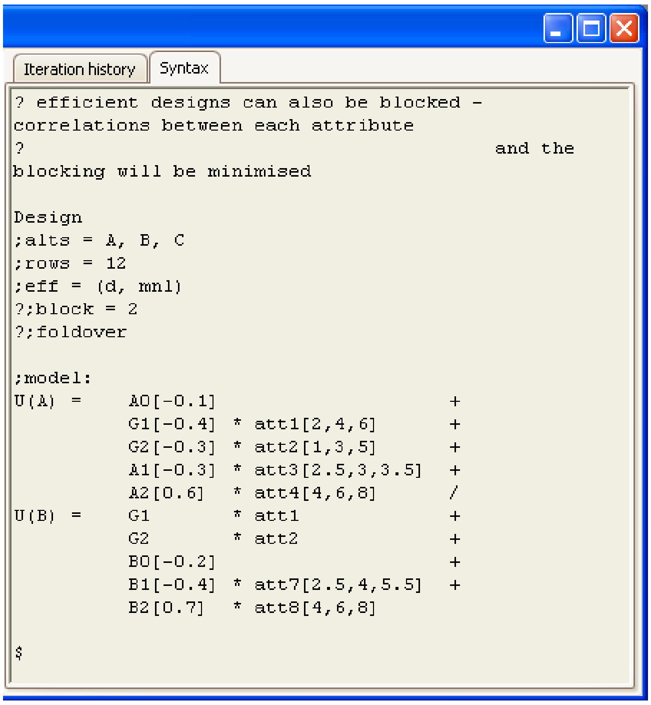
- Syntax error messages that will prematurely terminate a run.
- Warnings that alert the user to potential problems, but will not terminate a run.
- Notifications of assumptions made from a syntax specification.
- Information providing updates on algorithm progress.

A significant effort has been made by Ngene's authors to provide meaningful error messages. However, the authors welcome your feedback and suggestions on unclear messages.

The bottom of the Iteration History tab also presents the 'Current evaluation' number. This can help reassure the user that a search is still running when an improved design has not been found for some time. The 'Current number of invalid designs' reports how many designs have been found that have had to be discarded due to a problem with the evaluation of the design. For example, sometimes the calculation of a Bayesian efficient design results in a singular Fisher matrix. A small number of discarded designs can usually be tolerated, but a large number is symptomatic of some underlying problem, and should be investigated by the analyst.

## Syntax tab

The Syntax tab displays a read-only copy of the syntax that was used for the currently selected syntax run.



```
? efficient designs can also be blocked -
correlations between each attribute
?                                         and the
blocking will be minimised

Design
;alts = A, B, C
;rows = 12
;eff = (d, mnl)
?;block = 2
?;foldover

;model:
U(A) =    A0[-0.1]                    +
          G1[-0.4]  * att1[2,4,6]     +
          G2[-0.3]  * att2[1,3,5]     +
          A1[-0.3]  * att3[2.5,3,3.5] +
          A2[0.6]   * att4[4,6,8]     /
U(B) =    G1        * att1            +
          G2        * att2            +
          B0[-0.2]                    +
          B1[-0.4]  * att7[2.5,4,5.5] +
          B2[0.7]   * att8[4,6,8]

$
```

**The Syntax tab**

## 3.5 Design windows and files

The design window contains all available information about a design. It can be opened by double clicking on a row in the iteration history of the output window.

The title contains several pieces of information:

- **A performance measure (optional)**: If relevant, a performance measure. The actual measure will vary depending on the syntax. For example, efficient designs will report the efficiency measure being optimized (e.g. MNL d error in the example below), but orthogonal designs will not report anything for this field.
- **The evaluation number**: this indicates how many designs have been evaluated in the search to find this design.
- **The syntax filename**: the syntax file that was run to generate the design.

The design window itself contains three tabs, described below.

### Properties tab

The Properties tab, shown below, contains two key components.

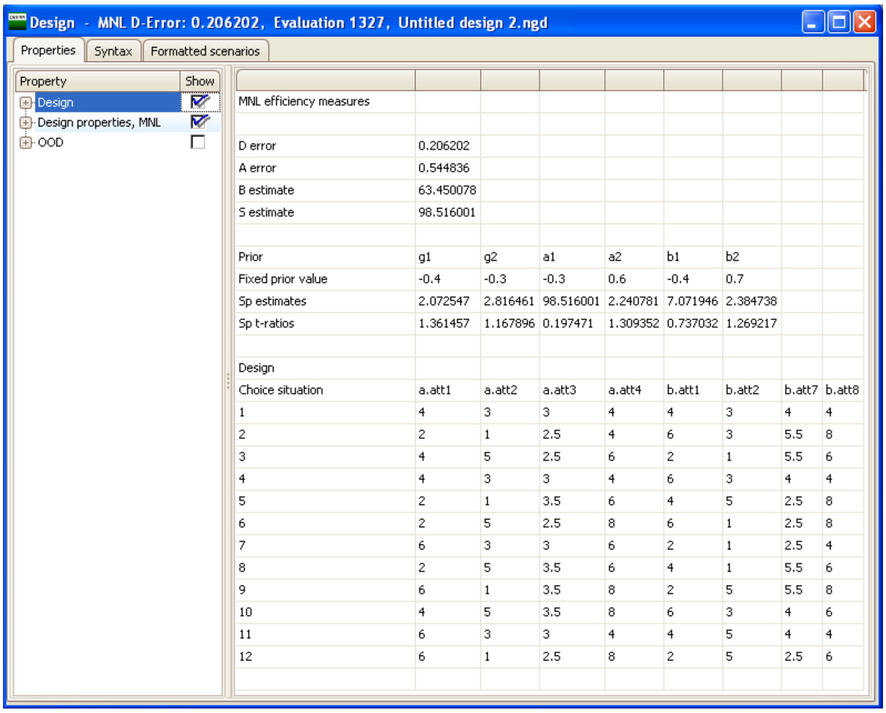
**The tree structure**: On the left is a tree structure that provides a list of properties of the design that can be reported. Related properties are grouped together, and the actual properties available will vary depending on the syntax of the design. The tree structure can be expanded and collapsed by clicking on the plus (+) and minus (-) symbols to the left of the tree. A property is selected for viewing by selecting its corresponding check box in the 'Show' column. Any number of properties can be selected for viewing.

**The output grid**: On the right is a grid that reports each of the selected properties. The properties themselves are typically tables, and the grid will adjust its size to accommodate all selected properties. The properties are listed in the grid in the order they were selected. Values in the grid are read-only and cannot be edited. However, the values may be copied and pasted into other software packages such as Microsoft Excel or Microsoft Word.

When a design is opened, certain properties are selected by default. The default selection will vary according to the syntax.

Many properties are calculated on the fly when they are selected. This prevents needless calculation of properties at earlier stages, say during a search. However, it does mean that some properties may be slow to display once selected. Properties that are known to frequently be slow to be calculated will have '(slow)' listed next to their name.

This section will not describe the actual available properties, or the corresponding outputs displayed in the grid. These outputs will be described in later chapters where appropriate. However, it is worth noting that the design matrix will always be available (the first property listed below), as will that design's correlation structure.

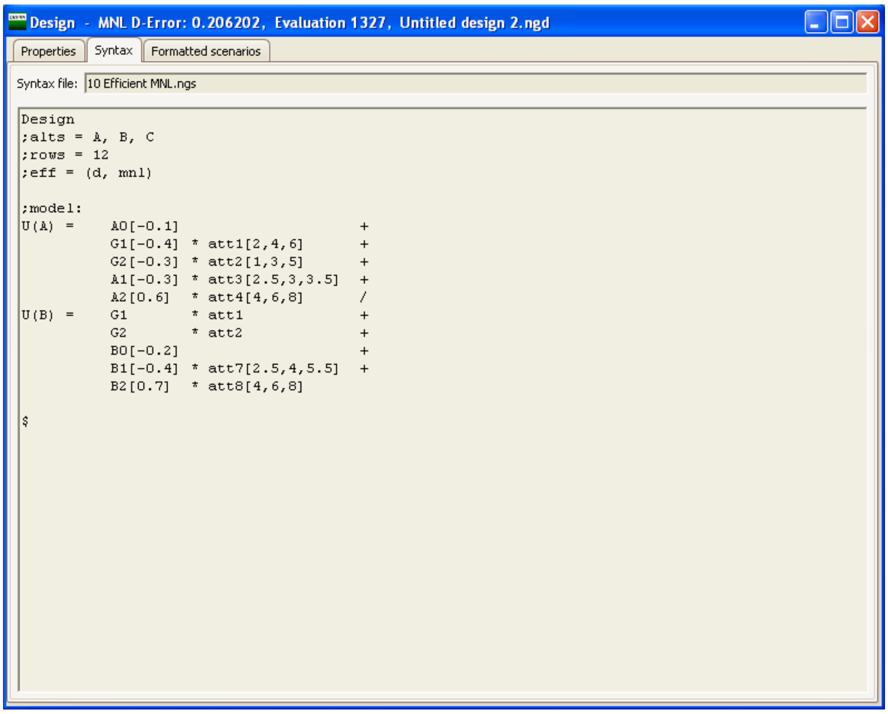


| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| MNL efficiency measures | | | | | | | | |
| | | | | | | | | |
| D error | 0.206202 | | | | | | | |
| A error | 0.544836 | | | | | | | |
| B estimate | 63.450078 | | | | | | | |
| S estimate | 98.516001 | | | | | | | |
| | | | | | | | | |
| Prior | g1 | g2 | a1 | a2 | b1 | b2 | | |
| Fixed prior value | -0.4 | -0.3 | -0.3 | 0.6 | -0.4 | 0.7 | | |
| Sp estimates | 2.072547 | 2.816461 | 98.516001 | 2.240781 | 7.071946 | 2.384738 | | |
| Sp t-ratios | 1.361457 | 1.167896 | 0.197471 | 1.309352 | 0.737032 | 1.269217 | | |
| | | | | | | | | |
| Design | | | | | | | | |
| Choice situation | a.att1 | a.att2 | a.att3 | a.att4 | b.att1 | b.att2 | b.att7 | b.att8 |
| 1 | 4 | 3 | 3 | 4 | 4 | 3 | 4 | 4 |
| 2 | 2 | 1 | 2.5 | 4 | 6 | 3 | 5.5 | 8 |
| 3 | 4 | 5 | 2.5 | 6 | 2 | 1 | 5.5 | 6 |
| 4 | 4 | 3 | 3 | 4 | 6 | 3 | 4 | 4 |
| 5 | 2 | 1 | 3.5 | 6 | 4 | 5 | 2.5 | 8 |
| 6 | 2 | 5 | 2.5 | 8 | 6 | 1 | 2.5 | 8 |
| 7 | 6 | 3 | 3 | 6 | 2 | 1 | 2.5 | 4 |
| 8 | 2 | 5 | 3.5 | 6 | 4 | 1 | 5.5 | 6 |
| 9 | 6 | 1 | 3.5 | 8 | 2 | 5 | 5.5 | 8 |
| 10 | 4 | 5 | 3.5 | 8 | 6 | 3 | 4 | 6 |
| 11 | 6 | 3 | 3 | 4 | 4 | 5 | 4 | 4 |
| 12 | 6 | 1 | 2.5 | 8 | 2 | 5 | 2.5 | 6 |

**Properties tab of the Design window**

## Syntax tab

The Syntax tab, shown below, displays a read-only copy of the syntax that was used to generate the design, in addition to the name of the syntax file that was run to generate the design.



Design - MNL D-Error: 0.206202, Evaluation 1327, Untitled design 2.ngd
Properties
Syntax
Formatted scenarios

Syntax file: 10 Efficient MNL.ngs

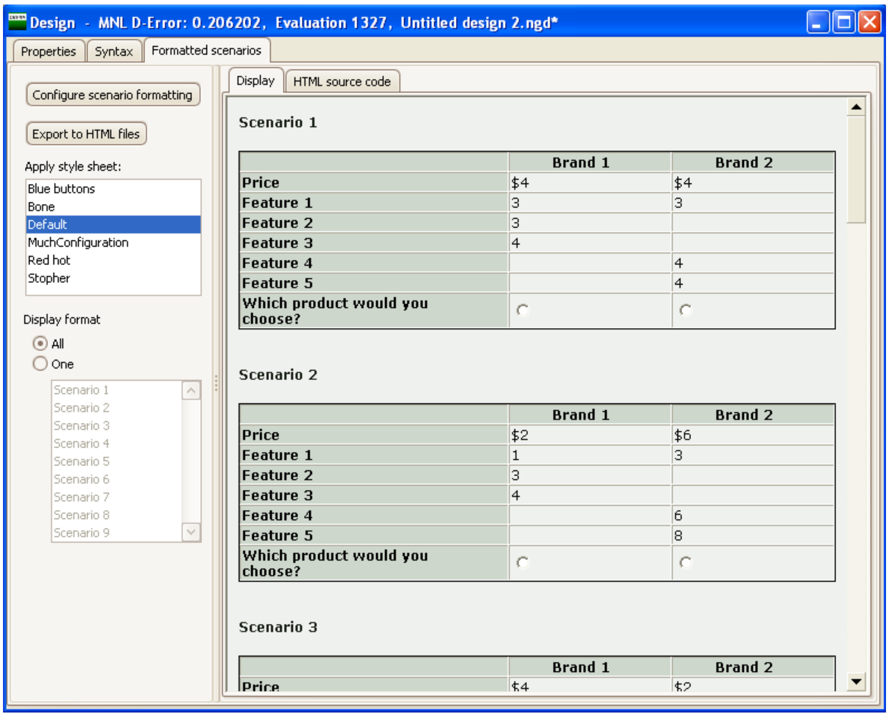
```
Design
;alts = A, B, C
;rows = 12
;eff = (d, mnl)

;model:
U(A) =    A0[-0.1]                  +
          G1[-0.4]  * att1[2,4,6]       +
          G2[-0.3]  * att2[1,3,5]       +
          A1[-0.3]  * att3[2.5,3,3.5]   +
          A2[0.6]   * att4[4,6,8]       /
U(B) =    G1        * att1              +
          G2        * att2              +
          B0[-0.2]                      +
          B1[-0.4]  * att7[2.5,4,5.5]   +
          B2[0.7]   * att8[4,6,8]

$
```

**Syntax tab of the Design window**

## Formatted scenarios tab

In addition to reporting the levels and various properties of the design, Ngene provides a mechanism, known as scenario formatting, for applying extensive formatting to the design. The results are presented in HTML, and the style of output can be rapidly transformed using cascading style sheets (CSS files). Choice matrices of any size can be generated and populated with arbitrary text, radio buttons for capturing choice, and design levels, which can themselves be formatted and transformed into labels. The formatting functionality is extensive, and is described in Chapter 10 "Formatting Experiments".

Scenario formatting is accessed through the Formatted scenarios tab of the Design window. A simple example is shown below.



Apply style sheet: Default

Display format: ● All

Scenario 1

| | Brand 1 | Brand 2 |
|---|---|---|
| Price | $4 | $4 |
| Feature 1 | 3 | 3 |
| Feature 2 | 3 | |
| Feature 3 | 4 | |
| Feature 4 | | 4 |
| Feature 5 | | 4 |
| Which product would you choose? | ○ | ○ |

Scenario 2

| | Brand 1 | Brand 2 |
|---|---|---|
| Price | $2 | $6 |
| Feature 1 | 1 | 3 |
| Feature 2 | 3 | |
| Feature 3 | 4 | |
| Feature 4 | | 6 |
| Feature 5 | | 8 |
| Which product would you choose? | ○ | ○ |

Scenario 3

| | Brand 1 | Brand 2 |
|---|---|---|
| Price | $4 | $2 |

**Formatted scenarios tab of the Design window**

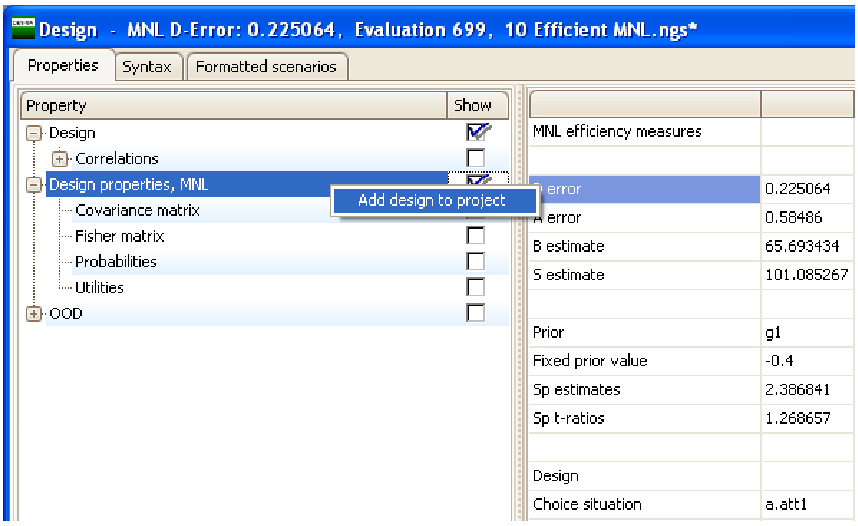
## Design files

Ngene designs are stored as Ngene design files, which have a .ngd suffix. These are plain text files, and can be examined by the curious. However, care must be taken if modifying an Ngene design file yourself, as changes may cause problems when the file is opened again in Ngene. The Ngene design file is registered in Windows on installation so that they will open in Ngene by default. The File menu section describes how design files can be opened and saved.

It is worth noting that .ngd files do not store all possible properties of the design. Instead, the file only contains the syntax used to generate the file, and the design levels. When the design is opened, the syntax is parsed and used to evaluate the stored design, making all properties available. This has several advantages, including a small file size and the ability for future versions of Ngene to report additional properties. The key disadvantage is that opening the .ngd file may be a little slow for some complex designs.

### Adding designs to a project

In addition to saving designs in isolation, designs can be added to a project that is currently open. From the Design window, right click anywhere on the Properties tab, and select 'Add design to project' from the popup menu, as shown below. Alternatively, designs can be added to the project from the iteration history list of the Output window.


Design - MNL D-Error: 0.225064, Evaluation 699, 10 Efficient MNL.ngs*
Properties
Syntax
Formatted scenarios
Add design to project

| Property | Show |
|---|---|
| Design | ☑ |
| Correlations | □ |
| Design properties, MNL | ☑ |
| Covariance matrix | |
| Fisher matrix | □ |
| Probabilities | □ |
| Utilities | □ |
| OOD | □ |

| | |
|---|---|
| MNL efficiency measures | |
| | |
| error | 0.225064 |
| A error | 0.58486 |
| B estimate | 65.693434 |
| S estimate | 101.085267 |
| | |
| Prior | g1 |
| Fixed prior value | -0.4 |
| Sp estimates | 2.386841 |
| Sp t-ratios | 1.268657 |
| | |
| Design | |
| Choice situation | a.att1 |

**Adding the design to the current project**

## 3.6 The unmanaged workspace

When a project is not open, Ngene operates with an unmanaged workspace. The unmanaged workspace treats files in the following way:

- New syntax files are only stored after they are explicitly saved for the first time.
- Design windows that have been opened from the Output window are only stored after they are explicitly saved for the first time.
- When syntax files, design files and data files are opened, the original version of the files are used.
- Designs cannot be added to a project, as no project is open.

The unmanaged workspace can be useful in the following situations:

- Only a small number of files are to be used.
- A design file needs to be interrogated.
- A simple search needs to be performed.

However, an unmanaged workspace with many windows can quickly become unwieldy, and in this case the user may wish to consider moving to a managed workspace by creating a project.

## 3.7 Projects: the managed workspace

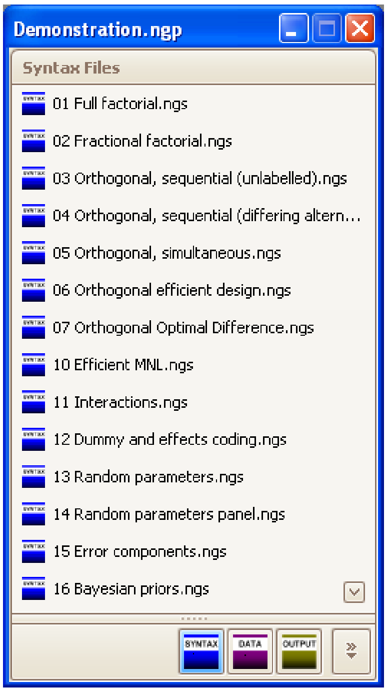
When a project is open, Ngene operates with a managed workspace. The managed workspace treats files in the following way:

- New syntax files are automatically stored in the project's folder when they are created.
- Design windows that have been opened from the Output window can be added to the project and thus stored in the project's folder. However, they are not automatically added to the project when they are opened.
- When syntax files, design files and data files are opened from outside the project's folder, a copy of the file is made to the project's folder and this copy is used by the project.

The unmanaged workspace can be useful when many files are required by the user. Once the project file is created, the user does not need to worry about where the project's files are stored. Only a single project file needs to be opened to resume from where the user left off in the previous session.

### The project window

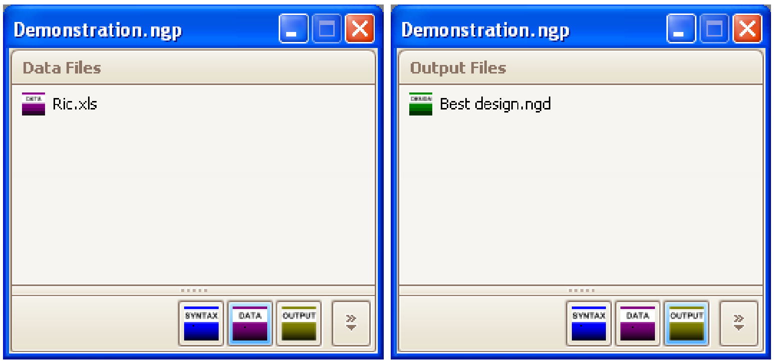
The project window (shown below) groups files in three categories: Syntax, Data and Output (the later contains design files). Each group has its own tab, which can be changed by selecting the appropriate button at the bottom of the project window.


Demonstration.ngp

Syntax Files

01 Full factorial.ngs
02 Fractional factorial.ngs
03 Orthogonal, sequential (unlabelled).ngs
04 Orthogonal, sequential (differing altern...
05 Orthogonal, simultaneous.ngs
06 Orthogonal efficient design.ngs
07 Orthogonal Optimal Difference.ngs
10 Efficient MNL.ngs
11 Interactions.ngs
12 Dummy and effects coding.ngs
13 Random parameters.ngs
14 Random parameters panel.ngs
15 Error components.ngs
16 Bayesian priors.ngs

**The Syntax Files tab of the project window**



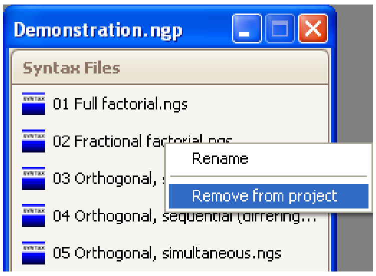
**The Data and Output tabs of the project window**

To open any file in the project as a window, click the file once in the Project window. The window will open, or if it was already open it will come into focus. The window will maximize if it was minimized.

## Managing files in a project

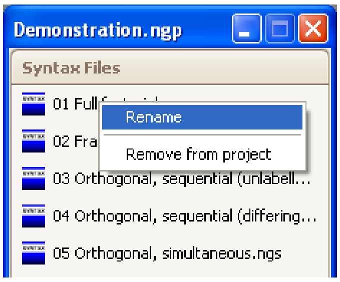
Files are added to the project whenever a file is created or opened. Additionally, the created or opened file is always stored in the project's folder.

To remove a file from a project, right click on the file in the project window, and select 'Remove from project' (see below). The file will not be deleted, but instead placed inside a subfolder of the project folder, called 'Removed Items'. You may wish to delete the file yourself from this folder, especially if it is a large data file.


Demonstration.ngp
Syntax Files
01 Full factorial.ngs
02 Fractional factorial.ngs
03 Orthogonal, s
04 Orthogonal, sequential (differing...
05 Orthogonal, simultaneous.ngs
Rename
Remove from project

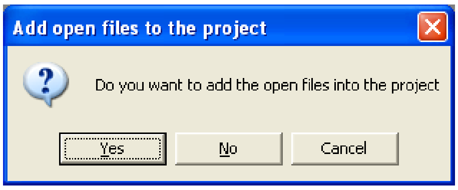
**Removing a file from a project**

Files in a project can be renamed within Ngene. Right click on the file you wish to rename in the project window, and select 'Rename' (see below). Enter the new file name in the dialog box that appears (see below).


Demonstration.ngp
Syntax Files
01 Full
02 Fra
Rename
Remove from project
03 Orthogonal, sequential (unlabell...
04 Orthogonal, sequential (differing...
05 Orthogonal, simultaneous.ngs

**Renaming a file in a project**

If you open or create a new project from an unmanaged workspace that contains open files, Ngene will ask you if you wish to move copies of the open files into the opened or new project (see below). This is particularly useful if, say, your unmanaged workspace is getting too complex and you want to consolidate all the files into a single project.


Add open files to the project

Do you want to add the open files into the project

**Option to add open files to opened or new project**

### The project file and its associated folder

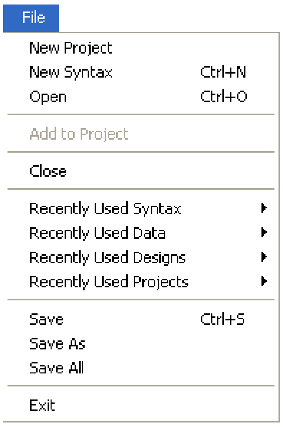
Project files end in the suffix .ngp. However, since projects can contain many files, Ngene creates a folder to store these in the same directory as the .ngp file. If the project file is called 'X.ngp', the associated folder will be called 'X project files'. If you copy a project to a different location, be sure to copy both the .ngp file and the associated folder in its entirety.

The .ngp file is registered in Windows on installation so that it will open in Ngene by default.

## 3.8 Menus

The following sections outline the options available from each of the menus in Ngene.

### 3.8.1 File menu



**The file menu**

#### New Project

Creates a new, empty project.

➢If the current project is unsaved or unsaved files are open, you will be asked if you wish to save them.
➢If the workspace is currently unmanaged but files are open, you will be asked if you wish to move copies of the files to the new project.
➢If syntax is executing, you will be asked if you wish to stop the execution.

#### New Syntax

Creates a new, blank syntax window.

➢If the workspace is unmanaged, the syntax file will not be stored until it is saved for the first time.
➢If a project is open, you will be asked for a file name, and the new syntax file will be listed in the project and stored in the project's folder.

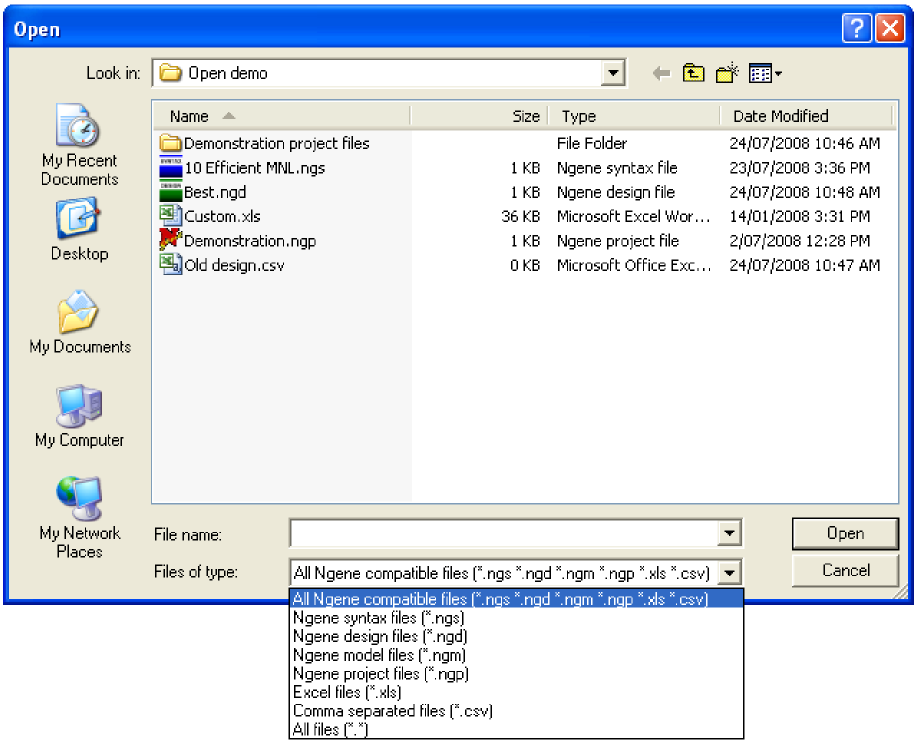
#### Open

Opens any of the following file types in Ngene:

- Ngene syntax files (.ngs)
- Ngene design files (.ngd)
- Ngene project files (.ngp)
- Excel files (.xls), read only
- Comma separated files (.csv), read only

➢If the workspace is unmanaged and you open a syntax, design, Excel or CSV file, the file will be opened from its original location.
➢If a project is open and you open a syntax, design, Excel or CSV file, the file will be copied to the project's folder and that copy will be opened.
➢If the workspace is unmanaged with open files, and you open a project, you will be asked if you wish to move copies of the files to the project that is being opened.
➢If the existing (managed or unmanaged) workspace contains unsaved files and you open another project, you will be asked if you wish to save them.

➢If you open a project while syntax is running, you will be asked if you wish to stop the run.



Look in: Open demo

| Name | Size | Type | Date Modified |
|---|---|---|---|
| Demonstration project files | | File Folder | 24/07/2008 10:46 AM |
| 10 Efficient MNL.ngs | 1 KB | Ngene syntax file | 23/07/2008 3:36 PM |
| Best.ngd | 1 KB | Ngene design file | 24/07/2008 10:48 AM |
| Custom.xls | 36 KB | Microsoft Excel Wor... | 14/01/2008 3:31 PM |
| Demonstration.ngp | 1 KB | Ngene project file | 2/07/2008 12:28 PM |
| Old design.csv | 0 KB | Microsoft Office Exc... | 24/07/2008 10:47 AM |

File name:

Files of type: All Ngene compatible files (*.ngs *.ngd *.ngm *.ngp *.xls *.csv)

All Ngene compatible files (*.ngs *.ngd *.ngm *.ngp *.xls *.csv)
Ngene syntax files (*.ngs)
Ngene design files (*.ngd)
Ngene model files (*.ngm)
Ngene project files (*.ngp)
Excel files (*.xls)
Comma separated files (*.csv)
All files (*.*)

**The Open dialog box**

## Recently Used Syntax
## Recently Used Data
## Recently Used Designs
## Recently Used Projects

The most recently opened syntax, data files, design files and projects will be listed, and may be opened quickly with this submenu. The toolbar options open the most recent project, syntax, and data files. An error message will be displayed if the file selected no longer exists.

## Close

Closes the active window.

➢If the project window is closed, but the project contains unsaved files, you will be asked if you wish to save them.
➢If syntax is running, you will be asked if you wish to stop the run.

## Save

Saves the active window in its current location.

➢If the file has not yet been saved, a save location will be requested, as per Save As.

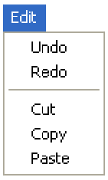
### Save As

Saves a copy of the active window in the location you specify.

### Exit

Exits Ngene. If you are running syntax, you will be asked if you wish to stop the current run. You may also be asked if you wish to save any unsaved syntax files or projects.

## 3.8.2 Edit menu



**The edit menu**

### Undo

Undo the last syntax window modification. Undo only works with syntax modifications - no other actions can be undone.

### Redo

Redo the last undone syntax window modification. Redo only works with syntax modifications - no other actions can be redone.

### Cut

Cut the selected text.

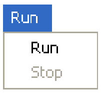
### Copy

Copy the selected text.

### Paste

Paste the selected text.

## 3.8.3 Run menu



**The run menu**

### Run ( / Pause / Resume )

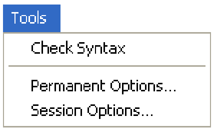
The Run menu item starts the syntax run. Run can only be selected when a syntax window is active. If any text is selected in the syntax window, only the selected text is run. If no text is selected, the first command is run.

While the syntax is being run, the Run menu item changes to Pause. If Pause is selected, the syntax run halts temporarily, and this menu item changes to Resume. If Resume is selected, the syntax run resumes. A syntax run can be paused and resumed any number of times.

### Stop

Many syntax runs will execute for a long time or indefinitely. A syntax run can be stopped at any stage by selecting the Stop menu item. Stop can only be selected if syntax is currently being run.

## 3.8.4 Tools menu



**The tools menu**

### Check Syntax

Rather than run a piece of syntax, you may wish to just check that the syntax is valid. When a syntax window is active, select this option and any syntax errors will be reported in the Output window.

### Permanent Options...
### Session Options...

There exist a number of settings and defaults in Ngene that may be changed by the user. Ngene allows users to change the defaults in two different ways.

The **Permanent Options** dialog box allows the user to make permanent changes to the various settings which will be saved and retained across sessions.

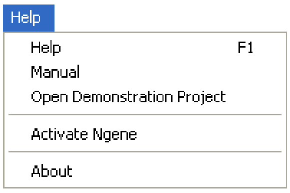
The **Session Options** dialog box allows the user to make changes to the various settings that will remain in effect only for a particular session. The settings will be retained until such time as the user further changes the default values or until the program is closed.

Refer to the Options dialog box section for details of the actual settings that can be configured.

### 3.8.5 Window menu

The window menu allows you to navigate between all open windows.

### 3.8.6 Help menu



**The Help menu**

#### Help

Opens the documentation in a Compiled HTML Help (CHM) file.

#### Manual

Opens the documentation as a single Acrobat (PDF) file.

#### Open demonstration project

Opens a copy of a demonstration project containing a collection of example syntax files. The user is prompted to select a location where the copy will be stored.

#### Activate Ngene

Allows you to activate Ngene and unlock the full version. For more information refer to License activation and management.

#### About

Provides specific information on the current installation of Ngene, including the specific version and build number. If you are reporting a bug or problem on the website, please quote your version and build number.

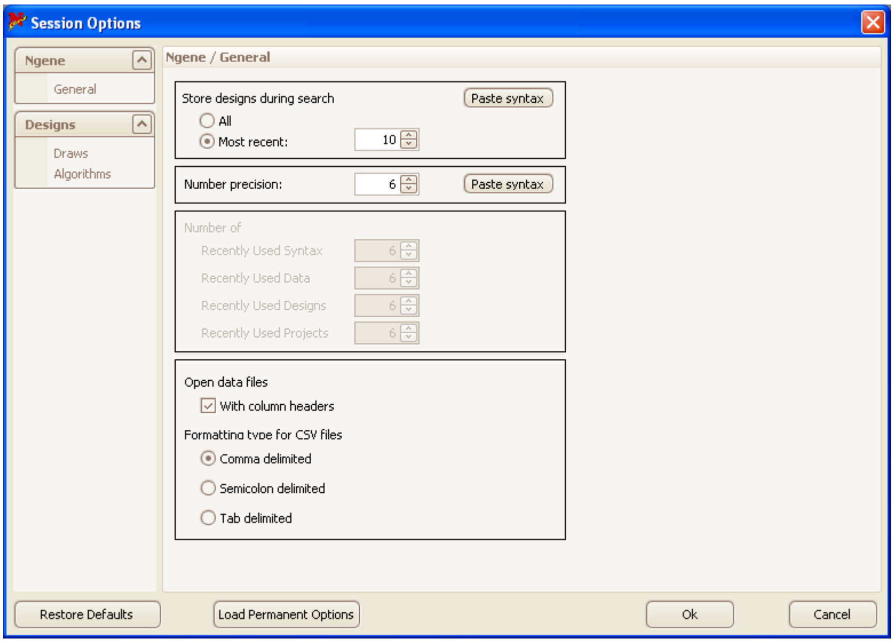
### 3.8.7 Options dialog box

There exist a number of settings and defaults in Ngene that may be changed by the user. Ngene allows users to change the defaults in two different ways via the two options dialog boxes located in the Tools menu. The Permanent Options dialog box allows the user to make permanent changes to the various settings which will be saved and retained across sessions. The Session Options dialog box allows the user to make changes to the various settings that will remain in effect only for a particular session. The settings will be retained until such time as the user further changes the default values or until the program is closed. The only differences between the two dialog boxes are the titles and slight differences to the functionality of the Load Options button. All screenshots will use the Session Options dialog box, but the actual settings options will be identical.

The screenshot below shows the basic layout of the Options dialog box. On the left are links to various pages (e.g. Draws and Algorithms), which are grouped under a heading (e.g. Designs). Selecting one of these links will load the corresponding screen on the right. The 'Restore Defaults' button will populate all settings with Ngene's 'factory' defaults. The 'Load Options' button varies between the two settings dialog boxes. In the Session Options dialog box, the 'Load Permanent Options' button will populate the session options with the current permanent options. In the Permanent Options dialog box, this same button is called 'Load Session Options', and will populate the permanent options with the current session options. Save will apply all changes to the settings and close the dialog box, while cancel will close the dialog box without making any changes.

Settings are described below, grouped by the page they appear on.

## General tab



Ngene / General

Store designs during search
○ All
● Most recent: 10

Number precision: 6

Number of
Recently Used Syntax 6
Recently Used Data 6
Recently Used Designs 6
Recently Used Projects 6

Open data files
☑ With column headers

Formatting type for CSV files
● Comma delimited
○ Semicolon delimited
○ Tab delimited

**The General tab of the Options dialog box**

### Store designs during search
### (default = 10)

When generating output, Ngene attempts to save as much output as possible, thus allowing the user to see how the output changes over different iterations. For example, in generating efficient designs (see Chapter 7), multiple designs are generated and tested. If a design is found to be more efficient, Ngene will store and save that design. Rather than throw away previously stored designs, Ngene allows the user to store these as well. In this way, the user may view ‘less efficient’ designs for purposes of comparison. Indeed, the user may for other reasons decide to use a less efficient design if so desired. Storing large numbers of designs may result in significant memory issues, particularly for some advanced designs. For this reason, Ngene allows the user to change the number of most recent designs that are stored. The first design in a search will always be stored. It is also possible to allow all designs to be retained, but the user must accept the risk of memory issues.

### Number precision
### (default = 6)

This settings allows the user to modify the number of decimal places reported. While calculations are made with maximum precision internally, large numbers of decimal places can be unwieldy when reported in the output.

**Number of Recently Used Syntax**
**Number of Recently Used Data**
**Number of Recently Used Designs**
**Number of Recently Used Projects**
**(default = 6)**

This alters the maximum number of most recently used syntax/data/designs/projects available in the File menu.

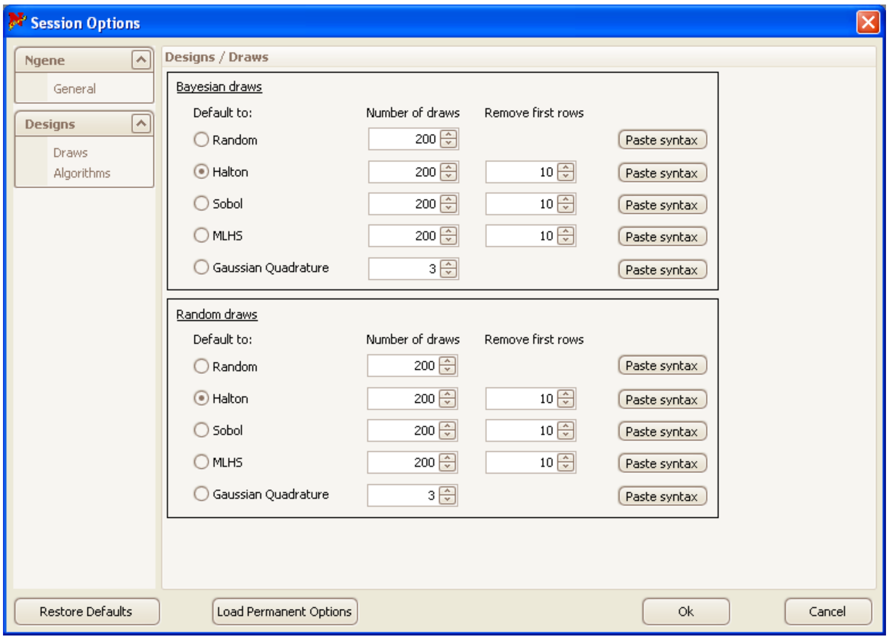
### Open data files with column headers

Change whether Ngene will look for a header row when opening a data file (checked), or start reading the data from the first row (unchecked). All data files will be opened using this setting.

### Formatting type for CSV files

Specify what character is used to delineate cells when opening CSV files: commas, semicolons, or tab characters. All CSV data files will be opened using this setting.

## Draws tab



Session Options

Designs / Draws

Bayesian draws

| Default to: | Number of draws | Remove first rows |
|---|---|---|
| ○ Random | 200 | |
| ● Halton | 200 | 10 |
| ○ Sobol | 200 | 10 |
| ○ MLHS | 200 | 10 |
| ○ Gaussian Quadrature | 3 | |

Random draws

| Default to: | Number of draws | Remove first rows |
|---|---|---|
| ○ Random | 200 | |
| ● Halton | 200 | 10 |
| ○ Sobol | 200 | 10 |
| ○ MLHS | 200 | 10 |
| ○ Gaussian Quadrature | 3 | |

**The Draws tab of the Options dialog box**

**Default to**
**(default = Halton)**

Sets the default type of draws to use for either Bayesian or Random draws.

**Number of draws**
**(default = 200)**

Sets the default number of draws for each draw type.

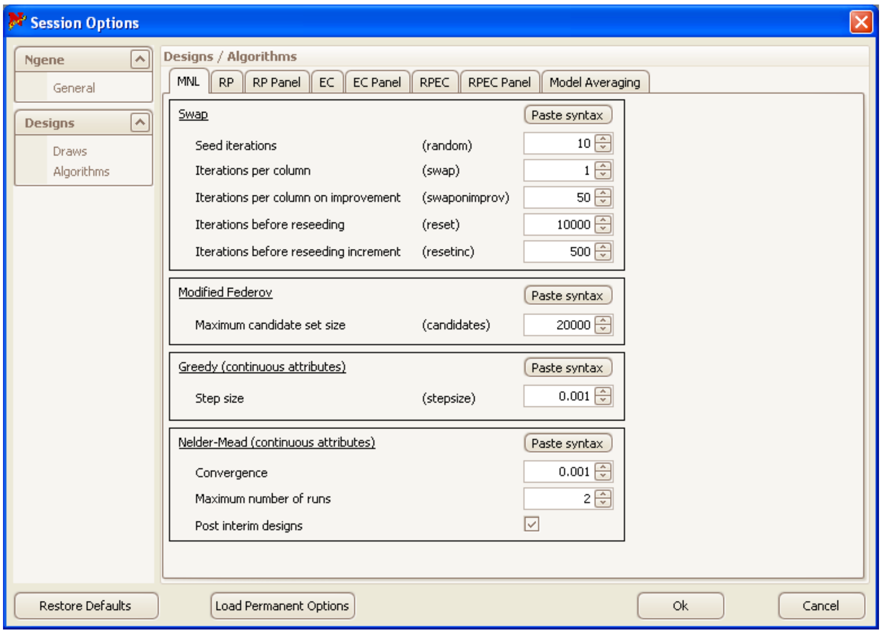
In most software packages that use simulated draws the default number of simulated draws is fixed. In Ngene, the user is able to change the default number of draws for each draw type. For functions that require the use of simulated draws (for example, Bayesian efficient designs), if the user fails to specify the number of draws, Ngene will use the default number of draws specified here.

**Remove first rows**
**(default = 10)**

Specifies how many initial rows to remove from the table used for the corresponding draw type.

Many types of draws, often referred to as intelligent or quasi random Monte Carlo draws (e.g., Halton sequences) are nothing more than tables of generated probabilities. These types of draws are constructed in a specific fashion so that as much space of a distribution will be covered. Many researchers have questioned certain aspects of these tables, in particularly Tables of Halton sequences. In particular, these researchers claim that the first few rows of tables (corresponding to the first few simulated draws) are correlated in an undesirable way (see Train 2003 for example). These researchers therefore suggest removing the first few simulated draws.

## Algorithms tab



Swap

| | | |
|---|---|---|
| Seed iterations | (random) | 10 |
| Iterations per column | (swap) | 1 |
| Iterations per column on improvement | (swaponimprov) | 50 |
| Iterations before reseeding | (reset) | 10000 |
| Iterations before reseeding increment | (resetinc) | 500 |

Modified Federov

| | | |
|---|---|---|
| Maximum candidate set size | (candidates) | 20000 |

Greedy (continuous attributes)

| | | |
|---|---|---|
| Step size | (stepsize) | 0.001 |

Nelder-Mead (continuous attributes)

| | | |
|---|---|---|
| Convergence | | 0.001 |
| Maximum number of runs | | 2 |
| Post interim designs | | ☑ |

**The Algorithms tab of the Options dialog box**

The generation of efficient experimental designs requires the exploration of impact of different attribute level combinations. How Ngene changes the attribute level combinations may be set by the user. Different algorithms (discussed in Algorithms for generating designs in Ngene) are available to the user. Most algorithms have several settings, which can be controlled through parameters specified in the 'alg' property. If no parameters are specified, the defaults specified here in the Options dialog box are used. The equivalent parameter names that can be supplied in the 'alg' property are listed in brackets next to the description.

When generating efficient designs, the type of model used to calculate the efficiency can have a large impact on performance, and this may be a consideration when setting algorithm parameters. For example, RP panel calculations are relatively slow, and so it may not be appropriate to allow as many seed iterations for RP panel designs. The algorithm parameter defaults can be set independently for each type of model by first choosing the appropriate tab. The model averaging tab applies whenever the 'eff' property specifies more than one model type to optimize on (See model averaging for more details).

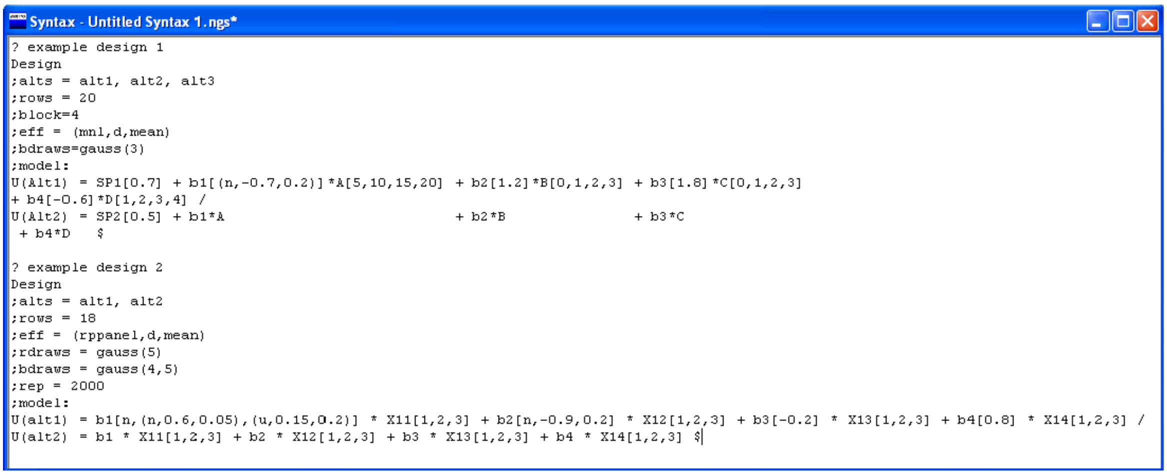
# Chapter 4

# Ngene Syntax

# 4 Ngene Syntax

## 4.1 Syntax command format

Most Ngene instructions follow a similar pattern. Each new routine must begin on a new line (see Figure 4.1), however specific instructions within a routine may use the same line. A routine may consist of as many lines as required. Ngene code is not case specific so that the user may freely use lower or capital case letters, and spaces may be used throughout the code.



```
? example design 1
Design
;alts = alt1, alt2, alt3
;rows = 20
;block=4
;eff = (mnl,d,mean)
;bdraws=gauss(3)
;model:
U(Alt1) = SP1[0.7] + b1[(n,-0.7,0.2)]*A[5,10,15,20]  + b2[1.2]*B[0,1,2,3]  + b3[1.8]*C[0,1,2,3]
+ b4[-0.6]*D[1,2,3,4] /
U(Alt2) = SP2[0.5] + b1*A                            + b2*B                + b3*C
 + b4*D   $

? example design 2
Design
;alts = alt1, alt2
;rows = 18
;eff = (rppanel,d,mean)
;rdraws = gauss(5)
;bdraws = gauss(4,5)
;rep = 2000
;model:
U(alt1) = b1[n,(n,0.6,0.05),(u,0.15,0.2)] * X11[1,2,3] + b2[n,-0.9,0.2] * X12[1,2,3] + b3[-0.2] * X13[1,2,3] + b4[0.8] * X14[1,2,3] /
U(alt2) = b1 * X11[1,2,3] + b2 * X12[1,2,3] + b3 * X13[1,2,3] + b4 * X14[1,2,3] $
```

**Figure 4.1: Sample syntax**

The general format of a command is:

```
VERB
; other information …
$
```

The syntax for a routine in Ngene will always begin with a verb and end with a dollar sign ‘$’. Specific properties related to the routine must usually be specified, and each begins with a semicolon ';'. For example, the typical structure of the Design command is as follows, which starts with ‘Design’, then sets properties, and closes with the dollar symbol ‘$’:

```
Design
;<property>
;<property>
;...
? comment
$
```

Comments in the syntax file can be indicated with a question mark symbol ‘?’ and all text subsequent to that symbol on the same line will be ignored.

The order of the properties or routine instructions does not matter.

Several routines may be run in sequence, by typing several routines into a single syntax file and pressing run, or highlighting several routines and pressing run. However, the generation of many types of designs will run indefinitely, even if an improved design is unlikely to be found after a certain period of time. In that case, you can specify stopping criteria. Refer to the syntax reference for the alg property for more details.

## 4.2 An example design syntax: Full factorial designs

In the following we will explain a simple syntax file as an introduction to the basic syntax structure. A list and description of all propertys and properties can be found in the Syntax Reference.

Several types of designs can be created using the Design property. One such design is the full factorial design, which uses only the most basic properties.

For designs, three properties will always be present in a syntax file, namely *alts*, *rows*, and *model*. The *alts* property defines which alternatives are present in the choice model. The *rows* property defines how many choice situations need to be generated. The *model* property defines the choice model by describing the complete utility function for each alternative.

For example,

```
;alts = A, B, C
;alts = car, bus, train
;alts = house1, house2
```

The alternatives can have any name (except for some reserved words) and need to be separated by commas. These same names are then used in the *model* property, defining their utility functions. This model property is a complex property and will be described in more detail.

Suppose that the alternatives are named ‘alt1’ and ‘alt2’. An example of setting the model property is:

```
;model:
U(alt1) = b1 + b2 * A[0,1,2] + b3 * B[0,1]     /
U(alt2) =      b2 * A        + b4 * C[2,4,6,8]
```

First, notice that complex properties consist of several lines, separated by a slash ‘/’. The first line after the ‘model:’ property describes the utility function for ‘alt1’, the second line for ‘alt2’. The utility functions are expressed as linear functions of attributes with associated weighting parameters. In the above example, ‘b1’ to ‘b4’ are the weighting parameter names, while ‘A’ to ‘C’ are the attribute names. The first name in the multiplication is the parameter name, the second is the attribute name, i.e., ‘b2’ is the parameter associated with attribute ‘A’. Note that constants like ‘b1’ are specified without an associated attribute. In all cases, the parameter name precedes the attribute name, which are separated by the asterisk multiplicative symbol ‘*’.

Note that ‘b2’ appears both in the utility function of ‘alt1’ and ‘alt2’, meaning that ‘b2’ is a generic parameter across both alternatives. On the other hand, ‘b1’, ‘b3’ and ‘b4’ are alternative specific parameters. Whenever the same name is used across alternatives, the parameter is assumed to be generic.

The values between square brackets located after an attribute name are the possible attribute levels for that specific attribute required by the user. For example, attribute ‘A’ can have the levels 0, 1, or 2, while attribute ‘B’ can only have the levels 0 or 1. If the same levels are used for a similar attribute with the same name in another alternative, then it is not necessary to repeat the levels, such that in the example above, the levels of ‘A’ can be omitted in the second utility function. If the levels are different, then the attribute level values will need to be added. Note that one can use the same attribute name in different utility functions as Ngene will treat them separately (Ngene will refer to them in the output as ‘alt1.A’ and ‘alt2.A’, etc.).

The number of choice situations to be generated has to be defined using the *rows* property. Normally this would be a whole number, but to prevent the user from having to calculate the number of rows in the full factorial manually, the following can be specified:

```
;rows = all
```

Finally, to specify that we want a factorial design, we specify:

```
;fact
```

The complete syntax would be:

```
Design
? This will generate a full factorial design
;alts = alt1, alt2
;rows = all
;fact
;model:
U(alt1) = b1 + b2 * A[0,1,2] + b3 * B[0,1] /
U(alt2) =      b2 * A        + b4 * C[2,4,6,8]
$
```

The above example will generate a full factorial design with 3x2x3x4 = 72 choice situations.

Attribute levels can be specified in an alternative way, with a lower and upper bound, and a step size. These three values are specified in sequence inside the square brackets, separated by a colon. Using this syntax, the above example would be:

```
Design
? This will generate a full factorial design
;alts = alt1, alt2
;rows = all
;fact
;model:
U(alt1) = b1 + b2 * A[0:2:1] + b3 * B[0:1:1] /
U(alt2) =      b2 * A        + b4 * C[2:8:2]
$
```

# Chapter 5

# Introduction to Experimental Design Theory

# 5 Introduction to Experimental Design Theory

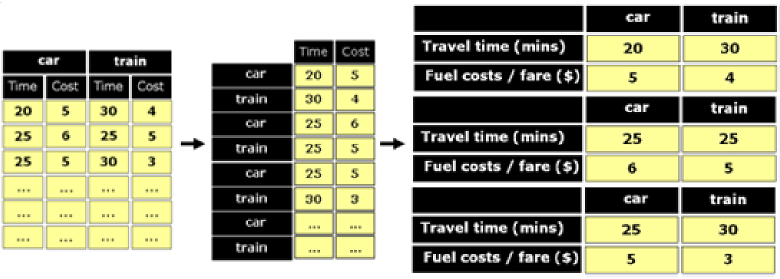
## 5.1 Introduction to experimental designs for stated choice experiments

SC (SC) experiments, as proposed by Louviere and Woodworth (1983) and Louviere and Hensher (1983), have received increasing attention in many different fields, including marketing, transportation, health economics, environmental economics, and resource economics. Theoretical advances in and estimation of discrete choice models has had a large impulse from the transportation community, where many state-of-the-art publications on this topic have appeared. In contrast, the main research in design of choice experiments has been in marketing and economics. Lately, the interest in the design of choice experiments has increased in the transportation field as well, and the purpose of this manual is to present the state-of-the-art in designing choice experiments using the knowledge gained over the years till present from all disciplines. While there exist good books with overviews for discrete choice modelling and estimation (Ben-Akiva and Lerman, 1985; Hensher et al., 2005; Louviere et al., 2000; Train, 2003), no such books exist for designing SC experiments.

The purpose behind conducting experiments is to determine the independent influence of different variables (attributes or factors depending on the literature cited) on some observed outcome. In SC studies, this translates into the desire to determine the influence of the design attributes upon the choices that are observed to be made by sampled respondents undertaking the experiment. However, an acknowledged limitation of SC studies is that unless the number of person specific observations captured in a survey is extremely large, it is necessary to pool the responses obtained from multiple respondents in order to produce statistically reliable parameter estimates. As such, SC studies typically consist of numerous respondents being asked to complete a number of choice tasks in which they are asked to select one or more alternatives from amongst a finite set of alternatives. In each task, the alternatives, whether labeled or unlabeled[1], are typically defined on a number of different attribute dimensions, each of which are further described by pre-specified levels drawn from some underlying *experimental design*. The number of choice tasks each respondent is asked to undertake will generally be up to the total number of choice situations drawn from the experimental design. Consequently, an archetypal SC experiment might require choice data be collected on 200 respondents, each of whom are observed to make eight choices, thus producing a total of 1600 choice observations.

Exactly how analysts distribute the levels of the design attributes over the course of an experiment (which typically is via the underlying experimental design), may play a big part in whether or not an independent assessment of each attribute's contribution to the choices observed to have been made by sampled respondents can be determined. Further, the allocation of the attribute levels within the experimental design may also impact upon the statistical power of the experiment insofar as its ability to detect statistical relationships that may exist within the data. This ability is related to the sample size of the study and given a large enough sample, the statistical power of an experimental design may not matter. Nevertheless, for sample sizes more commonly used in practice, the ability to retrieve statistically significant parameter estimates may be compromised given the selection of a relatively poor design. What constitutes a poor design is the focus of this chapter, however, at this stage it may be worth noting that there may exist a trade-off between the ability of a design to allow for an independent determination of the impact each design attribute has in a SC experiment (at least insofar as how independence is thought of in a traditional sense) and the ability of the design to detect statistically significant relationships. The experimental design chosen by the analyst may therefore play a significant role in SC studies.

Conceptually, an experimental design may be viewed as nothing more than a matrix of values that is used to determine what goes where in a SC survey. The values that populate the matrix represent the attribute levels that will be used in the SC survey, whereas the columns and rows of the matrix represent the choice situations, attributes and alternatives of the experiment. The actual layout of the design matrix is often set out in one of two ways. Some researchers set up the experimental design matrix such that each row represents a different choice situation and each column a different attribute within the experiment (see e.g., Bliemer and Rose 2006; Rose and Bliemer 2008). In such cases, groups of columns form different alternatives within each choice task. Other researchers however set-up the design matrix such that each row of the matrix represents an individual alternative and each column a different attribute (see e.g., Carlsson and Martinsson 2002; Huber and Zwerina 1996; Kanninen 2002; Kessels et al. 2006; Sándor and Wedel 2001, 2002). In these cases, multiple rows are grouped together to form individual choice situations. Independent of how the matrix is set out, the experimental design performs the same function; that being the allocation of attribute levels to choice tasks, as shown in Figure 5.1.



| car | | train | |
|---|---|---|---|
| Time | Cost | Time | Cost |
| 20 | 5 | 30 | 4 |
| 25 | 6 | 25 | 5 |
| 25 | 5 | 30 | 3 |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| ... | ... | ... | ... |

| | Time | Cost |
|---|---|---|
| car | 20 | 5 |
| train | 30 | 4 |
| car | 25 | 6 |
| train | 25 | 5 |
| car | 25 | 5 |
| train | 30 | 3 |
| car | ... | ... |
| train | ... | ... |

| | car | train |
|---|---|---|
| Travel time (mins) | 20 | 30 |
| Fuel costs / fare ($) | 5 | 4 |

| | car | train |
|---|---|---|
| Travel time (mins) | 25 | 25 |
| Fuel costs / fare ($) | 6 | 5 |

| | car | train |
|---|---|---|
| Travel time (mins) | 25 | 30 |
| Fuel costs / fare ($) | 5 | 3 |

**Figure 5.1: From experimental design to choice situation construction**

A number of competing explanations exist as to why this distinction has arisen in the past. The first explanation suggests that the distinction arose due to historical reasons, with Western Europeans, led predominately by John Bates in the early 1980s, adopting the column based approach whilst the row based approach remained a legacy from the traditional conjoint methods used by marketing researchers elsewhere in the world. A second explanation is that the different design formats tend to correspond to the use of either equations to derive the asymptotic variance-covariance (AVC) matrix (representing the column based approach) or matrix algebra (corresponding to the row based approach). Independent of how the design matrix is represented however, the end result remains the same.

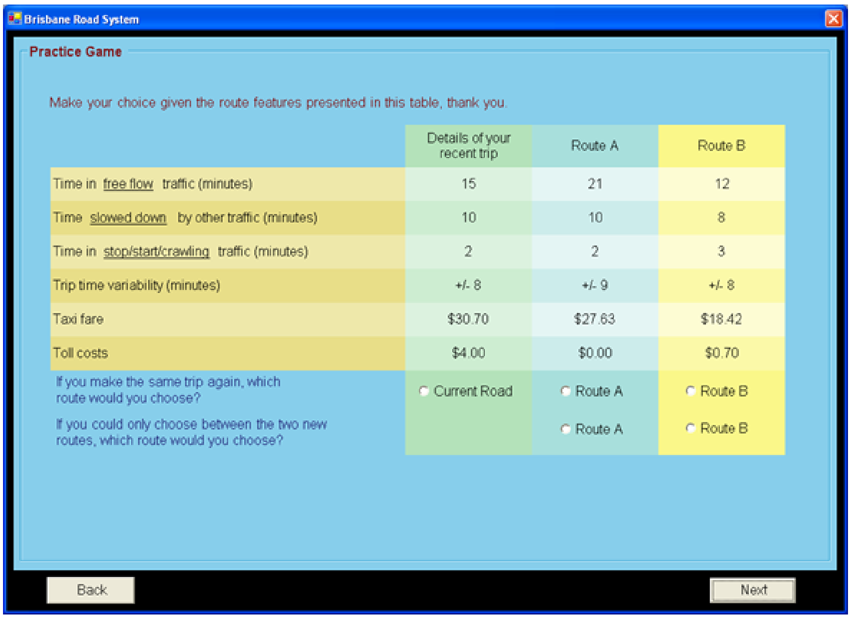
Given the above, the primary question for those generating experimental designs for SC studies is 'how best to allocate the attribute levels to the design matrix'. Traditionally, researchers have relied upon the use of orthogonal experimental designs to populate the hypothetical choice situations shown to respondents (see Louviere *et al*., 2000, for a review of orthogonal designs). More recently however, some researchers have begun to question the relevance of orthogonal designs when applied to SC experiments (e.g., Huber and Zwerina, 1996; Kanninen, 2002; Kessels et al., 2006; Sándor and Wedel, 2001, 2002, 2005). Generally, the argument against the use of orthogonality as a design criterion in the construction process is that the property of orthogonality is unrelated to the desirable properties of the econometric models used to analyse SC data (i.e., logit and probit models). The orthogonality (or otherwise) of an experimental design relates to the correlation structure between the attributes of the design with designs in which all between-attribute correlations are zero being said to be orthogonal (in some cases, this definition of an orthogonal design may be relaxed to define orthogonality as occurring when all attribute correlations are zero within alternatives but not necessarily between alternatives; see Louviere *et al*. (2000) discussion on sequential versus simultaneous generation of orthogonal designs). Whilst orthogonality is an important criterion to determine independent effects in linear models, discrete choice models are not linear (Train, 2003). In models of discrete choice, the correlation structure between the attributes is not what is of importance. Rather, given the derivation of the models, it is the correlations of the differences in the attributes which should be of concern.

Huber and Zwerina (1996) took the important step of relating the statistical properties of the SC experiments to the econometric models estimated on such data. In their paper, Huber and

Zwerina showed that designs that let go of orthogonality as a consideration in generating SC experiments and which attempt to reduce the asymptotic standard errors of the parameter estimates (i.e., the square roots of the diagonal elements of the asymptotic variance-covariance (AVC) matrix) will generally result in designs that either (i) improve the reliability of the parameters estimated from SC data at a fixed sample size or (ii) reduce the sample size required to produce a fixed level of reliability in the parameter estimates with a given experimental design. The linking of the experimental design generation process to attempts to reduce the asymptotic standard errors of the parameter estimates has resulted in a class of designs known as efficient or optimal designs, where designs that produce smaller asymptotic standard errors are thought of as being more efficient.

## 5.2 Overview of general steps for creating stated choice experiments

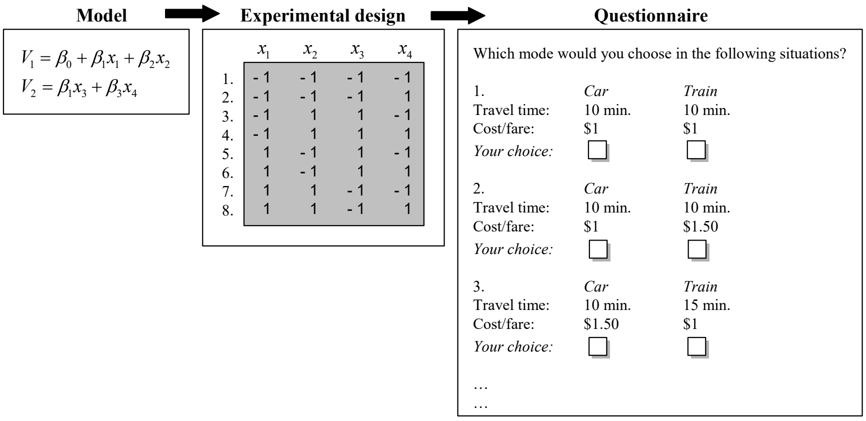
The aim of generating an experimental design is generally to help construct a SC experiment, for which an example is given in Figure 5.2. In creating a stated choice experiment, three main steps have to be taken, as illustrated in Figure 5.3. First of all, a complete model specification with all parameters to be estimated has to be determined. Based on this model specification, an experimental design type has to be selected and then the design can be generated. Finally, a questionnaire (on paper, internet, CAPI, etc.) is created based on the underlying experimental design and data can be collected. The three steps will be elaborated below. The main part of the chapter will be dedicated to the generation of experimental designs (step 2).



Make your choice given the route features presented in this table, thank you.

| | Details of your recent trip | Route A | Route B |
|---|---|---|---|
| Time in free flow traffic (minutes) | 15 | 21 | 12 |
| Time slowed down by other traffic (minutes) | 10 | 10 | 8 |
| Time in stop/start/crawling traffic (minutes) | 2 | 2 | 3 |
| Trip time variability (minutes) | +/- 8 | +/- 9 | +/- 8 |
| Taxi fare | $30.70 | $27.63 | $18.42 |
| Toll costs | $4.00 | $0.00 | $0.70 |
| If you make the same trip again, which route would you choose? | ○ Current Road | ○ Route A | ○ Route B |
| If you could only choose between the two new routes, which route would you choose? | | ○ Route A | ○ Route B |

Figure 5.2: Example of a screen in a stated choice experiment



Model

$$V_1 = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$
$$V_2 = \beta_1 x_3 + \beta_3 x_4$$

Experimental design

|  | $x_1$ | $x_2$ | $x_3$ | $x_4$ |
|---|---|---|---|---|
| 1. | -1 | -1 | -1 | -1 |
| 2. | -1 | -1 | -1 | 1 |
| 3. | -1 | 1 | 1 | -1 |
| 4. | -1 | 1 | 1 | 1 |
| 5. | 1 | -1 | 1 | -1 |
| 6. | 1 | -1 | 1 | 1 |
| 7. | 1 | 1 | -1 | -1 |
| 8. | 1 | 1 | -1 | 1 |

Questionnaire

Which mode would you choose in the following situations?

| 1. | *Car* | *Train* |
|---|---|---|
| Travel time: | 10 min. | 10 min. |
| Cost/fare: | $1 | $1 |
| *Your choice:* | □ | □ |
| 2. | *Car* | *Train* |
| Travel time: | 10 min. | 10 min. |
| Cost/fare: | $1 | $1.50 |
| *Your choice:* | □ | □ |
| 3. | *Car* | *Train* |
| Travel time: | 10 min. | 15 min. |
| Cost/fare: | $1.50 | $1 |
| *Your choice:* | □ | □ |

…
…

**Figure 5.3: Steps in designing a stated choice experiment**

## 5.2.1 Step 1 - Model specification

Each SC experiment is specifically created for estimating a specific model (or sometimes a range of models). Therefore, one needs to specify the model and the parameters to be estimated before creating an experimental design.

First, the problem studied should be refined and hypotheses developed. Secondary data search, focus groups, and in-depth interview can assist in this. Then the stimuli need to be refined, in which at least the following choices need to be addressed:

- Which alternatives need to be included?
- Which attributes to include for each alternative?

For example, *alternatives* can be existing or not-yet-existing transport modes in the area of interest. Each mode can have different *attributes* (travel time, waiting time, comfort, etc.). Additionally, the *model type* has to be chosen, appropriate to the problem. In other words, is the MNL model, the NL model, or perhaps the MMNL model suitable?

Essentially, the complete specification of the utility functions needs to be known. For the example in Figure 5.3, the chosen MNL model consists of two utility functions (hence two alternatives are considered), and each alternative has two attributes (the first alternative has attributes $x_1$ and $x_2$, while the second alternative has attributes $x_3$ and $x_4$.
Another important decision to make is whether an attribute is *generic* over different alternatives, or alternative-specific. In the example, $x_1$ and $x_3$ are assumed to be generic, as they have share the same generic parameter $\beta_1$, while the constant $\beta_0$ and the parameters $\beta_2$ and $\beta_3$ are alternative-specific. For example, the attribute travel time can be differently weighted in the utility functions of different mode alternatives, while it is typically weighted equally in case of different route alternatives. If one is not certain about parameters being generic or alternative-specific, then it is best to make them alternative-specific, as this can then be tested afterwards when estimating the parameters. However, each additional parameter in the model represents an extra *degree of freedom*[2], meaning that the experimental design may become larger (although this is typically not substantial). The minimum number of choice situations in the experimental design is discussed in Section 5.2.2.

Also of importance is to decide if any *interaction effects* (such as $x_1x_2$) besides the *main effects* will be included in the model. Finally, the decision has to be made if *nonlinear effects* are taken into account, either estimated using dummy-coded or effects-coded variables. These will introduce extra parameters to be estimated and also impact the number of attribute levels used in the experimental design.

Once the model has been completely specified, the experimental design can be generated. It is important to note that the experimental design will be specifically determined for the specified model and may be sub-optimal if other models are estimated using the data obtained from the stated choice experiment. Hence, estimating an MMNL model is done best using data from a stated choice experiment using a design generated based on the same MMNL model. Adding extra variables to the utility functions later in estimation, such as socio-economic data (age, gender, income, etc.), may make the experimental design again sub-optimal, hence is possible they should be taken into account from the beginning.

### 5.2.2 Step 2 - Generation of experimental design

Once the model specification is known, the experimental design can be created. An experimental design describes which hypothetical *choice situations* the respondents are faced with in the stated choice experiment. It typically consists of a table of numbers (as illustrated in Figure 6.3) in which each row represents a choice situation. The numbers in the table correspond to the *attribute levels* for each attribute (e.g., -1, 1) and are replaced by their actual attribute levels later on in the questionnaire (e.g., $1, $1.50). In the example, there are in total eight choice situations and four different columns for each of the four attributes. Different coding schemes can be used for representing the attribute levels in the experimental design. The most common ones are design coding (0, 1, 2, 3, etc.), orthogonal coding ({-1,1} for two levels, {-1,0,1} for three levels, {-3,-1,1,3} for four levels, etc.), or coding according to the actual attribute level values.

There are many experimental designs possible, and the aim here is to determine the best one. Before finding the best design, some design decisions have to be made. These include:

- Should the design be labelled or unlabelled?
- Should the design be attribute level balanced?
- How many attribute levels are used?
- What are the attribute level ranges?
- What type of design to be used?
- How many choice situations to use?

If the model specification has alternatives with alternative-specific parameters, then these alternatives need to be *labeled* (e.g., car, train, bus) in the experiment. If alternatives have generic parameters, they can be *unlabeled* (e.g., route A, route B, route C).

Almost all experimental designs created satisfy the *attribute level balance* property, which means that each attribute level appears an equal number of times for each attribute. In the example, in each column -1 and 1 both appear exactly four times. Although imposing attribute level balance may restrict the design to be sub-optimal, it is generally considered a desirable property. Having attribute level balance ensures that the parameters can be estimated well on the whole range of levels, instead of just having data points at only one or few of the attribute levels. For most designs, Ngene assumes designs will display the attribute level balance property. Where there are exceptions, such as designs generated using the Modified Federov algorithm, this will clearly be indicated in the manual.

The *number of attribute levels* to use depends on the model specification. If nonlinear effects are expected for a certain attribute, then more than two levels need to be used for this attribute in order to be able to estimate these nonlinearities. If dummy and/or effects coded attributes are included, then the number of levels to use for these attributes is predetermined. The more levels used, the higher the number of choice situations will be. Also, mixing the number of attribute levels for different attributes may yield a higher number of choice situations (because of attribute level balance). For example, if there are three attributes with 2, 3, and 5 levels, respectively, then the minimum number of choice situations will be 30 (since this is divisible by 2, 3, and 5). On the other hand, if one would use 2, 4, and 6 levels, then only a minimum of 12 choice situations would be enough. Therefore, it is wise not to mix too many different numbers of attribute levels, or at least have all even or all odd numbers of attribute levels.

Regarding the *attribute level range*, research suggests that using a wide range (e.g., $1-$6) is statistically preferable to using a narrow range (e.g., $3-$4) as this will theoretically lead to better parameter estimates (i.e., parameter estimates with a smaller standard error), although using too wide a range may also be problematic (see Bliemer and Rose, 2009). The reason for this is that the attribute level range will impact upon the likely choice probabilities obtained from the design, which we show later to impact upon the expected standard errors from that design. Having too wide a range will likely result in choice tasks with dominated alternatives (at least for some attributes) whereas too narrow a range will result in alternatives which are largely indistinguishable. We have to emphasize that this is a pure statistical property and that one should take into account the practical limitations of the attribute levels. The attribute levels shown to the respondents have to make sense. Therefore, there is a trade-off between the statistical preference for a wide range and practical considerations that may limit the range.

Several different *design types* can be considered. A *full factorial design* (see Section 6.1.1) consists of all possible different choice situations and with this design all possible effects (main and interaction effects) can be estimated. However, for a practical study the number of choice situations in a full factorial design is too large. Therefore, most people rely on so-called *fractional factorial designs* (see Section 6.2.2), and within this class there exist many different types of designs. One could randomly select choice situations from the full factorial, but clearly this is not the best way of doing it. Rather, one selects choice situations in a structured manner, such that the best data from the stated choice experiment will be produced for estimating the model. A fractional factorial design consists of subset of choice situations from the full factorial. The most well-known fractional factorial design type is the *orthogonal design* (see Section 6.1.2), which aims to minimize the correlation between the attribute levels in the choice situations. As will be shown in Section 6.1.6, these orthogonal designs have limitations and cannot avoid choice situations in which a certain alternative is clearly more preferred over the others (hence not providing much information). More recently, several researchers have suggested another type of fractional factorial designs, so-called *efficient designs* (see Chapter 7). Instead of merely looking at the correlation between the attribute levels, they aim to find designs that are statistically as efficient as possible in terms of predicted standard errors of the parameter estimates. Essentially, these designs try to maximize the information from each choice situation. Efficient designs will be able to outperform the orthogonal designs, however prior parameter estimates need to be available. Therefore, efficient designs rely on the accuracy of the prior parameter estimates. In order to obtain more stable designs that rely less on the accuracy of the priors, the last few years *Bayesian efficient designs* have been proposed (see Section 7.3). Instead of assuming fixed prior parameters, the priors are considered to be random parameters. Some other design types have been considered very recently, in which attribute level balance is abandoned, in which constraints on attribute levels are imposed, in which attribute levels are pivoted around realistic values for each respondent, or in which covariates (such as socio-economics data) are already considered when creating the design. These design types, being at the frontier of the current state-of-the-art, will be briefly discussed in Chapter 8.

Unlike most other data types where an observation typically represents information captured about a specific respondent or agent, in discrete choice data each alternative $j$ represents a unique observation. This is because each alternative is observed to be chosen or not, hence providing information down to this level of detail. In grouping the alternatives together in choice tasks, there therefore exist $J$-1 independent choice probabilities within each choice situations $S$ which will be estimated. As such, for first preference (pick one) tasks, the total number of independent choice probabilities obtained from any given design will be equal to ($J$-1)$S$ with the maximum number of parameters, $K$, including constants, that can be estimated from that design having to be less than or equal to this number. As such, the *number of choice situations* is bounded from below by ($J$-1)$S$ =$K$, and the number of choice situations required to ensure attribute level balance. Also the design type may restrict the number of choice situations. An orthogonal design sometimes needs (many) more choice situations than the minimum number determined by the number of degrees of freedom and attribute level balance, merely because an orthogonal design may not exist or may be unknown for these dimensions. A full factorial design has a predetermined number of choice situations, only influenced by the total number of attributes and the number of attribute levels.

It should be noted that determining a “good” experimental design is not a simple task as there are generally billions of possible designs and it is impossible to evaluate all of them. Typically, computer software is used to assist in this process.

### 5.2.3 Step 3 - Construction of questionnaire

Using the underlying experimental design, the actual questionnaire instrument can be constructed (see Figure 5.2). Obviously, the experimental design represented by a table of numbers is meaningless to a respondent, hence it needs to be transformed somehow so as to be meaningful for the respondent. Each row in the experimental design is translated into a choice situation as illustrated for the first three rows in Figure 2. In this example, all four attributes have two levels each, denoted by -1 and 1 in the experimental design. These numbers are replaced by meaningful values for each attribute, e.g., 10 minutes and 15 minutes for the travel time attribute for the car and train alternatives, and $1 and $1.50 for the cost/fare attribute for both alternatives. Furthermore, for each respondent the order of the choice situations should be randomized in order to rule out any possible effects the ordering may have on the estimation.

In the end, the questionnaire can be either written down on paper, can be programmed into software for computer-aided personal interviewing (CAPI), or implemented as an internet survey. Of course, CAPI and internet surveys are much more flexible (choice situations can be responsive to earlier responses or automatically tailor-made for each respondent), enable more advanced surveys, and make the data readily available without human data entry errors. Therefore, most stated choice surveys nowadays are computer-based.

## 5.3 Notation

For the remainder of this manual we will use the following notation when describing various aspects of experimental design. Let each alternative $j$, $j$ = 1, ..., $J$, have $K_j$ associate attributes. Let the number of choice situations be denoted by $S$, and the number of respondents by $N$. Suppose that each respondent $n$, $n$ = 1, ..., $N$, faces all $S$ choice situations. In each choice situation $s$, $s$ = 1, ..., $S$, each alternative has attributes with different attribute levels $x_{jks}$, $k$ = 1, ..., $K_j$. The objective is to determine the experimental design matrix $X_n = [x_{jksn}]$ for each respondent $n$ with $x_{jksn} \in \Lambda_{jkn}$ where $\Lambda_{jkn}$ is the set of possible attribute levels for each attribute for respondent $n$. Let $l_{jk} = |\Lambda_{jkn}|$ denote the number of levels for this attribute. In classical experimental designs, each respondent faces the same attribute levels in the same choice situations, hence the subindex $n$ can be omitted from the variables describing the attribute levels. However, in some cases a different design for each respondent is created, such that this subindex $n$ is important.

# Chapter 6

# Orthogonal Designs

# 6 Orthogonal Designs

## 6.1 Theory of full and fractional factorial designs

In this chapter we discuss how to generate orthogonal designs. Before doing so however, we first discuss the theory underlying the generation of these types of designs. We begin with a discussion of full factorial designs.

### 6.1.1 Full factorial designs

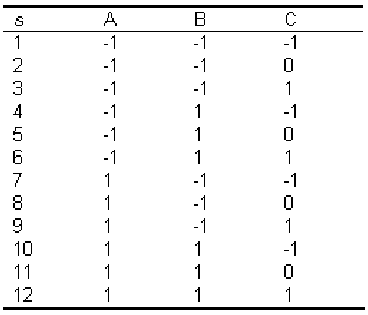
A *full factorial design* considers each possible choice situation, i.e., each possible combination of the attribute levels. Table 1 shows the full factorial design in case of three attributes (A, B, and C) with two, two, and three levels, respectively (using orthogonal coding). In total there are twelve choice situations.

In general, if there are *J* alternatives, each with $K_j$ attributes, where attribute k $\in$ $K_j$ has $l_{jk}$ levels, then the total number of choice situations in the full factorial design is

$$S^{ff} = \prod_{j=1}^{J}\prod_{k=1}^{K_j} \ell_{jk}. \tag{6.1}$$

In case of two alternatives, each having three attributes with four attribute levels each, the total number of combinations is (4 x 4 x 4) x (4 x 4 x 4) = $4^{2x3}$ = 4,096. Clearly, this number increases rapidly, and it is not feasible to let a single respondent face all these choice situations. Therefore, only for the smallest problems the full factorial design can be used. However, generating the full factorial design may be useful for determining other designs, such as certain fractional factorial designs (e.g., constrained designs, see Section 8.2).

**Table 6.1: Example full factorial design**



| s | A | B | C |
|---|---|---|---|
| 1 | -1 | -1 | -1 |
| 2 | -1 | -1 | 0 |
| 3 | -1 | -1 | 1 |
| 4 | -1 | 1 | -1 |
| 5 | -1 | 1 | 0 |
| 6 | -1 | 1 | 1 |
| 7 | 1 | -1 | -1 |
| 8 | 1 | -1 | 0 |
| 9 | 1 | -1 | 1 |
| 10 | 1 | 1 | -1 |
| 11 | 1 | 1 | 0 |
| 12 | 1 | 1 | 1 |

In the more practical *fractional factorial designs*, each respondent is only shown a subset of *S* choice situations from the total number of choice situations. One option is to randomly select choice situations from the full factorial. Another option is to give the first *S* choice situations to the first respondent, the second *S* choice situations to the second respondent, and so on. Both options can lead to biased outcomes, as for example a respondent may face only low or only high values of a certain attribute. This could be avoided by choosing the subsets in such a way that attribute level balance is satisfied. *Orthogonal designs* and *efficient designs* select subsets in a more structured way, as will be outlined in the next sections.

### 6.1.2 Orthogonal designs

Orthogonal designs have been used in experimental design for a long time. It should be noted that nowadays optimal/efficient designs exist (described in the next section) and are gaining in popularity among researchers. However, for reasons of history and inertia, orthogonal designs remain mainstream.

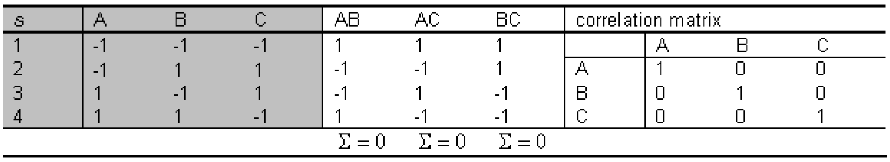
### 6.1.3 Definition of orthogonality

An *orthogonal design* is said to be orthogonal if it satisfies attribute level balance and all parameters are independently estimable. This translates into the definition that the attribute levels for each attribute column in the design need to be uncorrelated. In case of using orthogonal coding, an orthogonal design satisfies the property that the sum of the inner product of any two columns is zero:

$$\sum_{s=1}^{S} x_{j_1k_1s} x_{j_2k_2s} = 0, \quad \forall (j_1,k_1) \neq (j_2,k_2). \qquad (6.2)$$

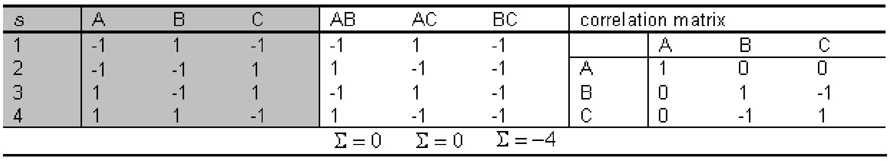
This is illustrated by the orthogonal design in Table 6.2. The design in Table 6.3 is not orthogonal, as the sum of the inner product of columns B and C is not equal to zero. As can be observed from the correlation matrix, columns B and C are perfectly (negatively) correlated.

**Table 6.2: Orthogonal design with three attributes having two levels**



| s | A | B | C | AB | AC | BC | correlation matrix | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | -1 | -1 | -1 | 1 | 1 | 1 | | A | B | C |
| 2 | -1 | 1 | 1 | -1 | -1 | 1 | A | 1 | 0 | 0 |
| 3 | 1 | -1 | 1 | -1 | 1 | -1 | B | 0 | 1 | 0 |
| 4 | 1 | 1 | -1 | 1 | -1 | -1 | C | 0 | 0 | 1 |
| | | | | $\Sigma = 0$ | $\Sigma = 0$ | $\Sigma = 0$ | | | | |

**Table 6.3: Non-orthogonal design with three attributes having two levels**



| s | A | B | C | AB | AC | BC | correlation matrix | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | -1 | 1 | -1 | -1 | 1 | -1 | | A | B | C |
| 2 | -1 | -1 | 1 | 1 | -1 | -1 | A | 1 | 0 | 0 |
| 3 | 1 | -1 | 1 | -1 | 1 | -1 | B | 0 | 1 | -1 |
| 4 | 1 | 1 | -1 | 1 | -1 | -1 | C | 0 | -1 | 1 |
| | | | | $\Sigma = 0$ | $\Sigma = 0$ | $\Sigma = -4$ | | | | |

Orthogonality is preserved if columns are left out, however not when rows are left out. Therefore, if an orthogonal array exists with more columns than is needed, one can randomly select columns to enter the design, and re-arrange them in any preferred order. Also, multiplying one or more columns by -1 preserves orthogonality. Therefore, from the orthogonal design in Table 6.2, in total eight different orthogonal designs can be generated using all possible combinations of column multipliers: (1,1,1), (-1,1,1), (1,-1,1), (1,1,-1), (-1,-1,1), (-1,1,-1), (1,-1,-1), and (-1,-1,-1). Furthermore, when replacing the orthogonal codes with the actual attribute levels when constructing the questionnaire, one is not restricted to assign the attribute levels in the same order as the orthogonal coded levels. For example, one is free to choose the replacement {-1,0,1} → {$1,$2,$3} or {-1,0,1} →{$2,$1,$3}, again preserving orthogonality.

### 6.1.4 Generating orthogonal designs

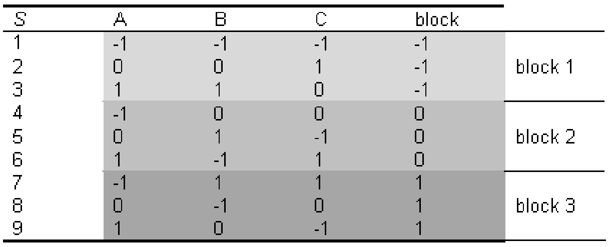
The problem of finding an orthogonal design can be described as follows:

> Given feasible orthogonal coded attribute levels $\Lambda_{jk}$ for all *j* and *k*, given a minimum number of choice situations *S*, determine the smallest level balanced design *X* with $X_{jks} \in \Lambda_{jk}$ such that Equation (6.2) is satisfied.

Determining orthogonal designs is not a straightforward task. Suppose that one searches for an orthogonal design for five attributes having three levels each. The smallest number of choice situations possible that satisfy the degrees of freedom and attribute level balance is six. However, in this case an orthogonal design with six choice situations does not exist. Even in nine or twelve choice situations it does not exist. We are able to find an orthogonal design with no less than 18 choice situations for this problem. Tables of orthogonal arrays have been derived mathematically for different numbers of columns and levels. These tables are limited and there may not be an orthogonal array known for the problem at hand. There are many lists with two, three, or even four levels, but higher levels become rare, and when mixing different numbers of levels it becomes even harder to find an orthogonal design. For example, Hahn and Shapiro (1966) have published tables with orthogonal designs for certain instances of numbers of attributes and attribute levels, but these are restricted to fairly small models. Computer programs can try to find near-orthogonal designs that can be used.

If an orthogonal design has been found, it may still be too large to give all choice situations to a single respondent. An often used procedure called *blocking* can split the orthogonal design into smaller designs. Each block is not orthogonal by itself, only the combination of all blocks is orthogonal. Blocking mainly ensures that attribute level balance is satisfied within each block, such that respondents do not just face only low or high attribute levels for a certain attribute. Blocks are typically determined by using an extra uncorrelated column with a number of levels equal to the number of blocks. This is illustrated in Table 6.4. One can check that the design for attributes A, B and C is orthogonal, and that also the blocking column is orthogonal to all other columns. The orthogonal design with nine choice situations is blocked into three blocks, such that each respondent now only has to face three choice situations instead of nine. Note that attribute level balance is satisfied within each of the blocks.

**Table 6.4 Blocking an orthogonal design in three blocks**



| *S* | A | B | C | block | |
|---|---|---|---|---|---|
| 1 | -1 | -1 | -1 | -1 | |
| 2 | 0 | 0 | 1 | -1 | block 1 |
| 3 | 1 | 1 | 0 | -1 | |
| 4 | -1 | 0 | 0 | 0 | |
| 5 | 0 | 1 | -1 | 0 | block 2 |
| 6 | 1 | -1 | 1 | 0 | |
| 7 | -1 | 1 | 1 | 1 | |
| 8 | 0 | -1 | 0 | 1 | block 3 |
| 9 | 1 | 0 | -1 | 1 | |

Orthogonal designs can be created manually, or can be found in documents such as Hahn and Shapiro (1966), or can be created automatically using software such as Ngene.

### 6.1.5 Reasons for using orthogonal designs

Aside from the fact that orthogonal designs allow for an independent estimation of the influence of each design attribute on choice, two other reasons lie behind the common use of orthogonal designs in practice. The first reason is that they are generally easy to construct or obtain (either from software packages or academic papers), although only for a limited number of combinations of attribute levels. Secondly, the common use of orthogonal designs in SC studies is largely a result of historical impetus. In the past, the experimental design literature has been primarily concerned with linear models (such as linear regression models), where the orthogonality of data is considered important. In linear regression models, this is because (a) orthogonality ensures that the model will not suffer from multicollinearity, and (b) orthogonality is thought to minimize the variances of the parameter estimates, which are taken from the variance-covariance (VC) matrix of the model. The VC matrix of a linear regression model is given in Equation (6.3).

$$VC = \sigma^2 [X'X]^{-1}, \tag{6.3}$$

where $\sigma^2$ is the model variance, and *X* is the matrix of attribute levels in the design or in the data to be used in estimation. Fixing the model variance (which simply acts as a scaling factor), the elements of the *VC* matrix for linear regression models are minimized when the *X* matrix is orthogonal. A design that results in a model where the elements contained within the *VC* matrix are minimized is preferable, for two reasons. Firstly, such a design will produce the smallest possible standard errors (i.e., square roots of the variances), and hence maximize the *t*-ratios produced from that model and secondly, an orthogonal design (or data set) will produce zero-off diagonals in the models *VC* matrix, thus ensuring that the parameter estimates are unconfounded with one another (i.e., no multicollinearity).

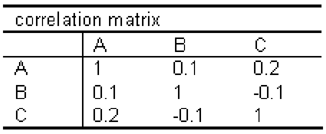
As such, orthogonal designs, at least in relation to linear models, meet the two criteria for a good design mentioned in the introduction; they allow for an independent determination of each attributes contribution on the dependent variable and they maximize the power of the design to detect statistically significant relationships (i.e., maximize the *t*-ratios at any given sample size). The question however is whether for discrete choice models, do orthogonal designs produce the same properties? Before we address this question, we first discuss several problems that often occur in practice between the mapping of design orthogonality to data orthogonality.

### 6.1.6 Discussion of orthogonal designs

It is important to understand that parameters are estimated from data sets underlined by SC experiments, not from the design itself. As we will demonstrate, only under exceptional circumstances will orthogonality be preserved within the data used to estimate discrete choice models, even if the experimental design is orthogonal. Indeed, with regards to choice data sets, one would expect orthogonality to be the exception rather than the rule. Further, even under circumstances where orthogonality is retained in a data set, as we show, orthogonality will likely be lost in the estimation process.

In case of non-response, in which a few choice situations are missing, the data will not be orthogonal. In case of blocking, if not all blocks are equally represented in the data set, then orthogonality will be lost. For example, consider again the blocked orthogonal design in Table 6.4. If blocks 1 and 2 appear twice in the data set and block 3 only once, then the data is correlated as indicated by the correlation matrix in Table 6.5. Removing data to preserve orthogonality is not common, as extra data is preferred above preserving orthogonality.

**Table 6.5: Correlation matrix with missing block**



correlation matrix

| | A | B | C |
|---|---|---|---|
| A | 1 | 0.1 | 0.2 |
| B | 0.1 | 1 | -0.1 |
| C | 0.2 | -0.1 | 1 |

Further, it is common practice to collect socio-demographic and contextual variables and include these in the utility functions of models of discrete choice. Even assuming equal representation of each choice situation of a design in the data, the current standard of sampling is such that analysts fail to ensure orthogonality between the design attributes and other variables within the data set. For example, if age, gender, or income is added as a variable in the utility function for estimation, then this attribute level is constant over all choice situations of this person, creating correlations between this variable and other attributes in the design.

Another reason that orthogonality may be lost is due to a poor transition between the design codes and the attribute level labels used within the experiment. Orthogonality of a design will only be maintained if the (quantitative) attribute level labels used are spaced equally along the range of that attribute. If unequal points are used along the attribute level range, then orthogonality will be lost. For example, if the orthogonal codes {-1,0,1} are replaced with quantitative attribute level labels {$2, $5, $15}, then the attribute levels are not equidistant in spacing. Therefore, the data will not be orthogonal.

The primary argument for using orthogonal fractional factorial designs is the ability of such designs to produce unconfounded estimates of the population parameters due to the enforced statistical independence between the attributes contained within the design. However, parameters are estimated from data sets underlined by SC experiments, not from the designs themselves. Unfortunately, only under exceptional circumstances will orthogonality be preserved within the data used to estimate discrete choice models, even if the experimental design used to construct the study is itself orthogonal. Indeed, with regards to choice data sets, one would expect orthogonality to be the exception rather than the rule (see however Lanscar et al. (2006) for an example where orthogonality has been transferred from the design through to the data). We offer three reasons for this statement.

Firstly, the principle of orthogonality as we have described it relates solely to the columns of the design matrix being uncorrelated with one another. In cases where respondents review the complete orthogonal matrix, this orthogonality will be preserved through to the data set. When respondents review subsets of the matrix however, problems can occur and orthogonality lost. If the subsets (or blocks) of the design are not replicated evenly over the survey and hence certain blocks are either over or under represented within the data, orthogonality will generally be lost.

Simply put, one cannot (i) add or remove rows of the design and/or (ii) replicate unevenly rows of the design over multiple respondents, and retain orthogonality within the data set. Note that the removal of columns from the design will not impact on the orthogonality of the design however. Thus, the onus is on the researcher to ensure that in allocating the choice tasks to respondents, that each choice task is equally represented in the final data. In cases of non-response or where the number of respondents in the study does not allow for each block to be equally distributed over the sample, this may be difficult to achieve (this last point is often missed by the literature, as it has implications on sampling and sample sizes that is rarely, if ever, discussed).

Secondly, it is typical in many choice studies to collect data on non-design attributes such as socio-demographic and contextual variables. In such cases, unless some form of strict sampling is imposed, any covariates within the data set will unlikely be orthogonal, not only amongst themselves, but also with the design attributes. For example, if age, gender, and income are added as variables in some form of analysis, correlations are not only likely to exist for these variables, but given that the variables described are constant over all choice situations within individual respondents, correlations between these variables and other attributes of the design are also likely to exist.

Finally, enforcement of orthogonality as a design principle does not ensure against the production of behaviorally implausible choice situations within the survey. Often, analysts after generating a design will review the final survey and locate choice situations in which they believe the attribute level combination of a particular alternative in a choice situation are such that that alternative has a probability of one of being chosen (i.e., that alternative dominates all other alternatives on offer in terms of preference). In such cases, no information is gained in terms of the possible trade-offs between the attributes of the alternatives. In other cases, analysts may locate choice situations whereby certain combinations of attributes are formed which may not be plausible in reality and which thus detract from the realism of the choice tasks. In these cases, analysts typically reject the choice situations (i.e., delete that row or combination of rows of the design), thus ensuring that the design and data will no longer be orthogonal (for a discussion of the benefits and costs of such strategies, see e.g., Lanscar and Louviere 2006).

Knowledge of these and other issues related to orthogonal designs are not new and have been well documented in the literature. For example, Hensher and Barnard (1990) have made a distinction between design orthogonality and estimation-data orthogonality in order to highlight that design orthogonality is not always preserved in model estimation. In making this distinction, they argued that estimation orthogonality based on discrete choice models requires that the differences in attribute levels be orthogonal, not the absolute levels themselves. Such arguments are similar to those that led to the creation of so called *difference designs* in which the absolute values of the attribute levels of the alternatives are forced to be as different as possible whilst the designs themselves remain orthogonal in the differences.

Given the above, a carefully determined orthogonal design is likely to produce non-orthogonal data in practice. As such, the question arises as to how important orthogonality is to SC experiments. In the next section efficient designs will be introduced, which seem to be outperforming orthogonal designs easily, although such designs have not been used much in practice yet.

To summarize, a carefully determined orthogonal experimental design is likely to produce non-orthogonal data. Therefore the question arises if orthogonality is that important. In the next section so-called optimal or efficient designs will be introduced, which seem to be outperforming orthogonal designs easily, although such designs have not been used much in practice yet.

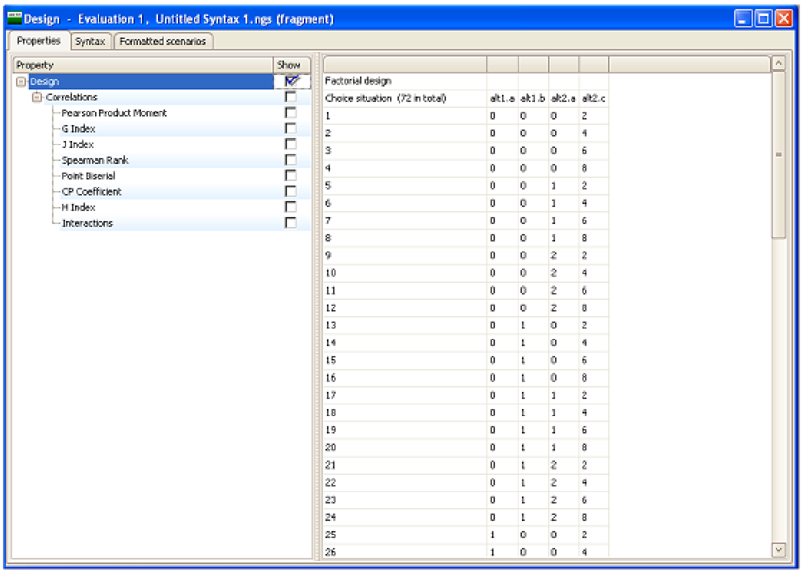
## 6.2 Generating orthogonal designs in Ngene

### 6.2.1 Full factorial designs

We demonstrated the syntax to generate full factorial designs in Section 4.2. The syntax we used was

```
Design
? This will generate a full factorial design
;alts = alt1, alt2
;rows = all
;fact
;model:
U(alt1) = b1 + b2 * A[0,1,2] + b3 * B[0,1]     /
U(alt2) =      b2 * A        + b4 * C[2,4,6,8]
$
```

In the above syntax, the *fact* property in conjunction with the *rows* property instructs Ngene to produce the full factorial design. The *rows* property is required to inform Ngene as to the number of rows that the user requires for a design. The above syntax will produce the output given in Figure 6.1. Users should be cautioned that for designs with large numbers of attributes and attribute levels, display of the full factorial may take some time. Further, full factorials with greater than 150,000 rows cannot be generated due to memory issues.



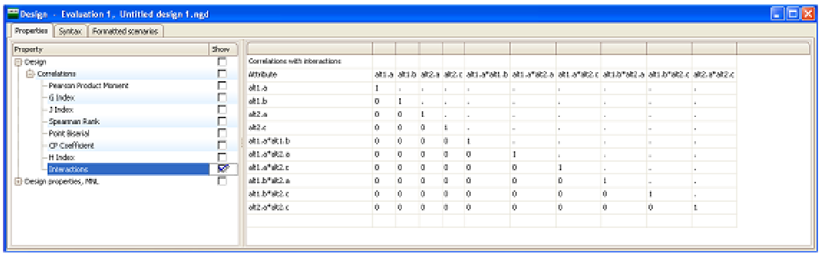
| Property | Show |
|---|---|
| Design | ☑ |
| Correlations | ☐ |
| Pearson Product Moment | ☐ |
| G Index | ☐ |
| J Index | ☐ |
| Spearman Rank | ☐ |
| Point Biserial | ☐ |
| CP Coefficient | ☐ |
| H Index | ☐ |
| Interactions | ☐ |

| | | | | |
|---|---|---|---|---|
| Factorial design | | | | |
| Choice situation (72 in total) | alt1.a | alt1.b | alt2.a | alt2.c |
| 1 | 0 | 0 | 0 | 2 |
| 2 | 0 | 0 | 0 | 4 |
| 3 | 0 | 0 | 0 | 6 |
| 4 | 0 | 0 | 0 | 8 |
| 5 | 0 | 0 | 1 | 2 |
| 6 | 0 | 0 | 1 | 4 |
| 7 | 0 | 0 | 1 | 6 |
| 8 | 0 | 0 | 1 | 8 |
| 9 | 0 | 0 | 2 | 2 |
| 10 | 0 | 0 | 2 | 4 |
| 11 | 0 | 0 | 2 | 6 |
| 12 | 0 | 0 | 2 | 8 |
| 13 | 0 | 1 | 0 | 2 |
| 14 | 0 | 1 | 0 | 4 |
| 15 | 0 | 1 | 0 | 6 |
| 16 | 0 | 1 | 0 | 8 |
| 17 | 0 | 1 | 1 | 2 |
| 18 | 0 | 1 | 1 | 4 |
| 19 | 0 | 1 | 1 | 6 |
| 20 | 0 | 1 | 1 | 8 |
| 21 | 0 | 1 | 2 | 2 |
| 22 | 0 | 1 | 2 | 4 |
| 23 | 0 | 1 | 2 | 6 |
| 24 | 0 | 1 | 2 | 8 |
| 25 | 1 | 0 | 0 | 2 |
| 26 | 1 | 0 | 0 | 4 |

**Figure 6.1: Full factorial design output**

For all experimental design types, clicking the first ‘Design’ check box will display the generated design. This is shown in Figure 6.1. Located under the ‘Design’ check box is the ‘Correlation’ branch. Ticking the ‘+’ symbol will reveal different correlation measures that may be used to examine the design. Depending on the type of data, different correlation formulas are appropriate. Clicking on one of the correlation checkboxes will result in the desired correlation measure being displayed. With the exception of the ‘Interactions’ check box, Ngene will display on the correlations for the main effects only. Selecting the ‘Interactions’ checkbox will display the correlations for the main effects and two way interaction terms for the design. This is shown in

Figure 6.2.



Correlations with interactions

| Attribute | alt1.a | alt1.b | alt2.a | alt2.c | alt1.a*alt1.b | alt1.a*alt2.a | alt1.a*alt2.c | alt1.b*alt2.a | alt1.b*alt2.c | alt2.a*alt2.c |
|---|---|---|---|---|---|---|---|---|---|---|
| alt1.a | 1 | . | . | . | . | . | . | . | . | . |
| alt1.b | 0 | 1 | . | . | . | . | . | . | . | . |
| alt2.a | 0 | 0 | 1 | . | . | . | . | . | . | . |
| alt2.c | 0 | 0 | 0 | 1 | . | . | . | . | . | . |
| alt1.a*alt1.b | 0 | 0 | 0 | 0 | 1 | . | . | . | . | . |
| alt1.a*alt2.a | 0 | 0 | 0 | 0 | 0 | 1 | . | . | . | . |
| alt1.a*alt2.c | 0 | 0 | 0 | 0 | 0 | 0 | 1 | . | . | . |
| alt1.b*alt2.a | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | . | . |
| alt1.b*alt2.c | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | . |
| alt2.a*alt2.c | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Figure 6.2: Interaction and main effect correlations for a full factorial design output**

## 6.2.2 Fractional factorial designs

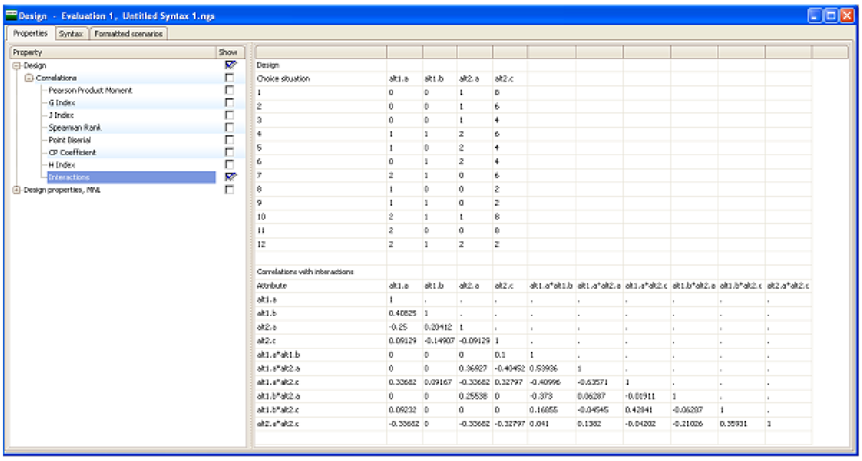
In order to create a fractional factorial design (i.e., one that does not enumerate all possible attribute level combinations), the user will need to specify the desired number of rows required for the design. The number of choice situations or rows of the design is defined using the *rows* property, which is used to restrict the number of choice situations in the design. For example, if a subset of only 12 choice situations is required, the *rows* property can be set as:

```
;rows = 12
```

Using the same design as in Section 4.2, the complete syntax would now be:

```
Design
? This will generate a fractional factorial design
;alts = alt1, alt2
;rows = 12
;fact
;model:
U(alt1) = b1 + b2 * A[0,1,2] + b3 * B[0,1]      /
U(alt2) =      b2 * A        + b4 * C[2,4,6,8]
$
```

The syntax would now generate a design of 12 choice situations by randomly choosing 12 choice situations from the full set of 72 choice situations. Figure 6.3 shows the output generated for the above syntax. Note that the generated design will change each time the syntax is run as the rows of the design are randomly taken from the full factorial design. Further, the design generated will be randomly constructed and hence need not display the attribute level balance property.



- Design ☑
  - Correlations □
    - Pearson Product Moment □
    - G Index □
    - J Index □
    - Spearman Rank □
    - Point Biserial □
    - CP Coefficient □
    - H Index □
    - Interactions ☑
- Design properties, MNL □

Design

| Choice situation | alt1.a | alt1.b | alt2.a | alt2.c |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 8 |
| 2 | 0 | 0 | 1 | 6 |
| 3 | 0 | 0 | 1 | 4 |
| 4 | 1 | 1 | 2 | 6 |
| 5 | 1 | 0 | 2 | 4 |
| 6 | 0 | 1 | 2 | 4 |
| 7 | 2 | 1 | 0 | 6 |
| 8 | 1 | 0 | 0 | 2 |
| 9 | 1 | 1 | 0 | 2 |
| 10 | 2 | 1 | 1 | 8 |
| 11 | 2 | 0 | 0 | 8 |
| 12 | 2 | 1 | 2 | 2 |

Correlations with interactions

| Attribute | alt1.a | alt1.b | alt2.a | alt2.c | alt1.a*alt1.b | alt1.a*alt2.a | alt1.a*alt2.c | alt1.b*alt2.a | alt1.b*alt2.c | alt2.a*alt2.c |
|---|---|---|---|---|---|---|---|---|---|---|
| alt1.a | 1 | . | . | . | . | . | . | . | . | . |
| alt1.b | 0.40825 | 1 | . | . | . | . | . | . | . | . |
| alt2.a | -0.25 | 0.20412 | 1 | . | . | . | . | . | . | . |
| alt2.c | 0.09129 | -0.14907 | -0.09129 | 1 | . | . | . | . | . | . |
| alt1.a*alt1.b | 0 | 0 | 0 | 0.1 | 1 | . | . | . | . | . |
| alt1.a*alt2.a | 0 | 0 | 0.36927 | -0.40452 | 0.53936 | 1 | . | . | . | . |
| alt1.a*alt2.c | 0.33682 | 0.09167 | -0.33682 | 0.32797 | -0.40996 | -0.63571 | 1 | . | . | . |
| alt1.b*alt2.a | 0 | 0 | 0.25538 | 0 | -0.373 | 0.06287 | -0.01911 | 1 | . | . |
| alt1.b*alt2.c | 0.09232 | 0 | 0 | 0 | 0.16055 | -0.04545 | 0.42841 | -0.06287 | 1 | . |
| alt2.a*alt2.c | -0.33682 | 0 | -0.33682 | -0.32797 | 0.041 | 0.1382 | -0.04202 | -0.21026 | 0.35931 | 1 |

**Figure 6.3: Fractional factorial design with interaction and main effect correlations**

If the *fact* property is not used in conjunction with the *rows* property, Ngene will attempt to optimize the design assuming that an MNL model is desired. Given that we have not assumed any priors, Ngene will assume them to be zero (see Chapter 7). Ngene will continue to run until the user intervenes and presses 'Stop'. We discuss optimization and priors further in Chapter 7.

A restriction on the minimum number of rows that can be generated is the number of parameters to be estimated, as well as the number of alternatives present in each choice task. As discussed in Section 5.2.2, ($J$-1)$S$=$K$ and hence $S$=$K$/($J$-1). The *rows* property needs to be set to a value greater than or equal to this number. In the above example, the number of parameters is four ('b1', 'b2', 'b3' and 'b4'), whilst the number of alternatives is two. As such, $S$=4/(2-1)=4. If the value specified in *rows* is not large enough, Ngene will again generate an error.

When the *fact* property is not included in the syntax and the number of rows is not equally divisible by all attribute levels specified in the utility functions of the design, Ngene will by default generate designs that do not necessarily display attribute level balance. In the above example, 12 choice situations is feasible as this number is divisible without remainder by all numbers of attribute levels (2, 3 and 4). If the number of rows is not feasible, then Ngene will generate a non-attribute level balanced design.

## 6.2.3 Orthogonal fractional factorial designs

Rather than randomly choosing choice situations from the full factorial, choice tasks may sometimes be chosen in such a way that the attribute levels are orthogonal (i.e., there are no correlations between the levels of the two attributes). The property *orth* instructs Ngene to generate such a design. Ngene can either generate a sequential orthogonal design, in which orthogonality only holds within each alternative, or generate a simultaneous orthogonal design, in which orthogonality also holds across alternatives. The properties would be:

```
;orth = seq
```

or

```
;orth = sim
```

for sequential or simultaneous orthogonal designs, respectively. Although attribute levels across alternatives are not orthogonal in a sequential orthogonal design, the sequential method of constructing orthogonal designs will typically lead to smaller designs in terms of the number of choice situations of the design. In the sequential method, first an orthogonal array is determined for the attributes of the first alternative. Next, the attribute levels of the other alternatives are derived from the levels in the first alternative. Therefore, sequential orthogonal designs can typically only be generated in cases where each utility function has the same attributes with the same levels (i.e., unlabelled alternatives). Where different alternatives have attributes with different attributes or attributes with different levels, the sequential design method described above will not work. An alternative approach available in Ngene for generating sequential orthogonal designs for experiments that have different design dimensions across alternatives (i.e., certain types of labeled choice experiments) combines separate orthogonal arrays for each alternative. That is, this approach will generate different orthogonal arrays for different alternatives and hence, each alternative can have different attributes and attribute levels. Note however that this procedure will cause correlations between alternatives but not within (similar to using *orth* = *seq*). This procedure will be used if in the syntax the *orth* property is defined differently, namely

```
;orth = seq2
```

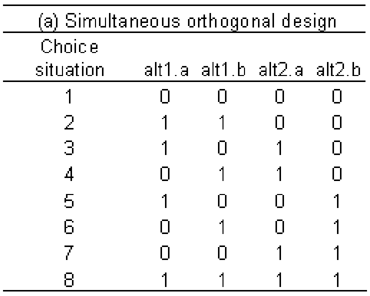
To demonstrate, consider the following two syntax routines. In the first, we have requested a simultaneously generated orthogonal design and in the second a sequentially generated design. Both designs have requested four choice situations be generated.

```
? This will generate a sequential orthogonal factorial design
Design
;alts = alt1, alt2
;rows = 4
;orth = seq
;model:
U(alt1) = b1 + b2 * A[0,1] + b3 * B[0,1] /
U(alt2) =      b2 * A      + b3 * B
$
```

```
? This will generate a simultaneous orthogonal factorial design
Design
;alts = alt1, alt2
;rows = 4
;orth = sim
;model:
U(alt1) = b1 + b2 * A[0,1] + b3 * B[0,1] /
U(alt2) =      b2 * A      + b3 * B
$
```

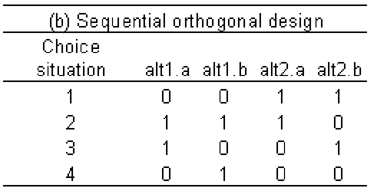
Table 6.6 shows two designs generated from the above syntax whilst Table 6.7 reports the correlation structures for the two designs. In generating the simultaneous design, Ngene was unable to locate a design in four rows where all the attributes, independent of the alternative to which it belongs to, are uncorrelated with each other. It should be noted that it cannot be guaranteed that an orthogonal design can be found with the number of choice situations specified in rows. In that case, Ngene will generate a warning message and attempt to locate an orthogonal design with more rows. In some cases an orthogonal design cannot be found at all (it may not exist or is unknown). In that case, the user will have to change some design dimensions (number of alternatives, attributes, attribute levels) and try again. It is in general easier to find sequential orthogonal designs than simultaneous orthogonal designs, as shown in the above example, where a sequential design could be located in four rows.

**Table 6.6: Simultaneous versus sequential orthogonal design generation processes**



(a) Simultaneous orthogonal design

| Choice situation | alt1.a | alt1.b | alt2.a | alt2.b |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 1 | 0 | 1 | 0 |
| 4 | 0 | 1 | 1 | 0 |
| 5 | 1 | 0 | 0 | 1 |
| 6 | 0 | 1 | 0 | 1 |
| 7 | 0 | 0 | 1 | 1 |
| 8 | 1 | 1 | 1 | 1 |



(b) Sequential orthogonal design

| Choice situation | alt1.a | alt1.b | alt2.a | alt2.b |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 |
| 2 | 1 | 1 | 1 | 0 |
| 3 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 |

**Table 6.7: Simultaneous versus sequential orthogonal correlation structures**

(a) Simultaneous orthogonal design

| | alt1.a | alt1.b | alt2.a | alt2.b |
|---|---|---|---|---|
| Alt1.a | 1 | 0 | 0 | 0 |
| Alt1.b | 0 | 1 | 0 | 0 |
| Alt2.a | 0 | 0 | 1 | 0 |
| Alt2.b | 0 | 0 | 0 | 1 |

(b) Sequential orthogonal design

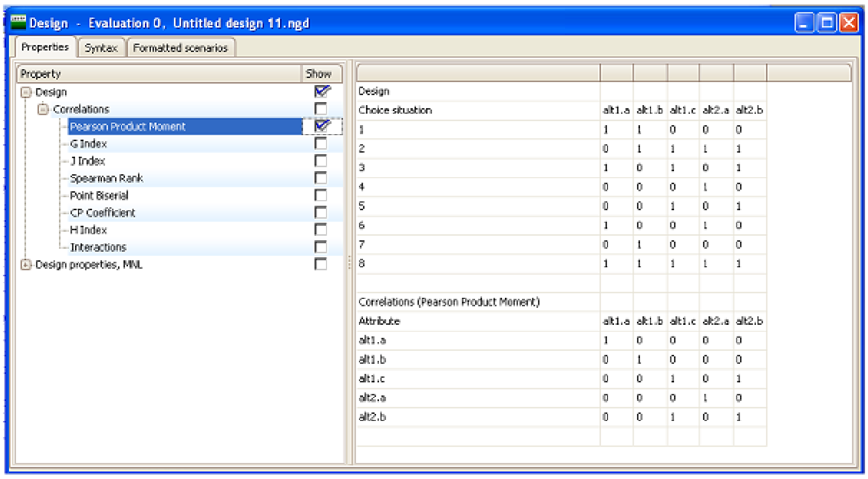
| | alt1.a | alt1.b | alt2.a | alt2.b |
|---|---|---|---|---|
| alt1.a | 1 | 0 | 0 | 0 |
| alt1.b | 0 | 1 | 0 | -1 |
| alt2.a | 0 | 0 | 1 | 0 |
| alt2.b | 0 | -1 | 0 | 1 |

For the example syntax, the smallest orthogonal design that Ngene was able to locate had eight choice situations. For the sequential design, Ngene was able to generate the design with four choice situations. Note however, that in doing so, there now exist correlations with the attributes between the alternatives.

An example design using different orthogonal arrays for each alternative is given below. Note that the first alternative has an additional attribute that is not included in the utility function given for the second alternative.

```
? This will generate a sequential orthogonal factorial design
Design
;alts = alt1, alt2
;rows = 8
;orth = seq2
;model:
U(alt1)  = b1 + b2 * A[0,1] + b3 * B[0,1]  + b4*C[0,1] /
U(alt2)  =      b2 * A      + b3 * B
$
```

Figure 6.4 shows Ngene output for the above syntax.



Design ☑
Correlations ☐
Pearson Product Moment ☑
G Index ☐
J Index ☐
Spearman Rank ☐
Point Biserial ☐
CP Coefficient ☐
H Index ☐
Interactions ☐
Design properties, MNL ☐

Design

| Choice situation | alt1.a | alt1.b | alt1.c | alt2.a | alt2.b |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 1 | 1 | 1 |
| 3 | 1 | 0 | 1 | 0 | 1 |
| 4 | 0 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 1 | 0 | 1 |
| 6 | 1 | 0 | 0 | 1 | 0 |
| 7 | 0 | 1 | 0 | 0 | 0 |
| 8 | 1 | 1 | 1 | 1 | 1 |

Correlations (Pearson Product Moment)

| Attribute | alt1.a | alt1.b | alt1.c | alt2.a | alt2.b |
|---|---|---|---|---|---|
| alt1.a | 1 | 0 | 0 | 0 | 0 |
| alt1.b | 0 | 1 | 0 | 0 | 0 |
| alt1.c | 0 | 0 | 1 | 0 | 1 |
| alt2.a | 0 | 0 | 0 | 1 | 0 |
| alt2.b | 0 | 0 | 1 | 0 | 1 |

**Figure 6.4: Fractional factorial design using *orth* = *seq2***

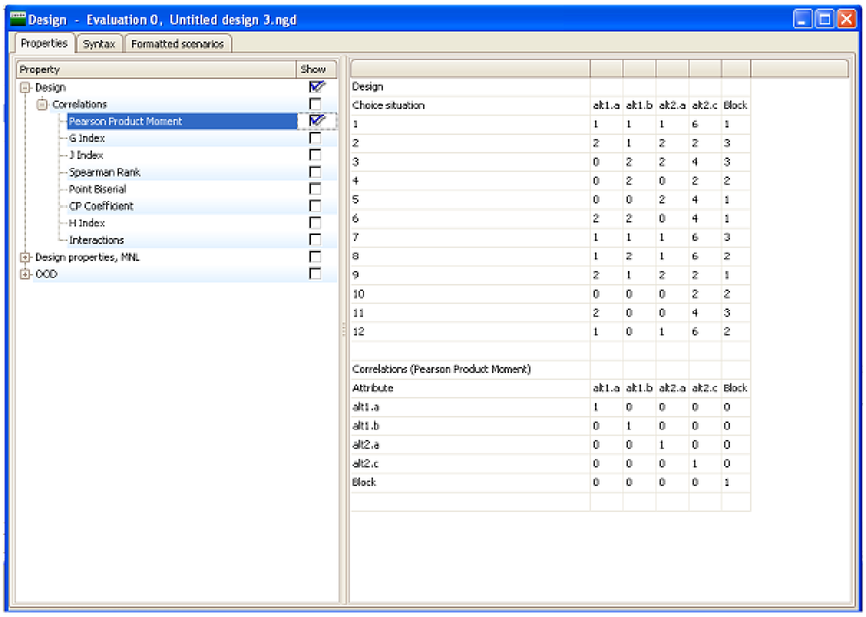
Often the number of choice situations needed to obtain an orthogonal design is too high to give to a single respondent. Therefore, the design is often blocked into smaller parts. Each block is not orthogonal by itself, only in combination with the other blocks. However, attribute level balance is maintained within each block as much as is possible. In order to automatically generate a blocked (orthogonal) design in Ngene, simply add the *block* property. In case of creating a design consisting of two blocks,

```
;block = 3
```

To demonstrate the blocking procedure, consider the following syntax. The syntax will produce an orthogonal fractional factorial design with 3 blocks.

```
? This will generate a simultaneous orthogonal factorial design with
three blocks
Design
;alts = alt1, alt2
;rows = 12
;orth = sim
;block = 3
;model:
U(alt1) = b1 + b2 * A[0,1,2] + b3 * B[0,1,2] /
U(alt2) =      b2 * A        + b4 * C[2,4,6]
$
```

Note that the number of blocks indicated in the blocking property represents the number of blocks required and not the number of choice tasks per block. Thus, the above syntax will produce an orthogonal blocking column with three blocks of four (12 / 3) choice sets each. An example design is shown in Figure 6.5.



| Property | Show |
|---|---|
| Design | ☑ |
| Correlations | □ |
| Pearson Product Moment | ☑ |
| G Index | □ |
| J Index | □ |
| Spearman Rank | □ |
| Point Biserial | □ |
| CP Coefficient | □ |
| H Index | □ |
| Interactions | □ |
| Design properties, MNL | □ |
| OOD | □ |

Design

| Choice situation | alt1.a | alt1.b | alt2.a | alt2.c | Block |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 6 | 1 |
| 2 | 2 | 1 | 2 | 2 | 3 |
| 3 | 0 | 2 | 2 | 4 | 3 |
| 4 | 0 | 2 | 0 | 2 | 2 |
| 5 | 0 | 0 | 2 | 4 | 1 |
| 6 | 2 | 2 | 0 | 4 | 1 |
| 7 | 1 | 1 | 1 | 6 | 3 |
| 8 | 1 | 2 | 1 | 6 | 2 |
| 9 | 2 | 1 | 2 | 2 | 1 |
| 10 | 0 | 0 | 0 | 2 | 2 |
| 11 | 2 | 0 | 0 | 4 | 3 |
| 12 | 1 | 0 | 1 | 6 | 2 |

Correlations (Pearson Product Moment)

| Attribute | alt1.a | alt1.b | alt2.a | alt2.c | Block |
|---|---|---|---|---|---|
| alt1.a | 1 | 0 | 0 | 0 | 0 |
| alt1.b | 0 | 1 | 0 | 0 | 0 |
| alt2.a | 0 | 0 | 1 | 0 | 0 |
| alt2.c | 0 | 0 | 0 | 1 | 0 |
| Block | 0 | 0 | 0 | 0 | 1 |

**Figure 7.5: Orthogonal fractional factorial design with orthogonal blocking column**

Note that this may be an issue if Ngene is unable to locate a design in the requested number of rows and is forced to increase the number of rows in generating a design. In this case, the number of blocks remains as specified but the number of choice tasks per block will automatically increase. If such a situation arises, the user may wish to re-specify the number of rows and blocks and generate a new design.

### 6.2.4 Orthogonal fractional factorial designs with two-way interactions

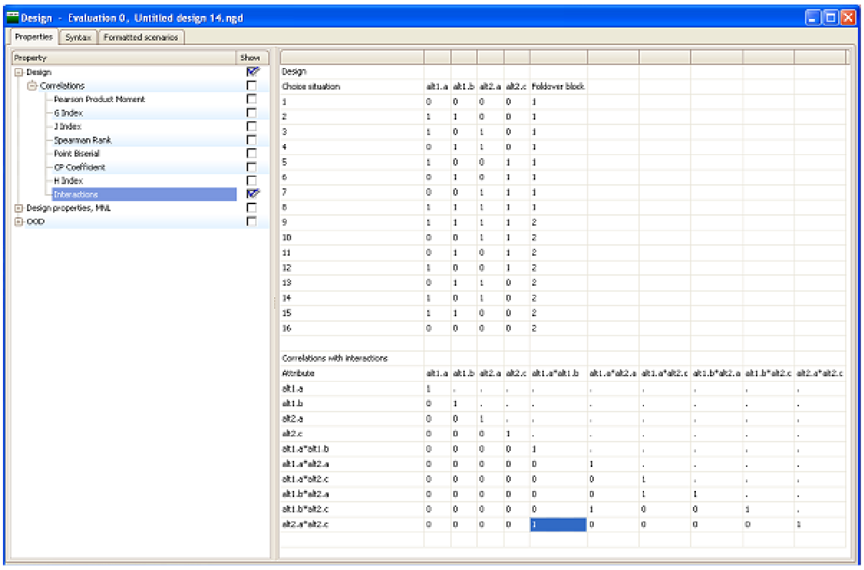
In case two-way interactions are important, one could generate a foldover design that will in many cases make all two-way interactions independent of all main effects. Note that this will not always work, but does appear to work in many instances. To create a foldover design, simply add the following property to the syntax:

```
;foldover
```

In this case, the number of choice situations will be twice as large as specified in the rows property, but the design will be blocked in two (a blocking column will be added), such that the total number of choice situations given to a single respondent does not increase. For example,

```
? use of the foldover property
Design
;alts = alt1, alt2
;rows = 8
;orth = sim
;foldover
;model:
U(alt1) = b1 + b2 * A[0,1] + b3 * B[0,1]   /
U(alt2) =      b2 * A      + b4 * C[0,1]   $
```

will result in a design similar to that shown in Figure 6.6.



- Design ☑
  - Correlations □
    - Pearson Product Moment □
    - G Index □
    - J Index □
    - Spearman Rank □
    - Point Biserial □
    - CP Coefficient □
    - H Index □
    - Interactions ☑
- Design properties, MNL □
- OOD □

Design

| Choice situation | alt1.a | alt1.b | alt2.a | alt2.c | Foldover block |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 1 | 1 | 0 | 0 | 1 |
| 3 | 1 | 0 | 1 | 0 | 1 |
| 4 | 0 | 1 | 1 | 0 | 1 |
| 5 | 1 | 0 | 0 | 1 | 1 |
| 6 | 0 | 1 | 0 | 1 | 1 |
| 7 | 0 | 0 | 1 | 1 | 1 |
| 8 | 1 | 1 | 1 | 1 | 1 |
| 9 | 1 | 1 | 1 | 1 | 2 |
| 10 | 0 | 0 | 1 | 1 | 2 |
| 11 | 0 | 1 | 0 | 1 | 2 |
| 12 | 1 | 0 | 0 | 1 | 2 |
| 13 | 0 | 1 | 1 | 0 | 2 |
| 14 | 1 | 0 | 1 | 0 | 2 |
| 15 | 1 | 1 | 0 | 0 | 2 |
| 16 | 0 | 0 | 0 | 0 | 2 |

Correlations with interactions

| Attribute | alt1.a | alt1.b | alt2.a | alt2.c | alt1.a*alt1.b | alt1.a*alt2.a | alt1.a*alt2.c | alt1.b*alt2.a | alt1.b*alt2.c | alt2.a*alt2.c |
|---|---|---|---|---|---|---|---|---|---|---|
| alt1.a | 1 | . | . | . | . | . | . | . | . | . |
| alt1.b | 0 | 1 | . | . | . | . | . | . | . | . |
| alt2.a | 0 | 0 | 1 | . | . | . | . | . | . | . |
| alt2.c | 0 | 0 | 0 | 1 | . | . | . | . | . | . |
| alt1.a*alt1.b | 0 | 0 | 0 | 0 | 1 | . | . | . | . | . |
| alt1.a*alt2.a | 0 | 0 | 0 | 0 | 0 | 1 | . | . | . | . |
| alt1.a*alt2.c | 0 | 0 | 0 | 0 | 0 | 0 | 1 | . | . | . |
| alt1.b*alt2.a | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | . | . |
| alt1.b*alt2.c | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | . |
| alt2.a*alt2.c | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

**Figure 6.6: Orthogonal fractional factorial design with foldover**

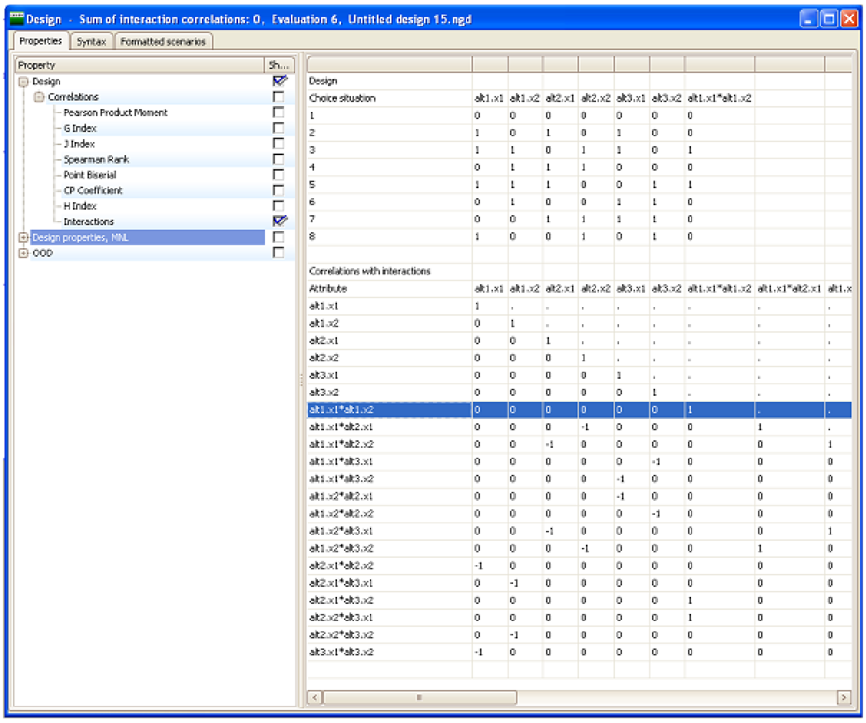
In Figure 6.6, it can be seen that the resulting design produces two-way interactions that are uncorrelated within each alternative, but which are perfectly correlated between alternatives. For unlabeled choice experiments, such correlation structures do not matter.

Rather than use foldover designs, the user may wish to specify specific interaction effects that are uncorrelated with both the main effects and other specified interaction effects. For orthogonal designs, Ngene allows the user to do so for two-way interaction effects. To specify a two-way interaction effect, the user first specifies a parameter estimate and then nominates which two attributes of the design to generate the interaction for. For example,

```
b3 * x1 * x2
```

will generate a design that will attempt to locate an uncorrelated two way interaction effect for the x1 and x2 attributes which must also be specified in the utility function of the model property. Note that in constructing designs for two way interactions, Ngene employs a search process and that there is no guarantee that such an uncorrelated interaction effect will be located. In such a case, Ngene will display the design with the requested interaction effects that have the minimal level of correlations that are possible within the search domain. Example syntax of how to construct an orthogonal fractional factorial design with a two way interaction effect is given below. Figure 6.7 shows a screen capture of a design generated using this syntax. In the screen capture, we have highlighted the requested interaction effect to demonstrate that Ngene was able to locate the requested two-way interaction effect.

```
? use of interactions specified in the model
Design
;alts = alt1, alt2, alt3
;rows = 8
;orth = sim
;model:
U(alt1) = b01 + b1 * x1[0,1] + b2 * x2[0,1] + b3 * x1 * x2 /
U(alt2) = b02 + b1 * x1       + b2 * x2                    /
U(alt3) =       b1 * x1       + b2 * x2                    $
```


Design - Sum of interaction correlations: 0, Evaluation 6, Untitled design 15.ngd
Properties
Syntax
Formatted scenarios
Property
Sh...

Design ☑
Correlations ☐
Pearson Product Moment ☐
G Index ☐
J Index ☐
Spearman Rank ☐
Point Biserial ☐
CP Coefficient ☐
H Index ☐
Interactions ☑
Design properties, MNL ☐
ODD ☐

Design

| Choice situation | alt1.x1 | alt1.x2 | alt2.x1 | alt2.x2 | alt3.x1 | alt3.x2 | alt1.x1*alt1.x2 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 4 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 5 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 6 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 7 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 8 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

Correlations with interactions

| Attribute | alt1.x1 | alt1.x2 | alt2.x1 | alt2.x2 | alt3.x1 | alt3.x2 | alt1.x1*alt1.x2 | alt1.x1*alt2.x1 | alt1.x |
|---|---|---|---|---|---|---|---|---|---|
| alt1.x1 | 1 | . | . | . | . | . | . | . | . |
| alt1.x2 | 0 | 1 | . | . | . | . | . | . | . |
| alt2.x1 | 0 | 0 | 1 | . | . | . | . | . | . |
| alt2.x2 | 0 | 0 | 0 | 1 | . | . | . | . | . |
| alt3.x1 | 0 | 0 | 0 | 0 | 1 | . | . | . | . |
| alt3.x2 | 0 | 0 | 0 | 0 | 0 | 1 | . | . | . |
| alt1.x1*alt1.x2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | . | . |
| alt1.x1*alt2.x1 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 1 | . |
| alt1.x1*alt2.x2 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 1 |
| alt1.x1*alt3.x1 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 |
| alt1.x1*alt3.x2 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 |
| alt1.x2*alt2.x1 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 |
| alt1.x2*alt2.x2 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 |
| alt1.x2*alt3.x1 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 1 |
| alt1.x2*alt3.x2 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 1 | 0 |
| alt2.x1*alt2.x2 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| alt2.x1*alt3.x1 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| alt2.x1*alt3.x2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| alt2.x2*alt3.x1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| alt2.x2*alt3.x2 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| alt3.x1*alt3.x2 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 6.7: Orthogonal fractional factorial design with specified two-way interaction effects**

## 6.3 Orthogonal optimal in the difference fractional factorial designs

A special type of a sequential orthogonal design is a so-called optimal orthogonal in the differences (OOD) design, following the design principles of Street *et al*. These researchers have identified an alternative optimality criteria to that used in generating efficient SC designs. As well as maintaining orthogonality, these researchers suggest that SC experiments should be constructed such that attributes common across alternatives should never take the same level over the experiment (see e.g., Burgers and Street, 2005; Street and Burgess, 2004; Street et al., 2001, 2005). Such designs are known as *D-optimal* designs. The argument for using this approach is that respondents are forced to trade on all attributes in the experiment, whilst the orthogonality of the design ensures that independent influence each attribute has upon choice can be determined. Optimality under this definition differs from that of *D-efficient* designs, in that *D-optimal* designs attempt to maximize attribute level differences whereas *D-efficient* designs attempt to minimize the elements that are likely to be contained within the AVC matrices of models estimated from data collected using the design. As such, a D-optimal design need not be optimal in terms of the criteria set out for D-efficient designs, with the opposite also being true. Indeed, the two optimality criteria are likely to be incompatible with one another for all but a small number of cases. Note that for constructing *D-optimal* designs, no prior parameters are used (i.e.., we assume the priors are all zeros), as one concentrates on the attribute level differences, hence efficiency will be lost in practice since the parameters are typically not equal to zero. For the interested reader, the specific steps in generating these types of designs (taken from Street et al., 2005) are outlined in detail in Appendix 6B.

In order to create these designs in Ngene, the *orth* property can be set as

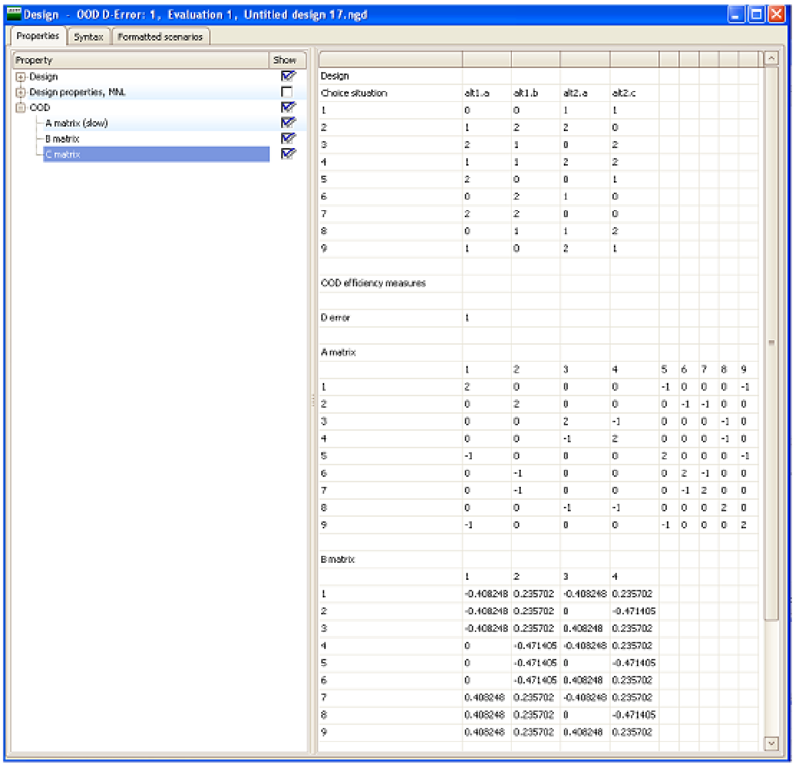
```
;orth = ood
```

Optimal orthogonal in the difference choice designs suffer from a number of issues which has not been widely discussed within the literature. Firstly, these designs may only be constructed for unlabeled SC experiments. Labeled choice experiments where attributes may not be common across alternatives, or where attribute levels may differ for common attributes are not possible for such designs, as such designs are not covered by the definition of optimality offered. Secondly, these designs may promote certain forms of behavioral response, such as lexicographic choice behavior. By forcing each attribute to be different across alternatives, a particularly dominant attribute level may govern the entire experiment[3].

Example syntax used to construct an OOD design is given below.

```
Design
;alts = alt1, alt2
;rows = 9
;orth = ood
;model:
U(alt1) = b1 * A[0,1,2] + b2 * B[0,1,2]  /
U(alt2) = b1 * A        + b2 * C[0,1,2]
$
```

Figure 6.8 presents a design generated using Ngene for the above syntax. Ngene will report a number of additional output for OOD type designs. This output can be accessed by clicking on the OOD tree structure located on the left hand side of the output screen. By clicking on the OOD click box, as shown in Figure 6.8, Ngene will report the D-efficiency value of the design (see Appendix 6B). This value represents the percentage of optimality of the design. Ngene also reports a number of matrices upon request. These matrices are used in the calculation of the D-efficiency measure. For information on what purpose these matrices serve, see Appendix 6B.


Design - OOD D-Error: 1, Evaluation 1, Untitled design 17.ngd
Properties
Syntax
Formatted scenarios

| Property | Show |
|---|---|
| Design | ☑ |
| Design properties, MNL | ☐ |
| OOD | ☑ |
| A matrix (slow) | ☑ |
| B matrix | ☑ |
| C matrix | ☑ |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Design | | | | | | | | | |
| Choice situation | alt1.a | alt1.b | alt2.a | alt2.c | | | | | |
| 1 | 0 | 0 | 1 | 1 | | | | | |
| 2 | 1 | 2 | 2 | 0 | | | | | |
| 3 | 2 | 1 | 0 | 2 | | | | | |
| 4 | 1 | 1 | 2 | 2 | | | | | |
| 5 | 2 | 0 | 0 | 1 | | | | | |
| 6 | 0 | 2 | 1 | 0 | | | | | |
| 7 | 2 | 2 | 0 | 0 | | | | | |
| 8 | 0 | 1 | 1 | 2 | | | | | |
| 9 | 1 | 0 | 2 | 1 | | | | | |
| | | | | | | | | | |
| OOD efficiency measures | | | | | | | | | |
| | | | | | | | | | |
| D error | 1 | | | | | | | | |
| | | | | | | | | | |
| A matrix | | | | | | | | | |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 2 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | -1 |
| 2 | 0 | 2 | 0 | 0 | 0 | -1 | -1 | 0 | 0 |
| 3 | 0 | 0 | 2 | -1 | 0 | 0 | 0 | -1 | 0 |
| 4 | 0 | 0 | -1 | 2 | 0 | 0 | 0 | -1 | 0 |
| 5 | -1 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | -1 |
| 6 | 0 | -1 | 0 | 0 | 0 | 2 | -1 | 0 | 0 |
| 7 | 0 | -1 | 0 | 0 | 0 | -1 | 2 | 0 | 0 |
| 8 | 0 | 0 | -1 | -1 | 0 | 0 | 0 | 2 | 0 |
| 9 | -1 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 2 |
| | | | | | | | | | |
| B matrix | | | | | | | | | |
| | 1 | 2 | 3 | 4 | | | | | |
| 1 | -0.408248 | 0.235702 | -0.408248 | 0.235702 | | | | | |
| 2 | -0.408248 | 0.235702 | 0 | -0.471405 | | | | | |
| 3 | -0.408248 | 0.235702 | 0.408248 | 0.235702 | | | | | |
| 4 | 0 | -0.471405 | -0.408248 | 0.235702 | | | | | |
| 5 | 0 | -0.471405 | 0 | -0.471405 | | | | | |
| 6 | 0 | -0.471405 | 0.408248 | 0.235702 | | | | | |
| 7 | 0.408248 | 0.235702 | -0.408248 | 0.235702 | | | | | |
| 8 | 0.408248 | 0.235702 | 0 | -0.471405 | | | | | |
| 9 | 0.408248 | 0.235702 | 0.408248 | 0.235702 | | | | | |

**Figure 6.8: Orthogonal optimal in the difference fractional factorial design**

## 6.4 Appendix 6A Correlation measures

Table 6A.1, adapted from Hensher and Smith, 1984, shows the appropriate formulae to use for different scaled data.

**Table 6A.1: Appropriate correlation formula**

Random variable scale definitions: R: Ratio; I: Interval, O: Ordinal; D: Dichotomous; N, nominal

| **Scale Pair<br>($X_1$, $X_2$)** | **Formula<br>1, 2, …, *N* observations<br>1, 2, …, *m* levels<br>$X_1$, $X_2$ = random variables** | **Test Name** |
|---|---|---|
| R,R or R, I | $r_\rho(X_1,X_2)=\dfrac{\sum_{i=1}^{N}(X_{1i}-\overline{X}_1)(X_{2i}-\overline{X}_2)}{\sqrt{\sum_{i=1}^{N}(X_{1i}-\overline{X}_1)^2}\times\sqrt{\sum_{j=1}^{N}(X_{2i}-\overline{X}_2)^2}}$ | Pearson product moment correlation coefficient [$\rho$] |
| D,D | $r_G(X_1,X_2)=\dfrac{A-B-C+D}{A+B+C+D}$ | G index [G] |

where
A = sum of positive agreeing responses ($X_1$ = +ve, $X_2$ = +ve)
B = sum of negative agreeing responses ($X_1$ = -ve, $X_2$ = -ve)
C = sum of non-agreeing responses ($X_1$ = -ve, $X_2$ = +ve)
D = sum of non-agreeing responses ($X_1$ = +ve, $X_2$ = -ve)
When the dichotomous variable (0, 1) is coded (-1, +1)

N,N or N,D

J index [J]

$$r_{\rho}(X_1, X_2) = \frac{\sum_{i=1}^{N}\sum_{s=1}^{N} X_{1i}^{(S)} X_{2i}^{(S)}}{\sqrt{\sum_{i=1}^{N}\sum_{s=1}^{N} X_{1i}^{(S)2}} \times \sqrt{\sum_{i=1}^{N}\sum_{s=1}^{N} X_{2i}^{(S)2}}}$$

where

$$d_i^2 = \begin{cases} d_x = -1 \text{ if } X_i = X_s \\ -1 \text{ otherwise} \end{cases}$$

and $d_x$ = number of categories for X

O,O

Spearman Rank correlation [SR]

$$r_{SR}(X_1, X_2) = \frac{\sum_{i=1}^{m}(X_{1i} - \overline{X}_1)^2 + \sum_{i=1}^{m}(X_{2i} - \overline{X}_2) - \sum_{i=1}^{m} d_i^2}{2\left[\sqrt{\sum_{i=1}^{m}(X_{1i} - \overline{X}_1)^2} \times \sqrt{\sum_{j=1}^{m}(X_{2i} - \overline{X}_2)^2}\right]}$$

where

$$d_i^2 = \sum_{i=1}^{m}\left(X_{1i} - \overline{X}\right)^2 - 2\sum_{i=1}^{m}\left(X_{1i} - \overline{X}\right)\left(X_{2i} - \overline{X}\right) + \sum_{i=1}^{m}\left(X_{2i} - \overline{X}\right)$$

D,R

Point Biserial correlation [PB]

$$r_{PB}(X_1^D, X_2^R) = \frac{\mu_{21} - \mu_{2o}}{\sigma_{X_2}}\sqrt{P(1-p)}$$

where $\sigma_{X_2}$ is the standard deviation of the ratio scaled random variable $X_2$, $\mu_{21}$ and $\mu_{22}$ are the means of the values of $X_2$, corresponding to the dichotomous $X_1$ variables values 1 and 0.

N,I

CP-coefficient [CP]

$$r_{CP}(X_1^I, X_2^N) = \frac{d \overbrace{\sum_{i=1}^{N}\sum_{j=1}^{N}}^{X_{2i}=X_{2j}} (X_{1i} - \overline{X}_1)(X_{1j} - \overline{X}_1)}{\sqrt{N^2 + d(d-2)\sum n_r^2} \times \sum_{i=1}^{N}(X_i - \overline{X})^2}$$

where
$n_r$ is the number of individuals with Y=r
d is the number of categories of the nominal attribute

I, I

H-INDEX [H]

$$r_H(X_1, X_2) = \frac{\sum_{i=1}^{N} X_{1i} X_{2i}}{\sqrt{\sum_{i=1}^{N}(X_{1i}^2)} \times \sqrt{\sum_{i=1}^{N}(X_{2i}^2)}}$$

## 6.5 Appendix 6B Optimal orthogonal in the differences designs

The construction of OOD designs is described in detail by Street *et al*. (2005). OOD designs are constructed to so as to maximise the differences in the attribute levels across alternatives, and hence maximise the information obtained from respondents answering SC surveys by forcing trading of all attributes in the experiment. OOD designs are limited orthogonal designs in that they are orthogonal within an alternative but have (often perfect negative) correlations across alternatives. As such, the design should generally only be applied to studies where all parameters are likely to be treated as generic (i.e., typically unlabeled choice experiments). The design generation process, as described here, also limits the experimental design to problems where each alternative has the same number of attributes, and each attribute has the same number of levels. Work has been conducted on removing some of these constraints, however we do not report on these here (see for example, Burgess and Street 2005). We restrict here our discussion to generating OOD designs to problems examining main effects only (those interested in constructing OOD designs for interactions are referred to Street *et al*. (2005) for further details). The steps for generating OOD designs are now presented.

*Step 1*: Construct an orthogonal design for the first alternative of the design (using design coding; i.e., 0, 1, 2, ..., *l*). It is from this initial design that subsequent alternatives will be constructed. The original orthogonal design can be obtained from software, cookbooks (e.g., Hahn and Shapiro 1966) or generated from first principles (see e.g., Kuehl 1994). Any orthogonal design will suffice, provided it has the same dimensions required for all alternatives in the design.

*Step 2*: Locate a suitable design generator. To do this, create a sequence of *K* values which are either equal to zero or are positive integers, where *K* is the number of attributes per alternative and each value in the sequence maps to an attribute of the second alternative. For each of the *K* values in the sequence, the value assumed can be any integer up to $l_k$ - 1, where $l_k$ is the number of levels that attribute *k* assumes.

For example, assuming the first attribute of an alternative has three levels and the second attribute has two levels, then the first value in the design generator can be zero or any integer value between one and two (i.e., between 1 and 3-1 = 2), whereas the second value in the design generator must be either zero or one (i.e., non zero, an integer and a value up to 2-1 = 1). Thus, for example, the analyst may consider as design generators sequences 11 or 21.

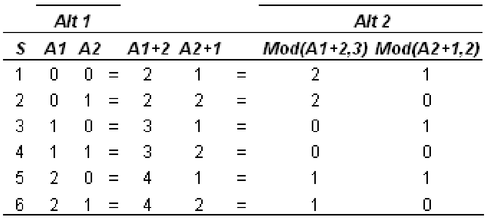
Subsequent alternatives are constructed in a similar fashion, however, where possible, design generator sequences should attempt to use unique values for each attribute of each new alternative. Design generators should also attempt to avoid using the value zero as this will lead perfectly correlated attributes in the design. For example, if the sequence 21 were used as the design generator for the second alternative, a third alternative might use the values 11 or 10. Where the same attribute across two or more alternatives have the same value in their design generators, the attributes will be perfectly confounded. For example, if we apply as design generators 21 and 11 for the second and third alternatives, the second attribute for each alternative will be perfectly confounded. Where zero is used in the generator, that attribute will be perfectly confounded with the attribute in the first alternative. For example, if we apply as design generators 21 and 10, then none of the attributes in alternatives two and three will be confounded, but the second attribute in alternative three will be perfectly confounded with the second attribute of alternative one.

*Step 3*: For each choice situation, add the sequence of values of the design generator in order of appearance to the attribute levels observed for the first alternative. For example, if the attribute levels in an alternative are 2 and 1 respectively, adding the design generator 21 results in the values 4 and 2 respectively (using design coding).

*Step 4*: Apply modulo arithmetic to the values derived in step 3. The appropriate modulo to apply for a particular attribute is equal to the number of levels for that attribute, $l_k$. Thus, for attribute one which has three levels, we use mod 3 and for the second attribute with two levels we would use mod 2. Using the design generator 21, applying mod 3 to the first attribute results in 4 Ξ 1 (mod 3) and applying mod 2 to the second attribute produces 2 Ξ 0 (mod 2). The values derived in this manner represent the levels of the second alternative. Subsequent alternatives are constructed by applying the appropriate design generator to the first alternative in the design, and applying the same modulo arithmetic rules. Table 6B.1 shows a design with six choice situations for the example problem above. Note that we have used the full factorial in constructing the first alternative. In generating experimental designs using this method, one can use a fractional factorial instead and our use of a full factorial is purely for demonstrative purposes only.

The above description represents a rather simplistic discussion on the construction of design generators for OOD designs. The reader interested in finding out more about the process is referred to Street *et al*. (2005) for a more detailed description.

**Table 6B.1: Constructing a second alternative for an OOD design**



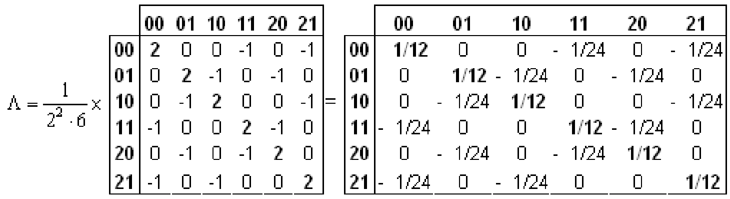
|  | *Alt 1* |  |  |  |  |  | *Alt 2* |  |
|---|---|---|---|---|---|---|---|---|
| *S* | *A1* | *A2* |  | *A1+2* | *A2+1* |  | *Mod(A1+2,3)* | *Mod(A2+1,2)* |
| 1 | 0 | 0 | = | 2 | 1 | = | 2 | 1 |
| 2 | 0 | 1 | = | 2 | 2 | = | 2 | 0 |
| 3 | 1 | 0 | = | 3 | 1 | = | 0 | 1 |
| 4 | 1 | 1 | = | 3 | 2 | = | 0 | 0 |
| 5 | 2 | 0 | = | 4 | 1 | = | 1 | 1 |
| 6 | 2 | 1 | = | 4 | 2 | = | 1 | 0 |

*Step 5*: Construct a symmetric matrix, Λ. The Λ matrix represents the proportion of times over all choice situations that each alternative (as represented by its sequence of attribute levels) appears with all other possible alternatives in the design. The Λ matrix will be a square matrix with dimensions equal to $\prod_{k=1}^{K} l_k$. Hence, working with the example above, the Λ matrix will be of dimensions 6x6 (i.e., (3x2)x(3x2)). Each column and row of the matrix relates to a potential unique combination of attribute levels that could exist within the design. In generating the matrix, we write out the full enumeration of attribute level combinations contained within a single alternative. For the above design, the combinations of attributes within an alternative can be expressed by the following sequences (using design coding); 00, 01, 10, 11, 20 and 21, where the first value in each sequence relates to the first attribute in the design and the second value, the second attribute.

To populate the Λ matrix, we simply count the number of times a particular sequence of attribute levels for one alternative appears with sequences of attribute levels in all other alternatives. For the above example, the sequence 00 appears in the first choice situation as the attribute levels in alternative 1 against the attribute levels 21 in alternative 2; The same sequence also appears in choice situation four, as the attribute levels for alternative 2 against the attribute level sequence 11 for alternative 1. Each time a combination appears together anywhere in the design, we add a -1 to the corresponding coordinates in the Λ matrix. To complete the matrix, the values of the leading diagonal are then chosen such that all rows and columns sum to zero.

We next need to scale the Λ matrix to account for the number of alternatives and choice situations in the design. To do this, we multiple each element of the matrix by $\frac{1}{J^2 S}$ where *J* is the number of alternatives in the design, and *S* is the number of choice situations. Table 6B.2 shows the Λ matrix for the above example, both before and after scaling.

**Table 6B.2: Λ matrix**



$$\Lambda = \frac{1}{2^2 \cdot 6} \times \begin{array}{c|cccccc} & \mathbf{00} & \mathbf{01} & \mathbf{10} & \mathbf{11} & \mathbf{20} & \mathbf{21} \\ \hline \mathbf{00} & \mathbf{2} & 0 & 0 & -1 & 0 & -1 \\ \mathbf{01} & 0 & \mathbf{2} & -1 & 0 & -1 & 0 \\ \mathbf{10} & 0 & -1 & \mathbf{2} & 0 & 0 & -1 \\ \mathbf{11} & -1 & 0 & 0 & \mathbf{2} & -1 & 0 \\ \mathbf{20} & 0 & -1 & 0 & -1 & \mathbf{2} & 0 \\ \mathbf{21} & -1 & 0 & -1 & 0 & 0 & \mathbf{2} \end{array} = \begin{array}{c|cccccc} & \mathbf{00} & \mathbf{01} & \mathbf{10} & \mathbf{11} & \mathbf{20} & \mathbf{21} \\ \hline \mathbf{00} & \mathbf{1/12} & 0 & 0 & -1/24 & 0 & -1/24 \\ \mathbf{01} & 0 & \mathbf{1/12} & -1/24 & 0 & -1/24 & 0 \\ \mathbf{10} & 0 & -1/24 & \mathbf{1/12} & 0 & 0 & -1/24 \\ \mathbf{11} & -1/24 & 0 & 0 & \mathbf{1/12} & -1/24 & 0 \\ \mathbf{20} & 0 & -1/24 & 0 & -1/24 & \mathbf{1/12} & 0 \\ \mathbf{21} & -1/24 & 0 & -1/24 & 0 & 0 & \mathbf{1/12} \end{array}$$

*Step 6*: Construct a matrix of contrasts for the *effects* that are of interest in the design (e.g., linear, quadratic, cubic, etc.). This matrix we call the *B* matrix. The number of rows of the *B* matrix will be equal to $\sum_{k=1}^{K} l_k - 1,$ where $l_k$ -1 corresponds to the number of effects attribute *k* can be used to test. Hence, each row will correspond to a particular effect of interest for each attribute in the design. The number of columns in the matrix will be exactly the same as the Λ matrix, which will be equal to $\prod_{k=1}^{K} l_k$. For the example above, the *B* matrix will therefore have three rows (i.e., (3-1) + (2-1) = 3) and six columns (i.e., 2x3 = 6), where the first two rows correspond to the linear and quadratic effects of the first attribute (which has three levels) and the last row to the linear effect of the second attribute (which has two levels).

To populate the *B* matrix, we first begin by determining what the coefficients of orthogonal polynomials are that correspond to each of the attributes in the design. The values that populate the matrix represent the full factorial of the possible combinations of coefficients of orthogonal polynomials. For our example, the linear coefficients of orthogonal polynomials for the first attribute are {-1, 0, 1}, and {1, -2, 1} for the quadratic effects. The linear effects for a two level attribute are simply {-1, 1}. The linear coefficients of orthogonal polynomials for the first attribute constitute the first row of the matrix, whilst the quadratic effects make up the second row. The final row represents in our example, the second attribute of the design. This row is constructed such that each level of the attribute appears against each of the linear and quadratic effects of the first attribute. Thus, the matrix of coefficients of orthogonal polynomials for our example is:

$$\text{Matrix of Coefficients of Orthogonal Polynomials} = \begin{bmatrix} -1 & 0 & 1 & -1 & 0 & 1 \\ 1 & -2 & 1 & 1 & -2 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \end{bmatrix}$$

We are next required to normalise this matrix by dividing each row of the matrix by the square root of the sum of the squares for each row of the non-normalised matrix. For the above, squaring all elements and summing each row produces values of four, 12 and six for rows 1, 2 and 3 respectively. Taking the square roots and dividing each row of the matrix of coefficients of orthogonal polynomials by these values, we obtain the B matrix which we show below.

$$B = \begin{bmatrix} -0.5 & 0 & 0.5 & -0.5 & 0 & 0.5 \\ 0.289 & -0.577 & 0.289 & 0.289 & -0.577 & 0.289 \\ -0.408 & -0.408 & -0.408 & 0.408 & 0.408 & 0.408 \end{bmatrix}$$

*Step 7*: Calculate the information matrix, *C* (El Helbawy and Bradley 1978). *C* is calculated using matrix algebra such that *C* = *B*Λ*B*'.

$$C = \begin{bmatrix} 0.06 & 0 & 0.03 \\ 0 & 0.06 & 0 \\ 0.03 & 0 & 0.11 \end{bmatrix}$$

When the *C* matrix is diagonal, all main effects will be independent, which is not the case with our example.

*Step 8*: Calculate the level of efficiency for the design. This requires first estimating the maximum value the determinant of the *C* matrix could assume and comparing this to the actual value of the C matrix for the design. The first step in determining the maximum value of the determinant of the *C* matrix is to calculate the value $M_k$ which represents the largest number of pairs of alternatives that can assume different levels for each attribute, *k*, in a choice situation. This value for each attribute *k*, can be established using Equation (6B.1). Note that the particular formula to adopt to calculate $M_k$ is a function of the number of alternatives in the design, *J*, and the number of levels of attribute *k*.

$$M_k = \begin{cases} (J^2-1)/4, & l_k = 2,\ J \text{ odd}, \\ J^2/4, & l_k = 2,\ J \text{ even}, \\ (J^2-(l_k x^2 + 2xy + y))/2, & 2 < l_k \le J, \\ J(J-1)/2, & l_k \ge J. \end{cases} \tag{6B.1}$$

and *x* and *y* are positive integers that satisfy the equation $J = l_k x + y$ for $0 \le y \le l_k$. For the case where an attribute has levels $2 < l_k \le J$, the analyst will need to fit integer values for *y* between zero and $l_k$ to obtain values of *x* that satisfies this equation. Any value of *y* that results in an integer value of *x* represents a possible candidate for the design.

For our example, the design has *J* = 2 with $l_1$ = 3 and $l_2$ = 2 and *S* = 6. As such, for the first attribute we obtain $M_1$ = $J$($J$-1)/2 = 2(2-1)/2 = 1 and for the second attribute, $M_2$ = $J^2$/4 = $2^2$/4 = 1.

Once the value of $M_k$ has been established for each attribute, the maximum value of the determinant of $C$ is calculated as:

$$\det(C_{\max}) = \prod_{k=1}^{K}\left(\frac{2M_k}{J^2(l_k-1)\prod_{i\neq k} l_i}\right)^{l_k-1}. \tag{6B.2}$$

Applying Equation (6B.2) to our example, the maximum value the determinant of $C$ could possibly achieve is

$$\det(C_{\max}) = \left(\frac{2\cdot 1}{2^2(3-1)2}\right)^{(3-1)} \cdot \left(\frac{2\cdot 1}{2^2(2-1)3}\right)^{(2-1)} = 0.002604.$$

For OOD designs, the level of efficiency of a design is expressed as a percentage referred to as *D-efficiency* in the literature. The D-efficiency of a design is calculated as follows:

$$D\text{-efficiency} = \left[\frac{\det(C)}{\det(C_{\max})}\right]^{\frac{1}{\sum_{k=1}^{K}(l_k-1)}} \times 100\%. \tag{6B.3}$$

The closer the D-efficiency to 100 percent, the more efficient the design is. For our example, the determinant of the $C$ matrix is 0.00362. From Equation (6B.3), the D-efficiency for our design is calculated as

$$D\text{-efficiency} = \left[\frac{0.000362}{0.002604}\right]^{\frac{1}{(3-1)+(2-1)}} \times 100\% = 51.79\%.$$

# Chapter 7

# Efficient Designs

# 7 Efficient Designs

## 7.1 Theory of efficient designs

In this section, we discuss the theory underlying efficient designs. Subsequent sections of the chapter outline how such designs are obtained using Ngene.

### 7.1.1 Efficient designs

In contrast to orthogonal designs, so-called *efficient* designs do not merely try to minimize the correlation in the data for estimation purposes, but aim to result in data that generates parameter estimates with as small as possible standard errors. These designs make use of the fact that the AVC matrix (the roots of the diagonal of this matrix are the asymptotic standard errors) of the parameters can be derived if the parameters are known. Unfortunately, since the objective of the SC experiment is to estimate these parameters, they are unknown. However, if some prior information about these parameters is available (e.g., parameter estimates available in the literature from similar studies, or parameter estimates from pilot studies), then this asymptotic variance-covariance matrix can be determined, assuming that the priors are correct. It can be argued that an orthogonal design is efficient only in cases where there is no knowledge about the parameters, but whenever there is any prior parameter information available (perhaps just knowledge of the sign of the parameter) then the design can be improved.

### 7.1.2 Definition of efficiency

An experimental design is called *efficient* if the design yields data that enables estimation of the parameters with as low as possible standard errors. These standard errors can be predicted by determining the AVC matrix based on the underlying experiment and some prior information about the parameter estimates. The following subsection will first briefly describe how to obtain this AVC matrix. Then, we will present several proposed efficiency measures for expressing the efficiency of an experimental design into a single value.

### 7.1.3 Deriving the asymptotic variance-covariance matrix

Let $\Omega_N$ denote the asymptotic variance-covariance matrix[4] (AVC) matrix given a sample size of N respondents (each facing S choice situations). This AVC matrix depends in general on the experimental design, X = [$X_n$], the parameter values, β, and the outcomes of the survey, $Y = [y_{jsn}]$, where $y_{jsn}$ equals one if respondent *n* chooses alternative *j* in choice situation *s* and is zero otherwise. Since the parameter values *β* are unknown, prior parameter values $\tilde{\beta}$ are used as best guesses for the true parameters.

The AVC matrix is the negative inverse of the expected Fisher Information matrix (e.g., see Train, 2003), where the latter is equal to the second derivatives of the log-likelihood function:

$$\Omega_N(X,Y,\tilde{\beta}) = -\left[E\left(I_N(X,Y,\beta)\right)\right]^{-1} = -\left[\frac{\partial^2 L_N(X,Y,\tilde{\beta})}{\partial\beta\partial\beta'}\right]^{-1}, \qquad (7.1)$$

where $I_N(X, Y, \beta)$ is the Fisher Information matrix with $N$ respondents, and $L_N(X, \tilde{\beta})$ is the log-likelihood function in case of $N$ respondents defined by

$$L_N(X,Y,\tilde{\beta}) = \sum_{n=1}^{N}\sum_{s=1}^{S}\sum_{j=1}^{J} y_{jsn} \log P_{jsn}(X,\tilde{\beta}). \qquad (7.2)$$

This formulation holds for each model type (MNL, NL, or MMNL), only the choice probabilities $P_{jsn}(X, \tilde{\beta})$ are different. Further information on these model types are given in Appendix 7A. For the MNL model, the choice probabilities given in Equation (7A.5) in Appendix 7A apply. Other probabilities are used for other model types, as discussed in Appendix 7A. There are two ways of determining the AVC matrix, either by Monte Carlo simulation, or analytically.

Most researchers have relied on *Monte Carlo simulation*. In this case, a sample of size *N* is generated and parameters are estimated based on simulated choices (by simply computing the observed utilities using some prior parameter estimates, adding random draws for the unobserved utilities, and then determine the chosen alternative by assuming that each respondent selects the alternative with the highest utility). Such an estimation also provides the results for the AVC matrix. This procedure is repeated a large number of times and the average AVC matrix gives the AVC matrix.

Many have not realized that the AVC matrix can be determined *analytically*, as suggested for MNL models with all generic parameters by McFadden (1974). In this case, the second derivative of the log-likelihood function in Equation (7.2) is determined and evaluated analytically. A potential problem is, that the vector of outcomes, *Y*, is part of the log-likelihood function, the reason why most researchers perform Monte Carlo simulations. However, it can be shown that the outcomes *Y* drop out when taking the second derivatives in case of the MNL model. This has been shown by McFadden (1974) for models with all generic parameters, and in Rose and Bliemer (2005a) for models with alternative-specific parameters, or a combination. Furthermore, Bliemer *et al*. (2009) have also derived analytical expressions for the second derivatives for the NL model. The outcomes *Y* do not drop out, but as shown in their paper, they can be replaced with probabilities leading to exactly the same AVC matrix, which has been confirmed by Monte Carlo simulation outcomes. Although more tedious, the second derivatives can also be derived for the MMNL model and a similar procedure holds for removing the outcome vector *Y*. Note that the MMNL model will always require some simulations, as the parameters are assumed to be random and therefore expected probabilities need to be approximated using simulation. However, these simulations have no connection with the simulations mentioned earlier for determining the AVC matrix. To conclude, $\Omega_N$ can be determined without knowing simulated outcomes *Y*, hence the dependency on *Y* disappears in Equation (7.2).

In the special (and most considered) case that all respondents face exactly the same choice situations, i.e., $X_n = X$ for all *n*, it can be shown that (see Rose and Bliemer, 2005a)

$$I_N(X,\tilde{\beta}) = N \cdot I_1(X,\tilde{\beta}), \qquad \text{hence} \qquad \Omega_N(X,\tilde{\beta}) = \frac{1}{N}\Omega_1(X,\tilde{\beta}). \qquad (7.3)$$

In other words, the AVC matrix corresponding to a sample size of *N* can be derived directly from the AVC matrix from a single respondent using a rate of 1/*N*. This means that the impact of sample size on the design can readily be investigated (under all assumptions made so far). The asymptotic standard errors $se_N(X, \tilde{\beta})$ are the roots of the diagonal of the AVC matrix, therefore these standard errors decrease with a rate of $1/\sqrt{N}$ of the sample size *N*. This is also illustrated in Figure 7.1 for a single parameter, clearly indicating a diminishing decreasing asymptotic standard error when the sample size increases. This is an important result, as it suggests that spending (much) more money on collecting data using a larger sample size does in the end not lead to significantly better parameter estimates, indicated by (*) in the figure. As the figure also suggests, it pays off much more to determine a design with a higher efficiency (design with attribute levels $X^{II}$ instead of $X^{I}$), in which the standard error can decrease significantly, indicated by (**) in the figure, without spending any extra money!

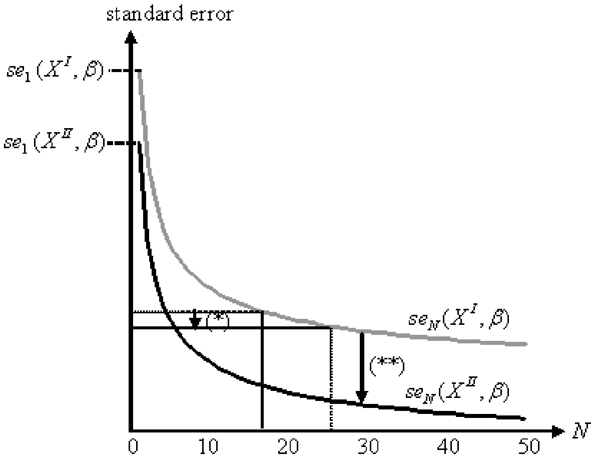


**Figure 7.1: Asymptotic standard error as a function of the sample size**

### 7.1.4 Efficiency measures

The efficiency of a design can be derived from the AVC matrix. Instead of assessing a whole AVC matrix, it is easier to assess a design based on a single value. Therefore, efficiency measures have been proposed in the literature in order to calculate such an efficiency value, typically expressed as an efficiency 'error' (i.e., a measure for the inefficiency). The objective then becomes to minimize this efficiency error.

The most widely used measure is called the *D-error*, which takes the determinant of the AVC matrix $\Omega_1$, assuming only a single respondent[5]. A design with the lowest D-error is called *D-optimal*. In practice it is very difficult to find the design with the lowest D-error, therefore we are satisfied if the design has a sufficiently low D-error, called a *D-efficient* design. Different types of D-error have been proposed in the literature, depending on the available information on the prior parameters $\tilde{\beta}$. We will distinguish three cases:

(a) No information is available; If no information is available (not even the sign of the parameters), then set $\tilde{\beta}$=0. This leads to a so-called $D_z$-error ('z' from 'zero').

(b) Information is available with good approximations of β; If the information is relatively accurate, $\tilde{\beta}$ is set to the best guesses, assuming they are correct. This leads to a so-called $D_p$-error ('p' from 'priors')

(c) Information is available with uncertainty about the approximations of β;
Instead of assuming fixed priors $\tilde{\beta}$, they are assumed to be random following some given probability distribution to express the uncertainty about the true value of β. This Bayesian approach leads to a so-called $D_b$-error ('b' from 'Bayesian').

The D-errors are a function of the experimental design $X$ and the prior values (or probability distributions) $\tilde{\beta}$, and can be mathematically formulated as:

$$D_z\text{-error} = \det\left(\Omega_1(X,0)\right)^{1/K}, \tag{7.4}$$

$$D_p\text{-error} = \det\left(\Omega_1(X,\tilde{\beta})\right)^{1/K}, \tag{7.5}$$

$$D_b\text{-error} = \int_{\tilde{\beta}} \det\left(\Omega_1(X,\tilde{\beta})\right)^{1/K} \phi(\tilde{\beta}\,|\,\theta)\, d\tilde{\beta}. \qquad (7.6)$$

where *K* is the number of parameters to be estimated. Note that the AVC matrix is a *K* x *K* matrix. In order to let the D-error be independent of the size of the problem, the D-error is normalized by the power 1/*K*. We recommend removing the rows and columns corresponding to the model constants in the AVC matrix as these parameters in general do not have a clear meaning in a SC experiment (in contrast to revealed choices). As the standard errors of these model constants can become fairly large, they could dominate the D-errors, therefore we advise to remove them before taking the determinant (and at the same time also adjust the value of *K*).

Equation (7.6) needs some more explanation. In the Bayesian D-error computation the priors $\tilde{\beta}$ are assumed to be random variables with a joint probability density function Φ(.) with given parameters Θ. For example, these priors could follow normal distributions $\tilde{\beta}$ ~ N(*μ*, Σ), or uniform distributions $\tilde{\beta}$ ~ *U*(*u*, *v*), or a mix, or other distributions. Normal and uniform distributions seem to be the only ones used in the literature so far.

Besides the D-error, other inefficiency measures have been proposed as well. Another well-known efficiency error is called the *A-error*, and the design with the lowest A-error is called *A-optimal*. Instead of taking the determinant, the A-error takes the trace of the AVC matrix, which is the summation of all diagonal elements of the matrix. Therefore, the A-error only looks at the variances and not at the covariances. In order to normalize the A-error it is divided by *K* (the same recommendation about the model constants applies). Similar to the D-error, different A-errors can be determined based on the availability of information on the parameters. The $A_p$-error is mathematically formulated as

$$A_p\text{-error} = \frac{\operatorname{tr}\left(\Omega_N(X,\tilde{\beta})\right)}{K}. \qquad (7.7)$$

The $A_z$-error and $A_b$-error can be derived using formulations equivalent to Equations (7.4) to (7.6) (see Bliemer and Rose, 2009). The A-error should be used with caution in case not all parameter values are of equal scale. By the simple summation of the variances it is likely that parameters with large values will overshadow the other parameters. Therefore, we suggest using a weighted summation. Using weights it is also possible to give more importance to certain parameters, that is, enable the estimating of these parameters more accurately than others.

A completely different efficiency measure has been introduced by Bliemer and Rose (2005a). They propose a measure that is related to the sample sizes required to estimate each parameter significantly. If the null hypothesis is that $\beta_k$ = 0 for a certain parameter, then this hypothesis is rejected if

$$\frac{\beta_k}{se_{N,k}(X,\beta)} \geq t_\alpha, \qquad (7.8)$$

where $t_\alpha$ is the *t*-value corresponding to the (1 - α)-confidence interval (e.g., $t_{0.05}$ = 1.96). Assuming that the priors are correct estimates for the true parameters and assuming that all respondents face the same choice situations, i.e., Equation (7.3) holds, then Equation (7.8) can be rewritten as

$$N \geq \left(\frac{se_{1,k}(X,\tilde{\beta})t_\alpha}{\tilde{\beta}_k}\right)^2. \qquad (7.9)$$

This number provides a lower bound on the necessary number of the sample size in order to obtain significant estimates for parameter $\beta_k$ (see Bliemer and Rose, 2009). The measure proposed by Bliemer and Rose (2005a, 2009) is derived from the observation that if some parameters need much higher sample sizes than others, it may be better in the experiment to focus more on the parameters that are difficult to estimate significantly. By spreading the information obtained from each choice situation in the design over all parameters, the design can be optimized for sample size, and is termed *S-optimality* (see Bliemer and Rose 2005a, 2009).

Note that Equation (7.9) merely provides a lower bound and does not guarantee significant parameter estimates due to random choice behavior and in the case of the MNL model, the assumption that all random components are independent, even if a single respondent faces multiple choice situations, may also impact upon the value derived. This will lead to some biases, yielding higher necessary sample sizes. The problem of dependent observations in a SC experiment is a known problem to which unfortunately no simple solution exists, besides putting the correlation structure in a random components model. Therefore, the S-optimality measure merely gives an indication in order to compare different designs on lower bounds for the sample sizes.

Several other efficiency criteria have been proposed within the literature (see e.g., Kessels *et al*., 2006) and many others can be formulated. Within Ngene, aside from D-, A-, and S-error measures of efficiency, there also exists (implemented only for the MNL model) an additional efficiency measure termed C-error (see Kanninen, 1993a,b and Scarpa and Rose 2008). The C-error measure in Ngene attempts to minimise the variance of the ratio of two parameters and as such is ideal for working with problems dealing with willingness to pay (WTP) issues. As shown in Scarpa and Rose (2008), the variance of two parameters may be approximated using Equation (8.10)

$$Var\left(\frac{\beta_k}{\beta_l}\right) \approx Var\left(\frac{\beta_l}{-\beta_k}\right) \cong \beta_k^{-2}\left[Var(\beta_l) - 2\beta_l\beta_k^{-1}Cov(\beta_l,\beta_k) + (\beta_l/\beta_k)^2 Var(\beta_k)\right]. \qquad (7.10)$$

The C-error criterion relates to the minimization of such variances. In most SC experiments, there will exist more than one WTP, with indeed up to *k*-1 potential WTPs. In such cases, the C-error has been set up to minimise the sum of the up to *k*-1 C-error values, with the user able to nominate which WTP values to include in the calculation.

## 7.1.5 Drawing from parameter distributions

In the previous section, we saw that there exist multiple efficiency criteria that one may use when generating efficient designs. We further saw that within each efficiency measure, there exist multiple approaches regarding the parameter priors assumed in generating efficient SC experiments. In the first approach, researchers have made the strong assumption that all parameter priors for the design are simultaneously equal to zero (e.g., Burgess and Street 2005; Grasshoff and Schwabe 2007; Huber and Zwerina 1996; Street and Burgess 2004; Street *et al*. 2001). Street et al. make this assumption for analytical reasons, enabling them to locate truly optimal (most efficient) orthogonal designs. This optimality will only exist under the assumption of zero parameter estimates, which is unlikely to hold in reality. A second approach that has sometimes been used is to assume that the parameter priors are non-zero and known with certainty (e.g., Carlsson and Martinsson 2003; Huber and Zwerina 1996; Rose and Bliemer 2005). In such an approach, a single fixed prior is assumed for each attribute. Whilst the assumption of perfect certainty is a strong one, the design generation process is such that researchers are able to test its impact on a design’s efficiency assuming misspecification of the priors. Sándor and Wedel (2001) introduced a third approach by relaxing the assumption of perfect *a priori* knowledge of the parameter priors through adopting a Bayesian approach to the design generation process.

The Bayesian approach to constructing efficient SC experiments requires that the efficiency of a design be evaluated over numerous different draws taken from the prior parameter distributions assumed in generating the design. The Bayesian efficiency of a design is then calculated as the expected value of whatever measure of efficiency is assumed over all the draws taken. The Bayesian approach therefore necessitates the use of simulation methods to approximate the expectations for differing designs.

For computing the Bayesian efficiency, a number of different simulation procedures are available to researchers, with the simplest being the use of pseudo random draws. In using pseudo random draws (often referred to as pseudo Monte Carlo, or PMC, draws), points from a distribution are randomly selected. Whilst simple to implement in practice, results obtained using PMC draws are susceptible to being specific to the particular draws taken from whatever distribution is assumed, with different sets of random draws likely to produce different coverage over the distribution space, possibly leading to widely different results when calculating the expectations. This risk is especially high with the use of a small number of draws. The precision of simulation processes may potentially be improved by using a more systematic approach in selecting points when sampling from a distribution. Such techniques are commonly referred to within the literature as quasi random Monte Carlo draws (see, for example, Bhat 2001, 2003; Hess *et al*. 2005; Sándor and Train 2003). The potential to provide better coverage of the distribution space for each prior parameter distribution should theoretically result in a lower approximation error in calculating the simulated choice probabilities for a given design. This in turn will result in greater precision in generating the design's AVC matrix, resulting in greater precision in terms of the Bayesian efficiency measure of that design. Other methods, such as Gaussian quadrature, also aim to minimize the approximation error when calculating the Bayesian efficiency.

Independent of the type of draws used, the researcher must decide on the number of draws to use. If too few draws are taken, it is probable that the resulting Bayesian measure of efficiency will be far from the true efficiency for a given design. If too many draws are used, the computation time in generating an efficient design will be unnecessarily high. The issue therefore becomes one of how many draws should be used before the Bayesian measure of efficiency will converge to the true efficiency level for a given design, or alternatively, fall within some acceptable error range around the true value. Unfortunately, the answer to this question will likely depend on the dimensions of the design itself, the number of Bayesian priors assumed, the population of the prior distributions, the type of econometric model used, as well as the type of draws employed. Kessels *et al*. (2006) argue that a well-designed systematic 20-point sample may be sufficient to give a good enough approximation of the Bayesian efficiency, at least in a first step of a search algorithm, although no claims can be given for general experiments. Improvements in search algorithms and in faster evaluations of the Bayesian efficiency should both lead to significantly smaller computation times for determining a Bayesian efficient design. From a search algorithm perspective (for unlabeled experiments), the reader is referred to Kessels *et al*. (2006) and Yu *et al*. (2008), which deal with determining Bayesian efficient designs for the MNL and MMNL model, respectively.

Ngene allows the use of the PMC method alongside three different types of quasi random Monte Carlo draws; namely Halton, Sobol, and Modified Latin Hypercube Sampling (MLHS) draws, and one Gaussian quadrature method, namely Gauss-Hermite approximation. Independent of the method, the principles in generating efficient SC experiments remain the same:

1)first, *R* values are drawn from the random distribution of the prior parameter values;
2)then, for each of these parameter values, the D-error is evaluated; and
3)an average D-error is computed over these values (giving the D*b*-error).

The PMC and quasi-random MC methods all take a simple (unweighted) average of the different D*b*-errors (or any other efficiency method), but differ in the way they take the draws from the random distribution. In the PMC method, these draws are completely *random*, whereas in the quasi-random MC methods they are intelligent and structured, and in most cases deterministic. The Gaussian quadrature methods construct intelligent and deterministic draws as well, but also determine specific weights for each draw and compute a weighted average.

Sandor and Wedel (2001, 2002) suggested that when generating Bayesian efficient designs, the generalised Asymptotic Fisher Information matrix be used instead of the Asymptotic Fisher Information matrix. This approach has also been proposed and used by Kessels, et al. (2006) and Yu et al. (2009). The generalised Asymptotic Fisher Information matrix is calculated as $\int_{\beta} \det\left[I_N\left(x,\tilde{\beta}\right)+S_{\beta}\right]^{\frac{-1}{K}} \phi(\tilde{\beta} \mid \theta) d\tilde{\beta}$ where $S_{\beta}$ are the prior parameter variances. Chaloner and Verdinelli (1995) argue in favour of the common $D_b$-error measure, as it allows for different prior information, to be used in the design and analysis and is appealing when a non-Bayesian framework is adopted in analysis. In addition, the traditional $D_b$-error is based on an asymptotic approximation of the posterior, and the prior vanishes in any case.

We now discuss each of these methods in turn. Further information on the impact of changing the number of draws by type is available in Bliemer *et al*. (2008).

### 7.1.5.1 Pseudo-random Monte Carlo (PMC) simulation

In PMC simulation, for each of the *K* parameters, *R* independent draws are taken from their given prior distributions. For each of these *R* draws of the prior parameters, the D*b*-error is computed. Finally, the average is taken of all computed efficiency measures. Let $\tilde{\beta}^{(r)} = [\tilde{\beta}_1^{(r)}, ..., \tilde{\beta}_K^{(r)}]$ denote draw *r*, *r* = 1, ..., *R*, from the corresponding prior random distributions described by the probability density functions $\Phi_k(\tilde{\beta}_k \mid \Theta_k)$. The approximation of the efficiency-error can be formalized as

$$E(f) \approx \frac{1}{R}\sum_{r=1}^{R} f(\tilde{\beta}^{(r)} \mid X). \qquad (7.11)$$

The total number of *efficiency* evaluations is equal to *R*. In order to determine the draws $\tilde{\beta}_k^{(r)}$, we let the computer generate for each parameter *R* pseudo-random numbers $u_k^{(r)}$ which are uniformly distributed on the interval [0,1], and then compute the draws by

$$\tilde{\beta}_k^{(r)} = \Phi_k^{-1}\left(u_k^{(r)}\right), \qquad (7.12)$$

where $\Phi(\tilde{\beta}_k \mid \Theta_k)$ denotes the cumulative distribution function corresponding to the probability density function $\Phi_k(\tilde{\beta}_k \mid \Theta_k)$.

### 7.1.5.2 Quasi-random Monte Carlo simulation

Randomness of the draws is not a prerequisite in the approximation of the integral; rather, Winiarski (2003) has argued that (a) correlation or a systematic structure between draws for different dimensions can have a positive effect on the approximation, and (b) one should aim for the draws to be distributed as uniformly as possible over the area of integration. Hence, the draws can be selected deterministically so as to minimize the integration error, which is exactly what quasi-random MC simulation methods aim to do. For a more detailed discussion on these methods we refer to Niederreiter (1992) and Fang and Wang (1994). Quasi-random MC simulation methods for approximating say the $D_b$-error are almost identical to the PMC simulation method, except that they use deterministic draws for $\tilde{\beta}_k^{(r)}$ (as opposed to purely random draws). Instead of generating pseudo-random numbers $u_k^{(r)} \sim U(0,1)$, these numbers $u_k^{(r)}$ are taken from different intelligent quasi-random sequences, also called low discrepancy sequences. Using these quasi-random sequences, faster convergence to the true value of the numerical integration can be achieved. PMC simulation has a slow rate of convergence of $O(1/\sqrt{R})$, while quasi-random MC simulation typically has a rate of convergence as good as $O(1/R)$. [6]

#### 7.1.5.3 Modified Latin Hypercube Sampling (MLHS)

The MLHS method (Hess *et al*. 2005) produces multi-dimensional sequences by combining randomly shuffled versions of one-dimensional sequences made up of uniformly spaced points. Formally, the individual one-dimensional sequences of length *R* are constructed as:

$$u_k^{(r)} = \frac{r-1}{R} + \xi_k, \quad r = 1,\ldots,R, \qquad (7.13)$$

where $\xi_k$ is a random number drawn between 0 and 1/*R*, and where a different random draw is used in each of the *K* different dimensions. In the resulting sequence, the distances between adjacent draws are all equal to 1/*R*, satisfying the condition of equal spacing. Multi-dimensional sequences are constructed by simple combination of randomly shuffled one-dimensional sequences, where the shuffling disrupts the correlation between individual dimensions.

#### 7.1.5.4 Halton sequences

Halton sequences (Halton 1960) are based on the one-dimensional Van der Corput sequence (Van der Corput, 1935) and are constructed according to a deterministic method based on the use of prime numbers, dividing the 0-1 space into $p_k$ segments (with $p_k$ giving the prime used as the base for parameter *k*), and by systematically filling in the empty spaces, using cycles of length $p_k$ that place one draw in each segment. Formally, the $r^{\text{th}}$ element in the Halton sequence based on prime $p_k$ is obtained by taking the radical inverse of integer r in base $p_k$ by reflection through the radical point, such that

$$r = \sum_{\ell=0}^{L} b_\ell^{(r)} p_k^{\ell}, \qquad (7.14)$$

where $0 \le b_\ell^{(r)} \le p_k - 1$ determines the *L* digits used in base $p_k$ in order to represent *r* (i.e., solving equation (7.14)), and where the range for *L* is determined by $p_k^L \le r < p_k^{L+1}$. The draw is then obtained as:[7]

$$u_k^{(r)} = \sum_{\ell=0}^{L} b_\ell^{(r)} p_k^{-\ell-1}. \qquad (7.15)$$

To allow for the computation of a simulation error, the deterministic Halton sequence can be randomized in several ways. Here, we use the approach discussed by amongst others Tuffin (1996), where the modified draws are obtained by adding a random draw $\xi_k$ to the individual draws in dimension *k*, and by subtracting one from any draws that now fall outside the 0-1 interval. A different random draw is used for each dimension.

### 7.1.5.5 Sobol sequences

The main problem with Halton sequences is the fact that the individual sequences are highly correlated, leading to problems with poor multi-dimensional coverage in higher dimensions. Aside from various transformations of the standard Halton sequence and other advanced methods (cf. Hess *et al*. 2005), one approach that has received exposure in the area of discrete choice modeling is the Sobol sequence, used amongst others by Garrido (2003). Like Halton sequences, Sobol sequences are based on Van der Corput sequences (cf. Niederreiter 1992). However, rather than in a *K*-dimensional problem using the first *K* primes (as in Halton sequences), Sobol sequences are based on prime 2 in each dimension, where different permutations are used to ensure that the resulting *K*-dimensional sequence obtains good coverage. We will use a randomized version of the Sobol sequences equivalent to the randomization in the Halton sequences by adding a random component to each of the draws in each dimension.

### 7.1.5.6 Gaussian quadrature

Polynomial cubature methods aim to approximate integrals using orthogonal polynomials. Gaussian quadrature is the best-known method, see e.g. Stoer and Bulirsch (2002). In case of a single variable, the use of *R* draws yields an exact approximation if the integrand is a polynomial up to degree (2*R*-1). General functions can be approximated by (high order) polynomials, hence the higher the degree (yielding more draws), the more accurate the approximation will be.

The principle of Gaussian quadrature is that not only the draws $\tilde{\beta}_k^{(r)}$ for the priors are selected intelligently, but also that weights $w_k^{(r)}$ are associated with each draw. The approximation of the D *b*-error using Gaussian quadrature can be formalized as

$$E(f) \approx \sum_{r_1=1}^{R_1} \cdots \sum_{r_K=1}^{R_K} w_1^{(r_1)} \cdots w_K^{(r_K)} f(\tilde{\beta}_1^{(r_1)}, \ldots, \tilde{\beta}_K^{(r_K)} \mid X). \qquad (7.16)$$

The draws for the priors and the associated weights depend on the random distribution. Different draws $\tilde{\beta}_k^{(r)}$ for each individual parameter are called abscissas. The draws for the whole vector $\tilde{\beta}^{(r)}$ are given by a rectangular grid of these abscissas[8]. In the case where $\tilde{\beta}_k \sim N(\mu_k, \sigma_k)$, the abscissas and weights can be computed using so-called Hermite polynomials. If $\tilde{\beta}_k \sim U(a_k, b_k)$, the abscissas and weights can be computed using so-called Legendre polynomials. The abscissas and weights for both situations are listed in Table 1 for up to 10 abscissas for each individual parameter. The weights always sum up to one, i.e., $\sum_{r=1}^{R} w_k^{(r)} = 1$ for each k. For each of the *K* parameters, the number of abscissas used, $R_k$, can be different.

Note that the total number of D-error evaluations in Gaussian quadrature is equal to $R = \prod_{k=1}^{K} R_k$, that is, the total number of all combinations of abscissas in all dimensions. This number of D-error evaluations grows exponentially if the number of random priors increases[9]. Therefore, Gaussian quadrature is typically not suitable for integrals of high dimensionality, although it is extremely powerful for low-dimensional problems.

### 7.1.6 Orthogonal versus efficient designs

In case any information about the parameters is available, then efficient designs will always outperform orthogonal designs. This is due to the fact that efficient designs use the knowledge of the prior parameters to optimize the design in which the most information is gained from each choice situation (e.g., dominant alternatives can be avoided as the utilities can be computed). We will come back to dominant alternatives when discussing the (un)importance of utility balancing in Section 7.1.8.

What happens in the case where no information about the parameters is available? In other words, which design is better, an orthogonal design, or a $D_z$-optimal design (which assumes $\tilde{\beta}$ =0)? As mentioned in Bliemer and Rose (2005b), there is a close correspondence between orthogonal designs and $D_z$-optimal designs. In fact, in case all model parameters are alternative-specific, a $D_z$-optimal design is orthogonal. In case all model parameters are generic, it is not necessary to choose between either orthogonality or $D_z$-efficiency as it is possible to determine orthogonal $D_z$-optimal designs. Street *et al*. (2001), Street and Burgess (2004) demonstrate how to create such $D_z$-optimal designs for generic designs with only two alternatives and where each attribute has a number of levels equal to the power of two (hence, two, four, eight, etc.). In Street *et al*. (2005) a nice overview is given for determining $D_z$-optimal (or nearly optimal) designs with multiple alternatives and different levels. However, these remain limited to models with generic parameters.

The design principles in Street *et al*. (2005) have some limitations. First of all, they are limited to the MNL model. Secondly, they are only optimal in case all parameters are equal to zero, which is clearly not the case. The fact that their designs are sub-optimal under the nonzero parameter case is due to the fact that they assume all equal probabilities in the MNL model. Finally, if alternative-specific parameters are present, then a simple principle that will lead to a $D_z$-optimal design does not exist.

If correlations in the design have a negative impact on the parameter estimates, then this should implicitly be reflected in the AVC matrix of the design, instead of explicitly in an orthogonal design. Hence, an efficient design will to a certain degree implicitly minimize the correlations in a design, hence it is not necessary to include orthogonality as an additional criteria to efficiency.

### 7.1.7 Importance of prior parameter values

The purpose of the SC experiment is to estimate the parameters of the specified model. But even without estimating them, some information and/or educated guesses regarding parameters are usually available. Again, we would like to stress that $D_p$-optimal designs will always outperform $D_z$-optimal designs in case any information about the parameters (even only the sign of the parameters) is available. We argue that it is always possible to obtain some information on the priors.

Just using reasoning alone, it should be possible to determine at minimum the signs of most parameters. For example, price attributes are typically negatively perceived, while comfort and service are attributes that will receive positive attitudes. Instead of using a prior parameter value equal to zero, already a slight positive or negative value would already improve the design.

Many surveys have been conducted around the world, and it is likely to find at least a few similar parameters. If no such studies can be found, then it may be very useful to conduct a small pilot study in order to get an initial idea about the parameter values. With the same amount of money, one could (i) conduct a large survey using an experimental design based on priors equal to zero (no information case), or (ii) conduct a slightly smaller survey using an experimental design based on priors obtained from a pilot study. As Figure 7.1 also suggested, the second option is preferred, since it can lead to significantly more reliable parameter estimates.

Obviously, a $\text{D}_p$-optimal design is sensitive to the chosen prior parameters. If they are not correct, then the design is sub-optimal (note that $\text{D}_z$-optimal will therefore always be sub-optimal). Fortunately, the design can be tested for robustness in case one or more prior parameter values are not correct. By taking a fixed design *X* and computing the AVC matrix as in Equation (7.1) (recall that the outcomes *Y* drop out) for different values of $\tilde{\beta}$, a sensitivity analyses of the design can be performed. Once the sensitivity of the efficiency of the design to each prior parameter is known, one can decide to either put more effort in determining the prior values for the most sensitive priors, or determine a new design (which may be less efficient, but more robust).

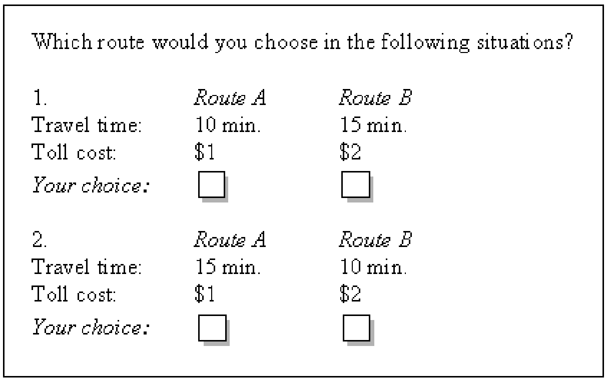
Another way of dealing with uncertainty about prior parameters was already mentioned when describing the Bayesian efficient designs. A Bayesian efficient design optimizes the expected efficiency of the design over a range of prior parameter values, thereby making it more robust to mis-specifying the priors. Priors with a higher uncertainty should see this uncertainty reflected into a larger standard deviation or spread of its probability distribution.

### 7.1.8 Utility balance

A couple of times the words “dominant alternatives” or “more information from choice situations” have been used. Here the concept of utility balancing, as suggested in (Huber and Zwerina, 1996) will be described.

As a simple example, consider two choice situations in an unlabelled stated choice experiment as illustrated in Figure 7.2. In the first choice situation, Route A has both a lower travel time as well as a lower toll cost, making it clearly the preferred alternative. The Route A alternative therefore clearly dominates in this choice situation, therefore no information will be gained. In contrast, in the second choice situation there is no clear dominant alternative and the respondent has to make a clear trade-off between travel time and toll cost, hence this will provide information.

The example illustrates that balancing the utilities of alternatives (i.e., having no alternatives that are clearly dominating the others) is of importance. At least, if it is very unbalanced, the choice situation does not provide information for estimating the parameters. This could lead to the understanding that in the most efficient design, all the choice situations are perfectly utility balanced. This is however not the case. If all alternatives have an equal observed utility, then the random unobserved component dominates. In other words, then the respondent has no clear preference for any of the alternatives and randomly selects one. This too does not give information. Therefore it can be concluded that an efficient design has some degree of utility balance, but not too much, and not too little.



Which route would you choose in the following situations?

| | | |
|---|---|---|
| 1. | *Route A* | *Route B* |
| Travel time: | 10 min. | 15 min. |
| Toll cost: | $1 | $2 |
| *Your choice:* | □ | □ |
| 2. | *Route A* | *Route B* |
| Travel time: | 15 min. | 10 min. |
| Toll cost: | $1 | $2 |
| *Your choice:* | □ | □ |

**Figure 7.2: Dominant alternative in choice situation 1**

Utility balance of a choice situation and a whole design can be expressed in a percentage. Consider a stated choice experiment with *J* alternatives. Consider a certain choice situation *s*. This choice situation would have perfect utility balance if all alternatives *j* have an equal probability, that is, $P_{js}$ = 1 / *J*. The utility balance of choice situation s can be defined as

$$B_s = \prod_{j=1}^{J} \left( \frac{P_{js}}{1/J} \right) \times 100\%. \tag{7.17}$$

For example, if *J* = 3 and all three alternatives have a probability of 1/3, then $B_s$ = 100%. If the probabilities are 1/2, 1/3, and 1/6, respectively, then the utility balance is $B_s$ = 75%. If one or more of the probabilities is equal to zero, then the utility balance is zero percent. The overall utility balance of the design, *B*, can be determined by averaging over all choice situations (Kessels et al., 2006):

$$B = \frac{1}{S} \sum_{s=1}^{S} B_s. \tag{7.18}$$

The optimal value for utility balance of a design cannot be given, but observations of the utility balance of efficient designs suggest that it lies in the range of 70-90 percent. Utility balance can be examined for each choice situation, thereby investigating if the design contains choice situations with clearly dominant alternatives, which should not occur in an efficient design. Hence, utility balance could be used in the algorithms for generating efficient designs. In Ngene, we refer to attempts to maximize utility balance as *B*-error. Similar to the *D*-, *A*-, *S*- and *C*-error measures, *B*-error may be implemented using either zero, fixed or Bayesian priors.

### 7.1.9 Generating efficient designs

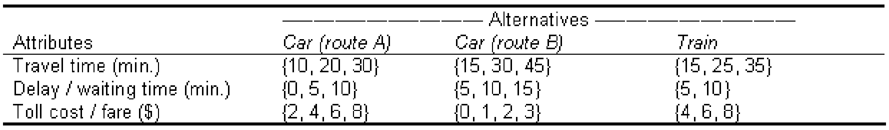
The problem of finding an efficient design can be described as follows:

> Given feasible attribute levels $\Lambda_{jk}$ for all $j$ and $k$, given the number of choice situations $S$, and given the prior parameter values $\tilde{\beta}$ (or probability distributions of $\tilde{\beta}$), determine a level balanced design $X$ with $x_{jks} \in \Lambda_{jk}$ that minimizes the efficiency error in Equations (7.4), (7.5), (7.6), (7.7), (7.9) or (7.10).

Note that in this formulation attribute level balance is added as a requirement, consistent with current state of practice. It should be stressed that an efficient design does not necessarily require attribute level balance. In fact, a more efficient design may be found by removing the level balance requirement as will be discussed in Section 8.1.

In order to solve the problem of determining the most efficient design, one could determine the full factorial design and then evaluate each different combination of $S$ choice situations from this full factorial. The combination with the lowest efficiency error is the optimal design. However, this procedure is not feasible in practice due to an extremely high number of possible designs to evaluate. For example, consider the problem of determining an efficient design for a hypothetical case with three alternatives as shown in Table 7.1. The full factorial design has $2^1$ x $3^8$ x $4^2$ = 209,952 choice situations. Suppose that we would like to find an efficient design with $S$ = 12 choice situations. Selecting 12 choice situations from this set of 209,952 different choice situations yields 7.3 x $10^{63}$ possible different designs. Clearly, it is not feasible to evaluate all possible designs, hence a smart algorithm is necessary to find an efficient as possible design.

**Table 7.1: Example dimensions for generating an efficient design**



| | Alternatives | | |
|---|---|---|---|
| Attributes | *Car (route A)* | *Car (route B)* | *Train* |
| Travel time (min.) | {10, 20, 30} | {15, 30, 45} | {15, 25, 35} |
| Delay / waiting time (min.) | {0, 5, 10} | {5, 10, 15} | {5, 10} |
| Toll cost / fare ($) | {2, 4, 6, 8} | {0, 1, 2, 3} | {4, 6, 8} |

There are row based algorithms and column based algorithms for finding an efficient design. In a *row based algorithm* choice situations are selected from a predefined candidate set of choice situations (either a full factorial or a fractional factorial) in each iteration. *Column based algorithms* (such as RSC algorithms) create a design by selecting attribute levels over all choice situations for each attribute. Row based algorithms can easily remove bad choice situations from the candidate set at the beginning (e.g., by applying a utility balance criterion), but it is more difficult to satisfy attribute level balance. The opposite holds for column based algorithms, in which attribute level balance is easy to satisfy, but finding good combinations of attribute levels in each choice situation is more difficult. In general column based algorithms offer more flexibility and can deal with larger designs, but in some cases (for unlabelled designs and for specific designs such as constrained designs, see Section 8.2) row based algorithms are more suitable.

The *Modified Federov algorithm* (Cook and Nachtsheim, 1980) is a row based algorithm and is illustrated in Figure 7.3. First, a candidate set is determined, either the full factorial (for small problems), or a fractional factorial (for large problems). Then, a (attribute level balanced) design is created by selecting choice situations from the candidate set. After that, the efficiency error (e.g., D-error) is computed for this design. Finally, if this design has a lower efficiency error than the current best design, the design is stored as the most efficient design so far, and one continues with the next iteration repeating the whole process again. The algorithm terminates if all possible combinations of choice situations have been evaluated (which is in general not feasible), or after a predefined number of iterations. Construction of $D_z$-optimal as described in Street *et al.* (2005) is also row based, in which in a smart way combinations of choice situations are made.

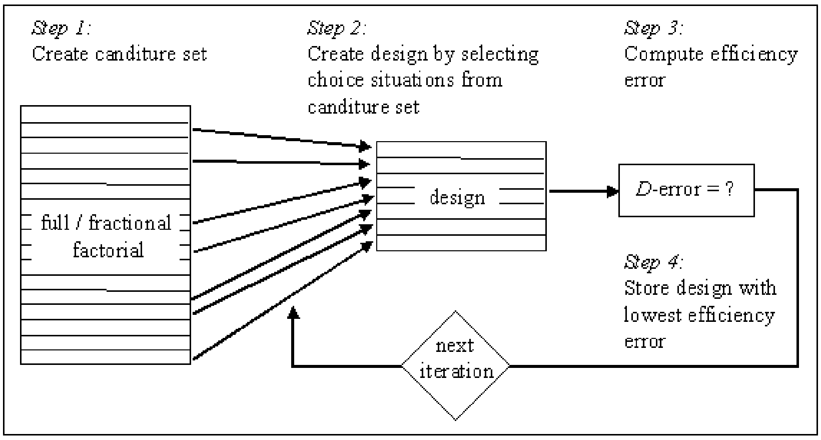

Step 1:
Create canditure set
Step 2:
Create design by selecting choice situations from canditure set
Step 3:
Compute efficiency error
full / fractional factorial
design
D-error = ?
Step 4:
Store design with lowest efficiency error
next iteration

**Figure 7.3: Modified Federov algorithm**

*RSC (Relabeling, Swapping & Cycling) algorithms* (Huber and Zwerina, 1996; Sándor and Wedel, 2001) are column based algorithms, illustrated in Figure 7.4. In each iteration, different columns for each attribute are created, which together form a design. This design is evaluated and if it has a lower efficiency error than the current best design, then it is stored. The columns are not created randomly, but – as the name suggests – are generated in a structured way using relabeling, swapping, and cycling techniques. Starting with an initial design, each column could be altered by relabeling the attribute levels. For example, if the attribute levels 1 and 3 are relabeled, then a column containing the levels (1,2,1,3,2,3) will become (3,2,3,1,2,1). Swapping means that some attribute levels switch place, for example if the attribute levels in the first and fourth choice situation are swapped, then (1,2,1,3,2,3) would become (3,2,1,1,2,3). Finally, cycling replaces all attribute levels in each choice situation at the time by replacing the first attribute level with the second level, the second level with the third, etc. Since this impacts all columns, cycling can only be performed if all attributes have exactly the same sets of feasible levels (e.g., in case all variables are dummy coded). Sometimes only swapping is used, sometimes only relabeling and swapping is used, as special cases of this algorithm type.

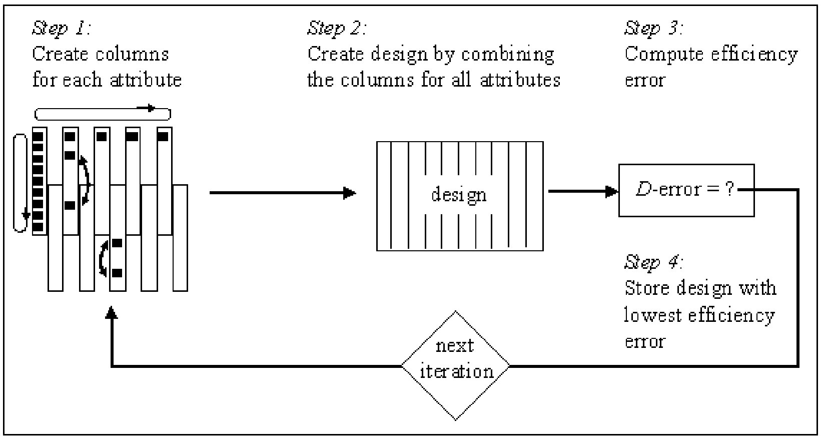

Step 1:
Create columns for each attribute
Step 2:
Create design by combining the columns for all attributes
Step 3:
Compute efficiency error
design
D-error = ?
Step 4:
Store design with lowest efficiency error
next iteration

**Figure 7.4: RSC algorithm**

If for some reason orthogonality is required in a $D_p$-efficient design, one could construct a single orthogonal design, from this design easily create a large (but not huge) number of other orthogonal designs, and then evaluate all these orthogonal designs and select the most efficient one. Creating other orthogonal designs from a single orthogonal design is relatively simple, as discussed in Section 6.1.3.

Evaluating each design for the efficiency error is the most time-consuming part of each algorithm, therefore the number of D-error or other efficiency error evaluations should be kept to a minimum by putting more intelligence into the construction of the designs. In determining Bayesian efficient designs this becomes even more important, as the integral in Equation (7.6) cannot be computed analytically, but only by simulation. Mainly pseudo-random Monte Carlo simulations have been performed for determining the Bayesian D-error for each design, which enables the approximation of this D-error by taking the average of all D-errors for the same design using pseudo-random draws for the prior parameter values. This is clearly a computation intensive process, such that finding Bayesian efficient designs is a very time consuming task. Bliemer *et al*. (2006) have proposed to use quasi-random draws (such as Halton or Sobol sequences) or preferably Gaussian quadrature methods instead of pseudo-random draws, which require less simulations and therefore enable the evaluation of more designs in the same amount of time.

For further information on generating efficient designs, see Appendix 7B.

### 7.1.10 Discussion of efficient designs

Efficient or optimal designs have been embraced by more and more researchers as the current best way of designing SC experiments. Practitioners are still somewhat hesitant to deviate from orthogonal designs which have been claimed to be best for a long time, but there is a growing support for such designs both in practice and within the academic literature.

Do the chosen feasible levels, determined before generating an efficient design, impact the potential efficiency of the design? The answer is ‘yes’, they have a significant impact on the efficiency. Broadly speaking, the less attribute levels and, more importantly, the wider the attribute level range, the higher the efficiency of the design can be. A wide attribute level range usually translates into smaller asymptotic standard errors. Therefore, the highest efficiency can theoretically be obtained using so-called *end-point designs*, which are two-level designs with extreme (wide range) levels. The disadvantage of this kind of designs is, that nonlinearities cannot be estimated (more levels are needed for this purpose). Furthermore, the extreme levels should be realistic.

The number of choice situations does not seem to have a large impact on the efficiency of a design, as long as the number of choice situations is not smaller than $K/(J\text{-}1)$. Clearly, more choice situations yield more data per respondent, hence the efficiency will automatically increase with more choice situations. Compensating for this effect by normalizing the efficiency error (i.e., assuming the same amount of data), it does not seem to make much difference how many choice situations are chosen. Therefore, the number of choice situations does not have to be very high (blocking as in orthogonal designs is therefore not necessary) and should mainly depend on the intuition how many choice situations respondents can handle. A higher number of choice situations means a higher task effort for the respondent. The maximum number of choice situations depends of course on the complexity of each choice situation, but roughly 10 to 20 choice situations should be possible.

Still, the efficient designs discussed in this section may be improved due to the somewhat restrictive assumptions commonly imposed. First of all, attribute level balanced has been imposed for efficient designs, which is typically only required for orthogonal designs. Attribute level balance is viewed as a desired property ensuring that all attribute levels appear equally in the data set, which intuitively provides a good basis for estimation. However, the attribute level balance requirement is mathematically speaking merely imposing another constraint on the problem of minimizing the efficiency error, thereby always leading to less efficient designs. By relaxing this assumption a more efficient design can be found (at least it is never less efficient). An optimal level unbalanced design is likely to be (close to) an end-point design using just the two extreme levels.

Another assumption made in this section is the assumption of independent observations, i.e., the outcomes of all choice situations from the same respondent are assumed independent. This assumption makes it easy to derive analytical expressions of the AVC matrix. However, it is likely that the data does not consist of independent observations, as the random unobserved utilities are correlated within each respondent, and this has to be taken into account. Using an error components structure one could simulate these correlations, but then the AVC matrix has to be computed by simulation instead of analytically, see Scarpa *et al*. (2005) and Ferrini and Scarpa (2006).

Some other assumptions were that all respondents face the same choice situations, and that socio-economic data is ignored. These assumptions are relaxed in the next section. Instead of relaxing some assumptions, it is also discussed how to deal with more constraints.

## 7.2 Generating efficient designs in Ngene

In contrast to orthogonal designs, more information on the model type and prior parameter values is needed when dealing with efficient designs. In the following, different syntax commands will be discussed for different model types. We will describe syntax for the MNL, the MMNL and the EC models.

### 7.2.1 Efficiency measures

All model types share that the efficiency measure to be optimized on has to be set. In Ngene this can be done using the property *eff*, by defining the model type together with the efficiency measure required for optimisation. For example, using the D-error measure for finding an efficient design for the MNL model, the following property is added to the syntax:

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

where the first part between brackets refers to the model type, while the second refers to the efficiency measure. Other examples include

```
;eff = (ecpanel,d)
;eff =(ec,a)
;eff = (rp,s)
;eff = (rppanel,b)
```

where mnl refers to the multinomial logit model, ‘rp’ and ‘rppanel’ refer to the MMNL cross sectional and panel models, and ‘ec’ and ‘ecpanel’ to the EC cross sectional and panel models. Furthermore, instead of using D-error, the A-error is minimised when the second argument is set to ‘a’, the sample size is minimised when set to ‘s’, and the utility balance of the design maximized when it is set to ‘b’.

For designs that are to be optimized based on the variance of the ratio of two or more parameters (WTP designs), the optimization routine allows for up to *k*-1 ratios to be specified. The routine then attempts to minimize the sum of the variances of the indicated parameter ratios. In this way, not all ratios need be used in the optimization routine. As such, when using the ‘wtp’ property, the user is required to also specify what parameter ratios to use in the calculation. This is handled via a separate *wtp* property in conjunction with the *eff* property, as shown below.

```
;eff = (mnl,wtp(ref1))
```

```
;wtp = ref1(*/b1)
```

where the ‘wtp’ argument in the *eff* property indicates that C-error efficiency measure is to be used (see Section 7.1.4) and ref1 links the *eff* property to the *wtp* property. For the *wtp* property the user first specifies a name (any name will suffice; we have used ref1 here for demonstrative purposes only) followed by what parameter ratios to use in the efficiency calculation. If an asterisk (i.e., *) is used, as in the example provided, then all parameters specified in the models utility functions will be used as numerators in the calculation, save for the parameter indicated after the back slash or divide symbol (i.e., /). The parameter named after the back slash represents the parameter (usually a cost parameter) that will be used as the denominator in the WTP calculations. Rather than use all parameters in the calculation, it is also possible to specify only a subset of parameters as shown in the following example.

```
;eff = (mnl,wtp(wtp1))
;wtp = wtp1(b2,b3,b5/b1)
```

In this case, the C-error measure will only be calculated using the sum of the variances of the ratios of the b2, b3 and b5 parameters to the b1 parameter. Any other parameters specified in the models utility specification will not form part of the calculation. Note that any name can be used in specifying the WTP measure with the name WTP1 now being substituted for the name ref1 used previously. Note also that the WTP property will only work for the MNL model.

Efficient designs are in general not orthogonal. However, it is possible to generate an efficient orthogonal design by adding the *orth* property as described before. Ngene will then search for the most efficient design that is orthogonal.

Although efficient designs typically require less choice situations than an orthogonal design, the number of choice situations may still be too large to give to a single respondent. Similar to creating blocks for orthogonal designs, add the *block* property to the syntax, and Ngene will block the design automatically based on the minimum correlation principle. Basically, it will try to minimize the average correlation between the blocking column and all other design columns.

It is possible that one would like to have a no-choice alternative (the option of not choosing any of the other alternatives). This alternative does not have a utility function (so the utility is basically set to zero for that alternative), but it does affect the choice probabilities and therefore the efficiency of the design. In case one would like to add a no-choice alternative, this alternative should be added in the *alts* property, but not have a utility function in the *model* property. Ngene automatically recognizes the alternative without a utility function as a no-choice alternative.

The definition of the *model* property is different for each of the model types, hence they will be discussed separately in the following.

## 7.2.2 Designs for estimating multinomial logit models

The multinomial logit (MNL) model is the basic logit model with fixed parameters. Prior parameters need to be specified for each fixed parameter, and this is done by adding them between square brackets in the *model* property behind the parameter names. For example, the syntax may look as follows:

```
Design
;alts = alt1, alt2
;rows = 12
;eff = (mnl,d)
;model:
```

```
U(alt1) = b1[-0.2] + b2[1.2] * A[0,1,2] + b3[2.5] * B[0,1]     /
U(alt2) =            b2      * A        + b4[1.1] * C[2,4,6,8]
$
```

When creating an efficient design, each parameter must have a prior value associated with it. For generic parameters like ‘b2’, the prior value should only be defined the first time the parameter appears and should not be defined again when the same parameter appears in another utility function.

Attribute levels can be specified in an alternative way, with a lower and upper bound, and a step size. These three values are specified in sequence inside the square brackets, separated by a colon. Using this syntax, the above example would be:

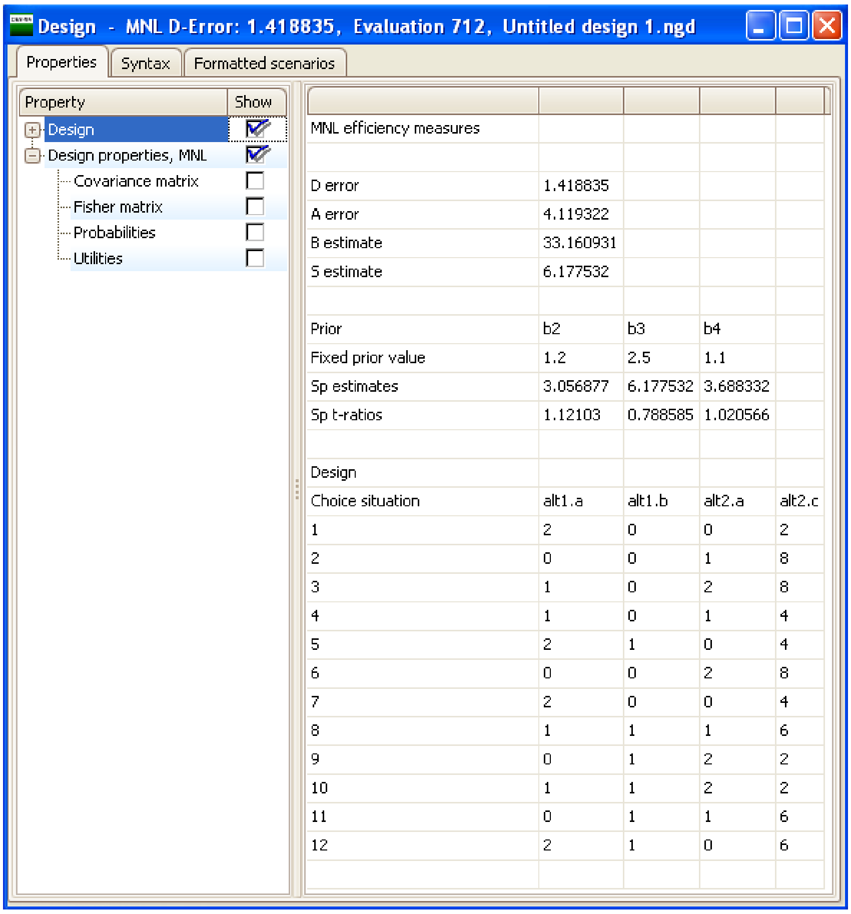
```
Design
;alts = alt1, alt2
;rows = 12
;eff = (mnl,d)
;model:
U(alt1) = b1[-0.2] + b2[1.2] * A[0:2:1] + b3[2.5] * B[0:1:1]     /
U(alt2) =            b2      * A        + b4[1.1] * C[2:8:2]
$
```

Figure 7.5 shows example output based on a design generated using the above syntax. In presenting the output for efficient designs, Ngene always first presents information on the efficiency measures related to the design, independent of what measure was used during optimization of the design. In the example output, the generated design has a D*p*-error of 1.42, A*p* -error of 4.12, a B*p*-error of 33.12 percent and an S*p*-error of  6.18(suggesting that given the priors assumed, the design would need to be replicated at least 6.18 times for all parameters to be statistically significant with a *t*-ratio of at least 1.96). Note that these values assume that the prior parameters assumed are correct.

Note that in calculating the efficiency measures, Ngene ignores any constants. The constant is typically ignored in these kind of studies, since usually the constant is of less importance to the researcher (indeed the constant is often considered meaningless in SC experiments as it is based on the choice shares over the hypothetical situations, *S*). Further, in many SC studies, it is often the ratios of two parameter values (e.g., to derive willingness to pay) that is of primary importance. Therefore, in calculating the efficiency for each design, we ignore the row and column for the constant in the AVC matrix when computing the efficiency statistic. If one wishes to include the constant in these calculations, then the con property can be added to the syntax. That is,

```
;con
```

Underneath the efficiency measures, more detailed information related to the S*p* -error is presented for each (non-constant) parameter estimate of the design. The parameter estimate priors and S*p* -errors for each of the parameters are presented here, as too are the expected *t*-ratios for each of the parameters assuming only a single replication of the design were to be used in practice. The last item of output presented automatically is the design itself.


Design - MNL D-Error: 1.418835, Evaluation 712, Untitled design 1.ngd
Properties
Syntax
Formatted scenarios

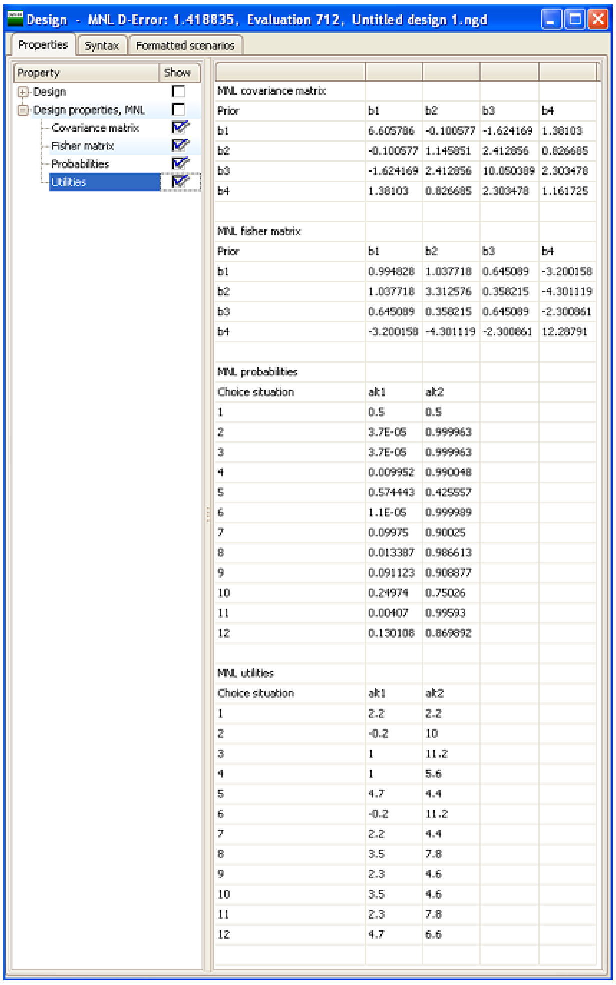
| Property | Show |
|---|---|
| Design | ☑ |
| Design properties, MNL | ☑ |
| Covariance matrix | □ |
| Fisher matrix | □ |
| Probabilities | □ |
| Utilities | □ |

| | | | | |
|---|---|---|---|---|
| MNL efficiency measures | | | | |
| | | | | |
| D error | 1.418835 | | | |
| A error | 4.119322 | | | |
| B estimate | 33.160931 | | | |
| S estimate | 6.177532 | | | |
| | | | | |
| Prior | b2 | b3 | b4 | |
| Fixed prior value | 1.2 | 2.5 | 1.1 | |
| Sp estimates | 3.056877 | 6.177532 | 3.688332 | |
| Sp t-ratios | 1.12103 | 0.788585 | 1.020566 | |
| | | | | |
| Design | | | | |
| Choice situation | alt1.a | alt1.b | alt2.a | alt2.c |
| 1 | 2 | 0 | 0 | 2 |
| 2 | 0 | 0 | 1 | 8 |
| 3 | 1 | 0 | 2 | 8 |
| 4 | 1 | 0 | 1 | 4 |
| 5 | 2 | 1 | 0 | 4 |
| 6 | 0 | 0 | 2 | 8 |
| 7 | 2 | 0 | 0 | 4 |
| 8 | 1 | 1 | 1 | 6 |
| 9 | 0 | 1 | 2 | 2 |
| 10 | 1 | 1 | 2 | 2 |
| 11 | 0 | 1 | 1 | 6 |
| 12 | 2 | 1 | 0 | 6 |

**Figure 7.5: MNL design output**

Additional output is also available by clicking on the check boxes located in the tree structure at the left of the output screen. Figure 7.6 shows the additional output available for designs generated assuming an MNL model formulation. Available to the researcher are the Fisher Information matrix, AVC matrix, the utility estimates and choice probabilities for each choice task contained within the design. The Fisher Information and AVC matrices are generated assuming a single design replication. The utilities and choice probabilities are often useful for diagnostic purposes. Examination of these outputs will often allow the user to observe if one alternative will tend to dominate others within the design, in which case large efficiency measures (and/or small B-error values) will generally be observed indicating difficulty in locating an efficient design. In the example output shown in Figure 7.6, examination of the choice probabilities shows that the second alternative will tend to dominate the first in most (but not all) choice situations. Should other attribute levels be used or different priors be assumed, then it might be possible to locate a more efficient design. Note however that assuming a different set of priors in generating the design  might not be ideal given that the priors assumed generally have to come from some other source (such as a pilot survey), and hence disregarding these and simply assuming another set of priors for the sake of statistical efficiency may have no scientifically valid basis.

Although not shown here, Ngene can calculate the correlation structure of the design in the same manner as with orthogonal designs. The various correlation measures are located in the tree structure to the left of the design output under the ‘Design’ button. Clicking on any of the correlation click boxes will have Ngene generate and show the requested correlation structure for the design.



Design - MNL D-Error: 1.418835, Evaluation 712, Untitled design 1.ngd

| Property | Show |
|---|---|
| Design | □ |
| Design properties, MNL | □ |
| Covariance matrix | ☑ |
| Fisher matrix | ☑ |
| Probabilities | ☑ |
| Utilities | ☑ |

| | | | | |
|---|---|---|---|---|
| MNL covariance matrix | | | | |
| Prior | b1 | b2 | b3 | b4 |
| b1 | 6.605786 | -0.100577 | -1.624169 | 1.38103 |
| b2 | -0.100577 | 1.145851 | 2.412856 | 0.826685 |
| b3 | -1.624169 | 2.412856 | 10.050389 | 2.303478 |
| b4 | 1.38103 | 0.826685 | 2.303478 | 1.161725 |
| | | | | |
| MNL fisher matrix | | | | |
| Prior | b1 | b2 | b3 | b4 |
| b1 | 0.994828 | 1.037718 | 0.645089 | -3.200158 |
| b2 | 1.037718 | 3.312576 | 0.358215 | -4.301119 |
| b3 | 0.645089 | 0.358215 | 0.645089 | -2.300861 |
| b4 | -3.200158 | -4.301119 | -2.300861 | 12.28791 |
| | | | | |
| MNL probabilities | | | | |
| Choice situation | alt1 | alt2 | | |
| 1 | 0.5 | 0.5 | | |
| 2 | 3.7E-05 | 0.999963 | | |
| 3 | 3.7E-05 | 0.999963 | | |
| 4 | 0.009952 | 0.990048 | | |
| 5 | 0.574443 | 0.425557 | | |
| 6 | 1.1E-05 | 0.999989 | | |
| 7 | 0.09975 | 0.90025 | | |
| 8 | 0.013387 | 0.986613 | | |
| 9 | 0.091123 | 0.908877 | | |
| 10 | 0.24974 | 0.75026 | | |
| 11 | 0.00407 | 0.99593 | | |
| 12 | 0.130108 | 0.869892 | | |
| | | | | |
| MNL utilities | | | | |
| Choice situation | alt1 | alt2 | | |
| 1 | 2.2 | 2.2 | | |
| 2 | -0.2 | 10 | | |
| 3 | 1 | 11.2 | | |
| 4 | 1 | 5.6 | | |
| 5 | 4.7 | 4.4 | | |
| 6 | -0.2 | 11.2 | | |
| 7 | 2.2 | 4.4 | | |
| 8 | 3.5 | 7.8 | | |
| 9 | 2.3 | 4.6 | | |
| 10 | 3.5 | 4.6 | | |
| 11 | 2.3 | 7.8 | | |
| 12 | 4.7 | 6.6 | | |

**Figure 7.6: Additional MNL design output**

## 7.2.3 Designs for estimating random parameters models

It is strongly advised to first generate a non-Bayesian design with the MNL model. This allows problems to be much more quickly identified. For example, the priors might lead to extreme choice probabilities of zero and one, and may need to be adjusted. An MNL design should always be generated quickly, so if it is not, then you know there is a problem with your design specification. Random parameter and Bayesian models are much slower to generate, and it may not be clear for some time that there is a problem with the design. A good principle to follow is: start simple, and gradually add complexity to the design.

In the mixed multinomial logit (MMNL) model, the parameters are assumed to be random instead of fixed as in the MNL model. Therefore, the parameters in the *model* property need to be defined as distributions. For example, suppose that parameter ‘b2’ is assumed to be normally distributed with a mean of 1.2 and a standard deviation of 0.3 (1.2 and 0.3 are now prior parameter values for the random parameter distribution), and suppose that ‘b4’ is uniformly distributed between 0.5 and 1.5, then the following syntax could be used:

```
? Cross sectional RP model
Design
```

```
;alts = alt1, alt2
;rows = 12
;eff = (rp,d)
;model:
U(alt1) = b1[-0.2] + b2[n,1.2,0.3] * A[0,1,2] + b3[2.5] * B[0,1]
   /
U(alt2) =              b2              * A          + b4[u,0.5,1.5] * C
[2,4,6,8]
$
```

Hence, [n,1.2,0.3] indicates a normal distribution with mean 1.2 and standard deviation 0.3, while [u,0.5,1.5] indicates a uniform distribution between 0.5 and 1.5.

For generating designs for the MMNL model, simulations are needed for evaluating the design over the complete parameter distributions. This is done by taking draws from the parameter distribution. The number of draws is determined by setting the *rdraws* (random draws) property. For taking 1,000 (pseudo) random draws, the following is added to the syntax:

```
;rdraws = random(1000)
```

Not only are pseudo random draws available, but also more intelligent sequences can be used, such as Halton sequences, Sobol sequences, or modified latin hybercube sampling (MLHS). These may be used by adding for example:

```
;rdraws = halton(50)
;rdraws = sobol(100)
;rdraws = mlhs(80)
```

Another approach is to use Gaussian quadrature. For this method, the number of abscissas per parameter is input. In the model described above, there are two random parameters. If one uses 5 abscissas per parameter, the total number of Gaussian draws will be 5×5 = 25 draws. In the syntax this would be:

```
;rdraws = gauss(5)
```

In case of few random parameters, Gaussian quadrature is extremely efficient. For higher numbers of random parameters, the number of Gaussian draws increases exponentially. For 5 random parameters with 5 abscissas each, this would yield 55 = 3,125 Gaussian draws. It is possible to set different numbers of abscissas per random parameter, for example:

```
;rdraws = gauss(3,3,2,4,5)
```

where the first two parameters have 3 abscissas, the third parameter 2 abscissas, etc. (in order of appearance). The total number of Gaussian draws would now be 3×3×2×4×5 = 360 draws.

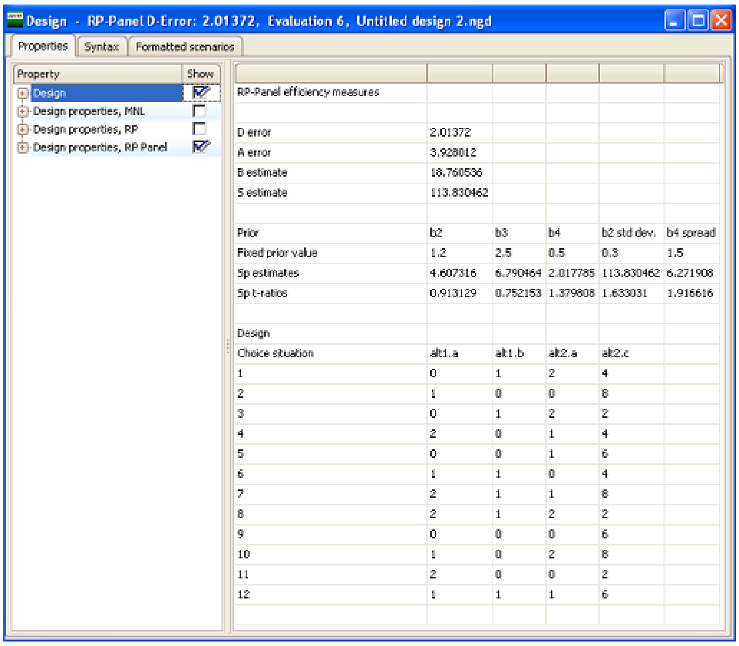
In the traditional (cross-sectional) MMNL model the observations from the choice situations are treated as independent. However, in SC experiments these choice observations are not independent as they are faced by the same respondent. Ngene has a unique feature in which this dependency can be taken into account by considering the panel MMNL model. Instead of model type ‘rp’ one can use ‘rppanel’ in the syntax. The evaluation of the design efficiency for the panel MMNL model is much more complex and time consuming than for the cross-sectional MMNL model as it requires sampling of respondents. The number of sampled (simulated) respondents is set by the *rep* property. The higher this value, the more accurate the computations, but the higher the computation time. The following two properties ensure that the panel MMNL model is used:

```
;eff = (rppanel,d)
;rep = 500
```

Thus the complete syntax for a MMNL design allowing for the pseudo panel nature of the SC design would look:

```
? Panel RP model
Design
;alts = alt1, alt2
;rows = 12
;eff = (rppanel,d)
;rep = 500
;rdraws = halton(250)
;model:
U(alt1) = b1[-0.2] + b2[n,1.2,0.3] * A[0,1,2] + b3[2.5]        * B[0,1]
    /
U(alt2) =              b2                * A          + b4[u,0.5,1.5] * C
[2,4,6,8]
$
```

The type of output generated for a MMNL design (cross-sectional and panel formulation) is identical to that for MNL designs discussed earlier. That is, Ngene will first display the overall efficiency measures of the design followed by S-error measures for each of the parameter estimates (including any standard deviation or spread parameters), after which the design itself is presented (see Figure 7.7 which shows a design generated using the above syntax).

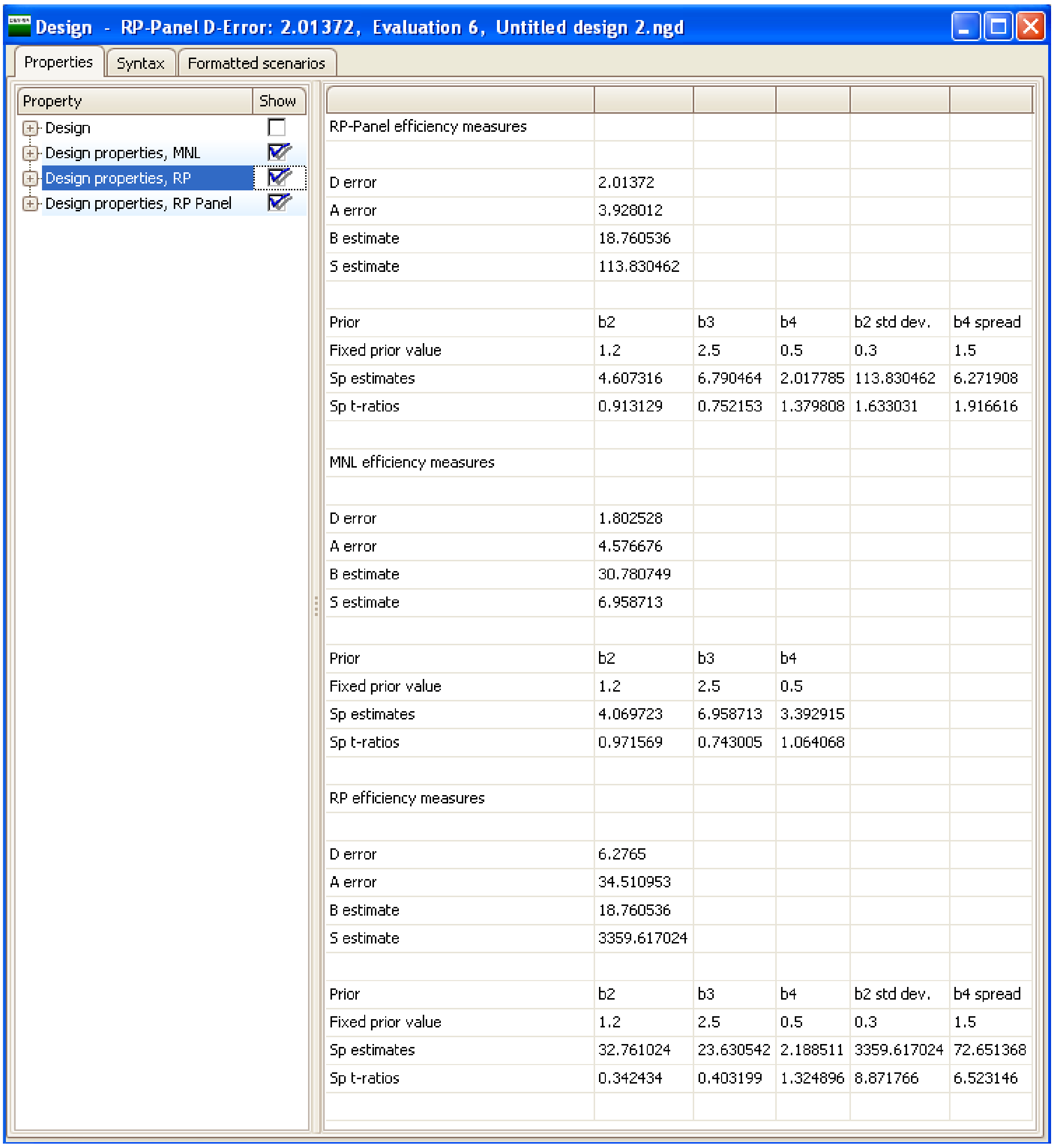


| | | | | | |
|---|---|---|---|---|---|
| RP-Panel efficiency measures | | | | | |
| | | | | | |
| D error | 2.01372 | | | | |
| A error | 3.928012 | | | | |
| B estimate | 18.760536 | | | | |
| S estimate | 113.830462 | | | | |
| | | | | | |
| Prior | b2 | b3 | b4 | b2 std dev. | b4 spread |
| Fixed prior value | 1.2 | 2.5 | 0.5 | 0.3 | 1.5 |
| Sp estimates | 4.607316 | 6.790464 | 2.017785 | 113.830462 | 6.271908 |
| Sp t-ratios | 0.913129 | 0.752153 | 1.379808 | 1.633031 | 1.916616 |
| | | | | | |
| Design | | | | | |
| Choice situation | alt1.a | alt1.b | alt2.a | alt2.c | |
| 1 | 0 | 1 | 2 | 4 | |
| 2 | 1 | 0 | 0 | 8 | |
| 3 | 0 | 1 | 2 | 2 | |
| 4 | 2 | 0 | 1 | 4 | |
| 5 | 0 | 0 | 1 | 6 | |
| 6 | 1 | 1 | 0 | 4 | |
| 7 | 2 | 1 | 1 | 8 | |
| 8 | 2 | 1 | 2 | 2 | |
| 9 | 0 | 0 | 0 | 6 | |
| 10 | 1 | 0 | 2 | 8 | |
| 11 | 2 | 0 | 0 | 2 | |
| 12 | 1 | 1 | 1 | 6 | |

**Figure 7.7: MMNL design output**

Unlike the output for MNL designs however, when a MMNL design is requested, Ngene will automatically generate additional outputs for each of the normally reported efficiency measures assuming other model types. This is shown in Figure 7.8 where Ngene reports the efficiency outputs for the same design given in Figure 7.7, assuming MNL, MMNL cross sectional (RP) and MMNL panel (RP Panel) model formulations. In generating the MNL model outputs, the means of any random parameter distributions are assumed as the prior parameter inputs. Note that whilst these values are both calculated and reported for the different model types, only the efficiency measure for the model type requested in the *eff* property is used in the design optimization routine.

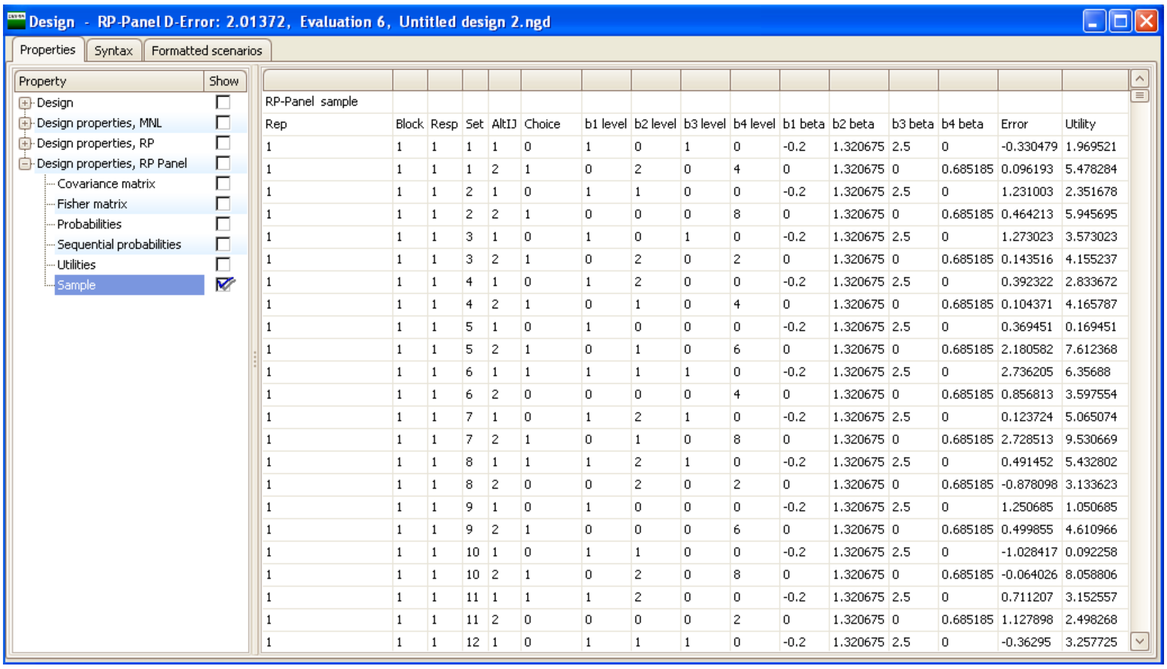
Finally, as briefly mentioned earlier, calculation of the Fisher Information matrix for the panel version of designs generated for a MMNL model requires the generation of a sample of respondents. This greatly increases the time required to construct such a design compared to designs generated assuming other model types. Ngene allows the user to observe the sample generated for these calculations (but only for panel MMNL designs) by clicking on the sample tick box within the RP panel tree structure. An example sample is shown in Figure 7.8. The simulated sample of respondents is set up in such a way that each row of data represents an alternative, with multiple rows representing a choice task in the design. The parameter estimates in the sample for random parameters are drawn from the prior parameter estimates provided by the user in generating the design using Halton sequences. By using Halton sequences, the parameter estimates for each simulated respondent is kept constant over multiple design generation iterations. Similarly, the EV1 error term is drawn using Halton sequences. The choice variable is constructed based on the alternative that is observed to have the highest utility within each choice task.


Design - RP-Panel D-Error: 2.01372, Evaluation 6, Untitled design 2.ngd
Properties
Syntax
Formatted scenarios

| Property | Show |
|---|---|
| Design | □ |
| Design properties, MNL | ☑ |
| Design properties, RP | ☑ |
| Design properties, RP Panel | ☑ |

| | | | | | |
|---|---|---|---|---|---|
| RP-Panel efficiency measures | | | | | |
| | | | | | |
| D error | 2.01372 | | | | |
| A error | 3.928012 | | | | |
| B estimate | 18.760536 | | | | |
| S estimate | 113.830462 | | | | |
| | | | | | |
| Prior | b2 | b3 | b4 | b2 std dev. | b4 spread |
| Fixed prior value | 1.2 | 2.5 | 0.5 | 0.3 | 1.5 |
| Sp estimates | 4.607316 | 6.790464 | 2.017785 | 113.830462 | 6.271908 |
| Sp t-ratios | 0.913129 | 0.752153 | 1.379808 | 1.633031 | 1.916616 |
| | | | | | |
| MNL efficiency measures | | | | | |
| | | | | | |
| D error | 1.802528 | | | | |
| A error | 4.576676 | | | | |
| B estimate | 30.780749 | | | | |
| S estimate | 6.958713 | | | | |
| | | | | | |
| Prior | b2 | b3 | b4 | | |
| Fixed prior value | 1.2 | 2.5 | 0.5 | | |
| Sp estimates | 4.069723 | 6.958713 | 3.392915 | | |
| Sp t-ratios | 0.971569 | 0.743005 | 1.064068 | | |
| | | | | | |
| RP efficiency measures | | | | | |
| | | | | | |
| D error | 6.2765 | | | | |
| A error | 34.510953 | | | | |
| B estimate | 18.760536 | | | | |
| S estimate | 3359.617024 | | | | |
| | | | | | |
| Prior | b2 | b3 | b4 | b2 std dev. | b4 spread |
| Fixed prior value | 1.2 | 2.5 | 0.5 | 0.3 | 1.5 |
| Sp estimates | 32.761024 | 23.630542 | 2.188511 | 3359.617024 | 72.651368 |
| Sp t-ratios | 0.342434 | 0.403199 | 1.324896 | 8.871766 | 6.523146 |

**Figure 7.8: MMNL design output by model type**


Design - RP-Panel D-Error: 2.01372, Evaluation 6, Untitled design 2.ngd
Properties
Syntax
Formatted scenarios

| Property | Show |
|---|---|
| Design | ☐ |
| Design properties, MNL | ☐ |
| Design properties, RP | ☐ |
| Design properties, RP Panel | ☐ |
| Covariance matrix | ☐ |
| Fisher matrix | ☐ |
| Probabilities | ☐ |
| Sequential probabilities | ☐ |
| Utilities | ☐ |
| Sample | ☑ |

RP-Panel sample

| Rep | Block | Resp | Set | AltIJ | Choice | b1 level | b2 level | b3 level | b4 level | b1 beta | b2 beta | b3 beta | b4 beta | Error | Utility |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | -0.2 | 1.320675 | 2.5 | 0 | -0.330479 | 1.969521 |
| 1 | 1 | 1 | 1 | 2 | 1 | 0 | 2 | 0 | 4 | 0 | 1.320675 | 0 | 0.685185 | 0.096193 | 5.478284 |
| 1 | 1 | 1 | 2 | 1 | 0 | 1 | 1 | 0 | 0 | -0.2 | 1.320675 | 2.5 | 0 | 1.231003 | 2.351678 |
| 1 | 1 | 1 | 2 | 2 | 1 | 0 | 0 | 0 | 8 | 0 | 1.320675 | 0 | 0.685185 | 0.464213 | 5.945695 |
| 1 | 1 | 1 | 3 | 1 | 0 | 1 | 0 | 1 | 0 | -0.2 | 1.320675 | 2.5 | 0 | 1.273023 | 3.573023 |
| 1 | 1 | 1 | 3 | 2 | 1 | 0 | 2 | 0 | 2 | 0 | 1.320675 | 0 | 0.685185 | 0.143516 | 4.155237 |
| 1 | 1 | 1 | 4 | 1 | 0 | 1 | 2 | 0 | 0 | -0.2 | 1.320675 | 2.5 | 0 | 0.392322 | 2.833672 |
| 1 | 1 | 1 | 4 | 2 | 1 | 0 | 1 | 0 | 4 | 0 | 1.320675 | 0 | 0.685185 | 0.104371 | 4.165787 |
| 1 | 1 | 1 | 5 | 1 | 0 | 1 | 0 | 0 | 0 | -0.2 | 1.320675 | 2.5 | 0 | 0.369451 | 0.169451 |
| 1 | 1 | 1 | 5 | 2 | 1 | 0 | 1 | 0 | 6 | 0 | 1.320675 | 0 | 0.685185 | 2.180582 | 7.612368 |
| 1 | 1 | 1 | 6 | 1 | 1 | 1 | 1 | 1 | 0 | -0.2 | 1.320675 | 2.5 | 0 | 2.736205 | 6.35688 |
| 1 | 1 | 1 | 6 | 2 | 0 | 0 | 0 | 0 | 4 | 0 | 1.320675 | 0 | 0.685185 | 0.856813 | 3.597554 |
| 1 | 1 | 1 | 7 | 1 | 0 | 1 | 2 | 1 | 0 | -0.2 | 1.320675 | 2.5 | 0 | 0.123724 | 5.065074 |
| 1 | 1 | 1 | 7 | 2 | 1 | 0 | 1 | 0 | 8 | 0 | 1.320675 | 0 | 0.685185 | 2.728513 | 9.530669 |
| 1 | 1 | 1 | 8 | 1 | 1 | 1 | 2 | 1 | 0 | -0.2 | 1.320675 | 2.5 | 0 | 0.491452 | 5.432802 |
| 1 | 1 | 1 | 8 | 2 | 0 | 0 | 2 | 0 | 2 | 0 | 1.320675 | 0 | 0.685185 | -0.878098 | 3.133623 |
| 1 | 1 | 1 | 9 | 1 | 0 | 1 | 0 | 0 | 0 | -0.2 | 1.320675 | 2.5 | 0 | 1.250685 | 1.050685 |
| 1 | 1 | 1 | 9 | 2 | 1 | 0 | 0 | 0 | 6 | 0 | 1.320675 | 0 | 0.685185 | 0.499855 | 4.610966 |
| 1 | 1 | 1 | 10 | 1 | 0 | 1 | 1 | 0 | 0 | -0.2 | 1.320675 | 2.5 | 0 | -1.028417 | 0.092258 |
| 1 | 1 | 1 | 10 | 2 | 1 | 0 | 2 | 0 | 8 | 0 | 1.320675 | 0 | 0.685185 | -0.064026 | 8.058806 |
| 1 | 1 | 1 | 11 | 1 | 1 | 1 | 2 | 0 | 0 | -0.2 | 1.320675 | 2.5 | 0 | 0.711207 | 3.152557 |
| 1 | 1 | 1 | 11 | 2 | 0 | 0 | 0 | 0 | 2 | 0 | 1.320675 | 0 | 0.685185 | 1.127898 | 2.498268 |
| 1 | 1 | 1 | 12 | 1 | 0 | 1 | 1 | 1 | 0 | -0.2 | 1.320675 | 2.5 | 0 | -0.36295 | 3.257725 |

**Figure 7.9: Panel MMNL simulated sample**

## 7.2.4 Designs for estimating error components models

In the error components (EC) model, the user can specifically add error terms (which are normally distributed with mean zero and a given standard deviation) into the utility functions in the *model* property. Prior values for the standard deviation need to be provided. The *eff* property will need to reflect the fact that an error component model is being used, setting the model type (the first argument of the *eff* property) to ‘ec’. In the utility functions, an error component will be recognized by putting ‘ec’ as a first argument in the square brackets following the parameter name, e.g. ‘s1 [ec,0.2]’ indicates that parameter ‘s1’ represents a normal distributed error component with mean zero and standard deviation 0.2. Multiple error components can be used, for example:

```
;eff = (ec,d)
;model:
U(alt1) = b1[-0.2] + b2[1.2] * A[0,1,2]   + s1[ec,0.2]      /
U(alt2) = b3[0.9]  + b4[0.8] * B[2,3,4,5] + s1 + s2[ec,0.5] /
U(alt3) =            b5[1.5] * C[1,2,3]        + s2
```

As in the MMNL model, draws need to be taken from the random error components, such that the *rdraws* property needs to be set in a similar fashion.

Similar to designs for the MMNL model, it is also possible to generate a panel version of error components type designs. This also requires the generation of a sample of simulated respondents which is handled in the exact same manner as with MMNL designs, as shown in the following syntax.

```
;eff = (ecpanel,d)
;rep = 500
```

An example set of syntax for an EC design allowing for the pseudo panel nature of the SC design might therefore look as follows.

```
Design
;alts = alt1, alt2, alt3
;rows = 12
;eff = (ecpanel,d)
;rep = 500
;rdraws = halton(250)
;model:
U(alt1) = b1[-0.2] + b2[1.2] * A[0,1,2]   + s1[ec,0.2]      /
U(alt2) = b3[0.9]  + b4[0.8] * B[2,3,4,5] + s1 + s2[ec,0.5] /
U(alt3) =            b5[1.5] * C[1,2,3]        + s2
$
```

The output for EC designs is exactly the same as that reported for random parameter logits models. As with MMNL designs, Ngene will report the efficiency measures for the MNL, EC cross sectional, and EC panel models whenever an EC cross sectional or panel design is requested. Also, as with panel MMNL designs, panel EC designs require the simulation of a sample of respondents. The simulated sample may be viewed in a manner similar to that when dealing with panel MMNL designs (see Section 7.2.3).

### 7.2.5 Designs for estimating combined random parameters and error components

In some cases, the analyst may wish to generate a design that contains both random parameters and error components. Setting the model type to 'rpec' lets Ngene know that both are used in the utility functions. In the following example, 'b2' and 'b4' are random parameters, while 's1' and 's2' are error components:

```
Design
;eff = (rpec,d)
;model:
U(alt1) = b1[-0.2] + b2[n,1.2,0.3]  * A[0,1,2]    + s1[ec,0.2]
    /
U(alt2)  = b3[0.9]   + b4[0.8]          * B[2,3,4,5] + s1           + s2
[ec,0.5]   /
U(alt3) =            b5[u, 0.2,1.5] * C[1,2,3]                + s2
```

Similar to 'rp', 'ec' and 'rpec' can be used in a panel approach. Setting the model type to 'ecpanel' or 'rpecpanel' and setting the *rep* property will tell Ngene to create a sample of respondents for doing the computations for the panel approach. A full example of syntax is provided below.

```
Design
;alts = alt1, alt2, alt3
;rows = 12
;eff = (rpecpanel,d)
;rep = 500
;rdraws = halton(250)
;model:
U(alt1) = b1[-0.2] + b2[n,1.2,0.3] * A[0,1,2]    + s1[ec,0.2]
   /
U(alt2)  = b3[0.9]   + b4[0.8]         * B[2,3,4,5] + s1           + s2
[ec,0.5]   /
U(alt3) =             b5[u,0.2,1.5] * C[1,2,3]                + s2
```

```
$
```

In case the model specification contains error components, but the *eff* property indicates an ‘rp’ model, then the error components will be ignored in the efficiency calculations. Similarly, if the model specification contains random parameters, but the model type is given as ‘ec’ in the *eff* property, the random parameters will be assumed fixed (i.e., set to the mean value of the distribution) when optimizing the design.

Once more, the output for this type of model is similar to that described earlier for the MMNL and EC models, the main difference being that efficiency measures may now be obtained for all model types, not just for MNL or MMNL or EC models. Also, as previously described, simulated samples may be generated for the MMNL and error component panel models.

### 7.2.6 Reporting efficiency measures for different model types

In the previous sections, the example syntax assumed that the utility specifications matched perfectly the model type described in the *eff* property. For example, in using

```
b2[n,1.2,0.3]
```

to specify a random parameter, we used either

```
;eff = (rppanel,d)
```

or

```
;eff = (rp,d)
```

Similarly, when an error component was included in the utility function, the *eff* property referred to either a cross sectional or panel EC model form. In Ngene, it is possible however to specify one type of model form in the set of utility functions but optimize on another type of model in the *eff* property. The syntax below demonstrates one such case where the utility specification assumes random parameters for b2 and b4, but the design is optimized assuming an MNL model (with fixed parameters).

```
Design
;alts = alt1, alt2
;rows = 12
;eff = (mnl,d)
;rep = 500
;rdraws = halton(250)
;model:
U(alt1) = b1[-0.2] + b2[n,1.2,0.3] * A[0,1,2] + b3[2.5]        * B[0,1]
    /
U(alt2) =            b2              * A        + b4[u,0.5,1.5] * C
[2,4,6,8]
$
```

In cases such as this, Ngene will optimize on the model type requested in the *eff* property but will also report the efficiency measure outcomes for model forms outlined in the utility specifications. When optimized assuming an MNL model, Ngene will assume the average value of any random parameter distribution as being the true prior value and ignore any error components. For designs generated assuming a random parameter type model, any error component will be ignored in the optimization routine. Similarly, for EC designs, any random parameter estimate located in the utility functions will also be ignored in optimization of the design. The benefit of allowing different forms of utility specifications from the efficiency measure being optimized is that the user can easily examine how the design is likely to perform assuming a different model type than that for which it was optimized for. We discuss a similar concept in Section 7.4 when we discuss the use of model averaging in the design generation process.

### 7.2.7 Designs with no choice alternatives

The question as to whether or not to include a 'status quo' alternative (sometimes referred to as a 'no choice' or 'opt out' alternative in various literatures) in SC studies has been widely debated in many discipline areas. Within the literature, significant differences in results of SC experiments with and without the presence of status quo alternatives have been found (see e.g., Dhar and Simonson 2003), and in general, the recommendation has been that status quo alternatives should be used in such experiments (e.g., Louviere *et al*. 2000; Adamowicz and Boxall 2001; Bennett and Blamey 2001; Bateman *et al*. 2003). These recommendations have grown from a number of arguments that have been put forward for the use of status quo alternatives. These arguments include that the inclusion of a status quo alternative leads to an increase in the realism of SC tasks (see e.g., Louviere and Woodworth 1983; Batsell and Louviere 1991; Carson *et al*. 1994), an increase in the external validity of welfare estimates derived from SC experiments (see e.g., Adamozicz and Boxall 2001) and an improvement in the statistical efficiency of parameters estimated from discrete choice models (see e.g., Louviere et al. 2000; Anderson and Wiley 1992). For a further overview of these arguments, see e.g., Kontoleon and Yabe (2003) or Dhar and Simpson (2003).

Traditionally, where used, the no choice or status quo alternative has been represented in SC data as either being an alternative labelled as *'none'* and devoid of any attribute levels or alternatively as an option labelled as *'your current alternative'* with attribute levels given simply as *'at the current level'* (see e.g., Tversky and Shafir 1992; Dhar 1997; Kontoleon and Yabe 2003). Whilst both versions of the status quo alternative have different implications given different interpretational meanings (i.e., the *'none'* option represents a complete opt-out of all non-status quo alternatives by the respondent whereas the *'your current alternative'* option represents the choice of an already experienced or known alternative and hence is not strictly a no choice alternative), it is the impact upon respondents of including such alternatives in SC experiments that requires careful consideration. Where a *'none'* option is used, there exists little possibility of interpretation differences in terms of what the alternative means to respondents as the choice of selecting none of the other alternatives presented within a choice task should have the same meaning for the entire sample. Where the status quo alternative is described simply as *'your current alternative'* however, interpretation differences may arise as different respondents may have different current alternatives, or in the case where all respondents face the same status quo alternative, may possess different perceptions as to the current attribute levels that that alternative possesses.

Independent of the form of the no choice alternative, one or more no choice alternatives can be accounted for in generating a design by naming an alternative in the *alts* property but not specifying a utility function after the model property. Note that this can be done for any model type, and can also be used when generating orthogonal designs. An example of syntax for an MNL design allowing for a no choice alternative is given below.

```
Design
;alts = alt1, alt2, alt3
;rows = 12
;eff = (mnl,d)
;model:
U(alt1) = b11[-0.2] + b2[1.2] * A[0,1,2] + b3[2.5] * B[0,1]     /
U(alt2) = b12[0.3]  + b2      * A        + b4[1.1] * C[2,4,6,8]
$
```

In generating such designs, Ngene will assume that the utility for a no choice alternative is zero and that no attribute levels are attached to the alternative. We therefore make a distinction between this form of design and one where the no choice alternative does indeed have attribute levels. For example, many designs employ a form of status quo option, which we term *reference alternative*, which is similar to the *'at the current level'* status quo alternative format but which involves the capturing and often relation back to respondents as part of SC choice tasks of the (perceived) attribute levels of respondent specific currently (or recently) experienced real life alternatives. That is, respondents are asked what their perceptions are of the attribute levels for a current (usually chosen) real world alternative, and these are used as an alternative in the choice tasks that they view. We discuss this specific form of design in Section 8.3.

### 7.2.8 Designs with dummy and effects coded attributes

Rather than estimate a *single param*eter for each attribute (assuming a linear relationship between changes in the attribute and utility), one can also estimate *multiple unique parameters* associated with *l*-1 of an attribute's levels (suggesting that different levels have a different impact upon utility, and hence assuming a nonlinear relationship). Typically, such nonlinear relationships are represented using one of two data coding structures, these being dummy coding and effects coding.

To demonstrate, consider an example where an attribute representing color can take on three levels; blue, red and yellow. Within the utility specification, this attribute might be represented as follows.

```
U(alt1) = color[0.17] * color[0,1,2]
```

Graphically, the marginal utility for this color attribute may be represented as per Figure 7.10. In this case, the marginal utility difference between ‘blue’ and 'red', is the same as the difference between the marginal utilities for ‘red’ and ‘yellow’. This is because a single parameter has been assigned to the attribute and thus has the same impact upon utility as one moves from any one level to the next adjacent level.

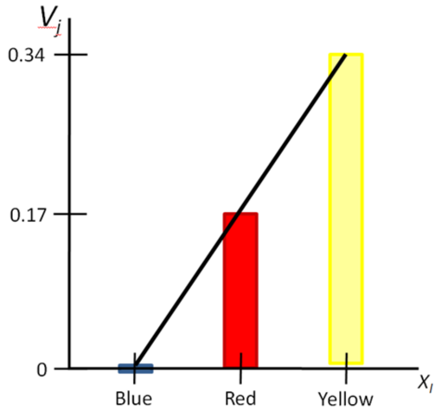
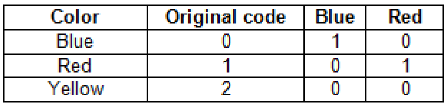


**Figure 7.10: Linear relationship between color and utility**

Rather than assume a linear relationship between the attribute levels of an attribute and utility, dummy and effects coding requires that the analyst create unique variables for up to *l*-1 levels of an attribute, each of which may then be associated with distinct parameter estimates. As such, rather than having a single parameter estimate, the analyst now has *l*-1 parameter estimates, each of which represents the marginal utility associated with their corresponding attribute levels, with the L$^{th}$ level having a marginal utility set to zero.

Dummy coding utilizes a series of 0s and 1s to relate each attribute level of the original variable to the newly created columns. Table 7.2 demonstrates the dummy coding concept for the color example given above. First, the analyst creates *l*-1 columns corresponding in this case to the creation of 2 additional columns (3 levels – 1 = 2). In this example, we relate the two new columns to the colors ‘blue’ and ‘red’. Note that it does not matter which attribute levels one creates the new columns for, as discrete choice models produce estimates of relative utility, and hence, any order will produce the same result. Next, every time an attribute level appears in the design (data), the column corresponding to that level receives a value of 1, otherwise it receives a value of 0. For the attribute level with no corresponding column (in Table 7.2 this is represented by the color ‘yellow’), for all constructed columns it will take the value of 0 (i.e., ‘blue’ and ‘red’). In the design generation (or estimation process), the analyst now estimates parameters for the newly created *l*-1 dummy variables.

**Table 7.2: Example dummy code**



| Color | Original code | Blue | Red |
|---|---|---|---|
| Blue | 0 | 1 | 0 |
| Red | 1 | 0 | 1 |
| Yellow | 2 | 0 | 0 |

Figure 7.11 demonstrates the marginal utilities that could arise from the dummy coding exercise presented in Table 7.2. Using only the newly created ‘blue’ and ‘red’ dummy variables, two unique parameter estimates will be obtained, one for each. The ‘yellow’ level, not having a dummy variable column will automatically have a marginal utility of zero (hence the ‘blue’ and ‘red’ dummy variable parameters will be relative to this ‘base’ level).

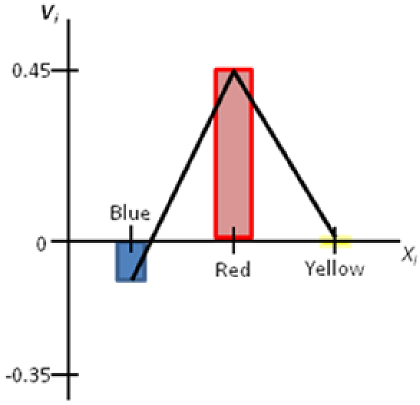


**Figure 7.11: Nonlinear relationship between color and utility represented using dummy codes**

Dummy coding in Ngene is performed via minor modifications to how a parameter is handled in the model utility functions. Firstly, in specifying the parameter value, the analyst will need to add the syntax .dummy after naming the parameter, such as

```
<parameter name>.dummy
```

Next, the analyst needs to provide *l*-1 unique parameter priors associated with the *l*-1 newly created dummy variables. This is done by separating *l*-1 parameter priors using a | symbol. Note that if the analyst does not specify the attribute levels for a dummy coded variable, Ngene will use the levels 0,1,…, *L* when presenting the design. Where attribute levels have been specified, Ngene will report these values when presenting the final design despite using the dummy coded variables in the design generation process. Example syntax is presented below for the color example, where the first color level has been assigned a prior parameter value of -0.15, the second 0.45 and the final omitted level, a value of zero.

```
U(alt1) = color.dummy[-0.15|0.45] * color
```

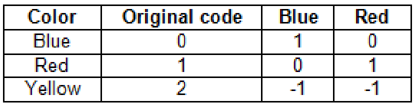
or

```
U(alt1) = color.dummy[-0.15|0.45] * color[0,1,2]
```

Effects coding is similar to dummy coding in that it allows the analyst to detect nonlinearities in the marginal utilities for levels of attributes rather than assuming a linear relationship between an attributes levels and overall utility. However, effects coding offers a number of theoretical advantages over dummy coding. In particular, if two or more attributes are dummy coded, then each will have its own ‘base’ level where all dummy coded columns are set at zero. For example, if both color and gender are dummy coded, then the marginal utility for ‘yellow’ will have the same marginal utility as say ‘male’ (assuming male = 0). In this way, the ‘base’ levels of several dummy coded variables will be perfectly confounded with each other, or a model constant if one is present.

Effects coding overcomes this by changing the base level in the coding structure in such a way as to allow for a unique estimate for that level. This is done by changing 0 to -1 in each column for the base attribute level, as shown in Table 7.3. In this way, the base level will not be equal to zero, but rather will be equal to minus the sum of the remaining parameter estimates. This is shown both in Equation (7.19) and Figure 7.12.

**Table 7.3: Example dummy code**



| Color | Original code | Blue | Red |
|---|---|---|---|
| Blue | 0 | 1 | 0 |
| Red | 1 | 0 | 1 |
| Yellow | 2 | -1 | -1 |

$$\beta_{yellow} = -\beta_{red} - \beta_{blue} \quad (7.19)$$

Effects coding and dummy coding should provide the same results in terms of the estimated utilities for each alternative as well as producing the same choice probabilities. Differences however will exist in the parameter priors (estimates) for the model constants as well as between the dummy or effects coded variables. Indeed, the effects coded priors (estimates) should be similar to the dummy coded priors (estimates) up to some scale.

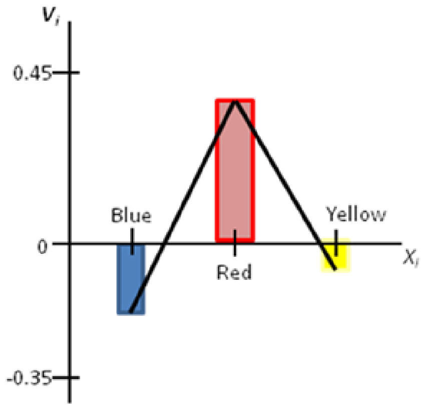


**Figure 7.12: Nonlinear relationship between color and utility represented using effects codes**

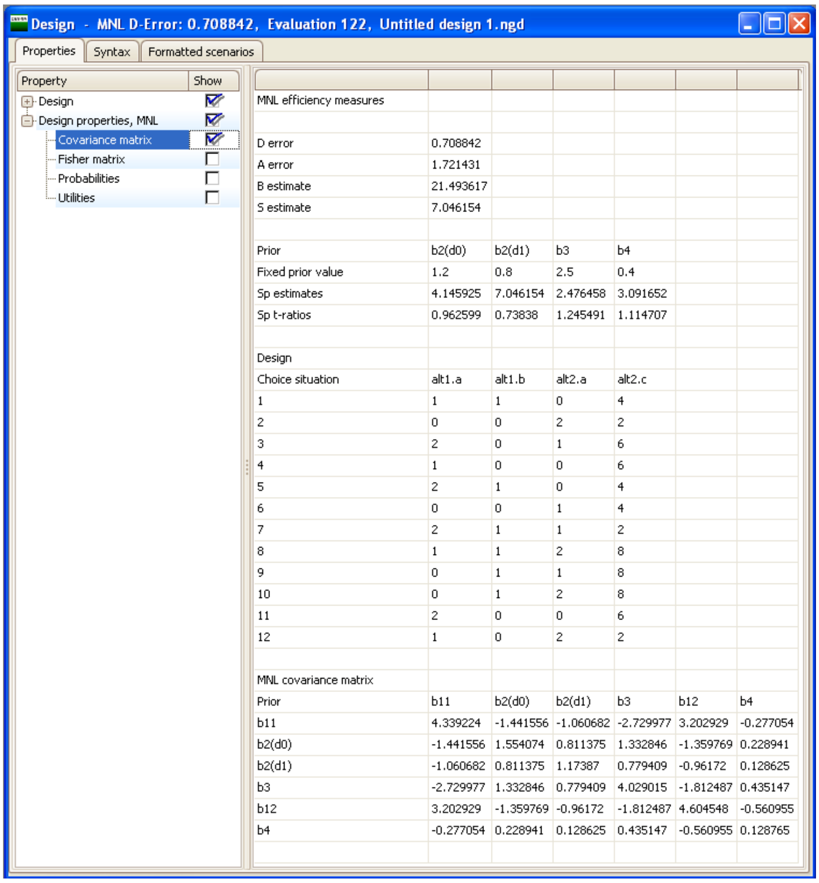
In Ngene, the process to specify effects codes is similar to that for specifying dummy codes. The analyst must still specify *l*-1 parameter priors, however rather than use the syntax .dummy, .effects is used instead. This is shown in the following example. As with dummy codes, the analyst need not specify the levels of the attribute (in which case Ngene will report the levels as 0,1,…*L* in the design output), however if levels are provided by the analyst, these levels will be used in any output provided. Also, just like the dummy coding, the Ngene automatically codes the last attribute level as the base.

```
U(alt1) = color.effects[-0.36|0.4] * color[0,1,2]
```

An example syntax shown dummy codes is given below.

```
Design
;alts = alt1, alt2, alt3
;rows = 12
;eff = (mnl,d)
;model:
U(alt1) = b11[-0.2] + b2.dummy[1.2|0.8] * A[0,1,2] + b3[2.5] * B[0,1]
  /
U(alt2) = b12[0.3]  + b2                    * A          + b4[0.4] * C
[2,4,6,8]
$
```

Output showing a design generated using the above syntax is given in Figure 7.13. Note that in presenting the design, Ngene does not present the dummy or effects coded columns but rather the levels of the design as if they were not dummy or effects coded. This is because, even though the optimization routine treated these variables as dummy or effects coded, conversion of the design into a choice survey is best done if the attributes are viewed as per those given in Figure 7.13. Note once more that even though the design is represented as if it were treated as linear in the attributes, the optimization routine does indeed treat the design as if it were dummy or effects coded. This can be confirmed by examining either the Fisher Information matrix or AVC matrix, where additional rows and columns will be present for each dummy or effects coded prior parameter. This can be clearly seen in Figure 7.13 where the AVC matrix now has additional rows and columns for the two dummy priors assumed in the syntax.


Design - MNL D-Error: 0.708842, Evaluation 122, Untitled design 1.ngd
Properties
Syntax
Formatted scenarios

| Property | Show |
|---|---|
| Design | ☑ |
| Design properties, MNL | ☑ |
| Covariance matrix | ☑ |
| Fisher matrix | □ |
| Probabilities | □ |
| Utilities | □ |

| | | | | | | |
|---|---|---|---|---|---|---|
| MNL efficiency measures | | | | | | |
| | | | | | | |
| D error | 0.708842 | | | | | |
| A error | 1.721431 | | | | | |
| B estimate | 21.493617 | | | | | |
| S estimate | 7.046154 | | | | | |
| | | | | | | |
| Prior | b2(d0) | b2(d1) | b3 | b4 | | |
| Fixed prior value | 1.2 | 0.8 | 2.5 | 0.4 | | |
| Sp estimates | 4.145925 | 7.046154 | 2.476458 | 3.091652 | | |
| Sp t-ratios | 0.962599 | 0.73838 | 1.245491 | 1.114707 | | |
| | | | | | | |
| Design | | | | | | |
| Choice situation | alt1.a | alt1.b | alt2.a | alt2.c | | |
| 1 | 1 | 1 | 0 | 4 | | |
| 2 | 0 | 0 | 2 | 2 | | |
| 3 | 2 | 0 | 1 | 6 | | |
| 4 | 1 | 0 | 0 | 6 | | |
| 5 | 2 | 1 | 0 | 4 | | |
| 6 | 0 | 0 | 1 | 4 | | |
| 7 | 2 | 1 | 1 | 2 | | |
| 8 | 1 | 1 | 2 | 8 | | |
| 9 | 0 | 1 | 1 | 8 | | |
| 10 | 0 | 1 | 2 | 8 | | |
| 11 | 2 | 0 | 0 | 6 | | |
| 12 | 1 | 0 | 2 | 2 | | |
| | | | | | | |
| MNL covariance matrix | | | | | | |
| Prior | b11 | b2(d0) | b2(d1) | b3 | b12 | b4 |
| b11 | 4.339224 | -1.441556 | -1.060682 | -2.729977 | 3.202929 | -0.277054 |
| b2(d0) | -1.441556 | 1.554074 | 0.811375 | 1.332846 | -1.359769 | 0.228941 |
| b2(d1) | -1.060682 | 0.811375 | 1.17387 | 0.779409 | -0.96172 | 0.128625 |
| b3 | -2.729977 | 1.332846 | 0.779409 | 4.029015 | -1.812487 | 0.435147 |
| b12 | 3.202929 | -1.359769 | -0.96172 | -1.812487 | 4.604548 | -0.560955 |
| b4 | -0.277054 | 0.228941 | 0.128625 | 0.435147 | -0.560955 | 0.128765 |

**Figure 7.13: Example Ngene design output with a dummy coded variable**

Care should be taken when using dummy or effects codes in generating designs however. A commonly observed problem occurs when a dummy or effects coded variable takes the value 1 over the entire design only a few times. For example, if a variable is coded 0,1,2,4 and the design is generated with 16 choice situations, then each attribute level will appear four times over the design. If the variable is now dummy coded however, then the value 1 will appear only four times for each dummy coded variable with the remainder of the variable taking the value of 0 (i.e., the variable will have four 1s and 12 0s over the 16 choice situations). In this way, the design can become quite sparse in terms of non-zero values, the result of which will either be an inefficient design, or a design with a near singular Fisher Information matrix meaning that it cannot be inverted to obtain the design's AVC matrix.

### 7.2.9 Efficient designs with interactions

Previously, in Section 6.2.4, interactions were introduced in the context of orthogonal designs. Interactions can also be specified for efficient designs. In fact, Ngene is not limited to two-way interactions. It can handle interactions of any order.

The two-way interaction is the most basic form of interaction in Ngene. As with two-way interactions for orthogonal designs, the syntax is specified by introducing a parameter, and multiplying that parameter by two attributes that have already been specified. The key difference is that for efficient designs, a parameter prior would typically be specified for the interaction parameter. An example is provided below, where an interaction parameter i1, with a prior value of 0.1, is introduced in the first alternative, for the interaction of attributes A and B.

```
Design
;alts = alt1, alt2
;rows = 12
;eff = (mnl,d)
;model:
U(alt1) = b1[-0.2] + b2[0.7] * A[0,1,2] + b3[0.8] * B[0,1]     + i1[0.1]
* A * B /
U(alt2) =            b2      * A        + b4[0.2] * C[2,4,6,8]

$
```

The interaction parameter will be reported in the Fisher and AVC matrices, and be included in the calculation of the efficiency measures. The design reported will include two columns for the attributes, as well as an additional column for the interaction (even though this typically would not be shown in a survey).

To specify interactions of a higher order, simply multiply the parameter by more attributes. In the example below, a new attribute, D, has been introduced, and included in the interaction, resulting in a three-way interaction.

```
Design
;alts = alt1, alt2
;rows = 12
;eff = (mnl,d)
;model:
U(alt1) = b1[-0.2] + b2[0.7] * A[0,1,2] + b3[0.8] * B[0,1]     + b5
[0.05] * D[1,2] + i1[0.1] * A * B * D /
U(alt2) =            b2      * A        + b4[0.2] * C[2,4,6,8]

$
```

In some situations, an attribute is only important in an interaction term, not as a main effect. As of version 1.1, Ngene allows an attribute to be introduced in the interaction, without first being specified with a parameter for a main effect. The attribute will still be reported in its own column when the design levels are reported, however the level will only be used in the interaction, and so no main effect parameter will be included in the Fisher and AVC matrices. The example below is a modification of the previous example, where the attribute D has been removed as a main effect, and only included in the three-way interaction.

```
Design
;alts = alt1, alt2
;rows = 12
;eff = (mnl,d)
```

```
;model:
U(alt1) = b1[-0.2] + b2[0.7] * A[0,1,2] + b3[0.8] * B[0,1]     + i1[0.1]
* A * B * D[1,2]  /
U(alt2) =           b2      * A        + b4[0.2] * C[2,4,6,8]

$
```

Finally, you may wish to specify an interaction with a dummy or effects coded attribute. In this context, it probably does not make sense to treat the dummy or effects coded attribute as continuous in the interaction. As of version 1.1, Ngene allows an interaction to be specified with a specific attribute level, rather than all possible levels of an attribute. When referencing an existing dummy or effects coded attribute in the interaction, use the syntax `<attribute>.dummy [<exact level of attribute>]`. This will evaluate to 1 if the attribute takes on the attribute level specified, or 0 otherwise. This is best demonstrated using the example below.

```
Design
;alts = alt1, alt2, alt3
;rows = 12
;eff = (mnl,d)
;model:
U(alt1) = b11[-0.2] + b2.dummy[-0.6|-0.35] * A[1,2,3] + b3[0.8] * B[1,2]
    + i1[0.1] * A.dummy[2] * B /
U(alt2) = b12[0.3]  + b2                   * A        + b4[0.2] * C
[2,4,6,8]
$
```

Here, attribute A is dummy coded, with a prior of -0.6 for level 1 and -0.35 for level 2, with level 3 forming the base. In the interaction, `A.dummy[2]` will evaluate to 1 if attribute A is 2. Note that in this example, levels were specified for attribute A (`A[1,2,3]`), even though each level will be coded as 0 or 1 internally when evaluating the dummy coded main effect. If the levels were not explicitly specified, they would default to `[0,1,2]`, and these levels would need to be referenced in the interaction term. If each level of a dummy or effects coded attribute needs to be interacted with another attribute, then one interaction needs to be added for each level.

Note that dummy coding of an attribute level in an interaction does not require that that attribute's main effect be dummy or effects coded. The example below is the same as above, except that in the interaction, level 2 of attribute B is dummy coded, even though it was not for the main effect.

```
Design
;alts = alt1, alt2, alt3
;rows = 12
;eff = (mnl,d)
;model:
U(alt1) = b11[-0.2] + b2.dummy[-0.6|-0.35] * A[1,2,3] + b3[0.8] * B[1,2]
    + i1[0.1] * A.dummy[2] * B.dummy[1] /
U(alt2) = b12[0.3]  + b2                   * A        + b4[0.2] * C
[2,4,6,8]
$
```

## 7.3 Bayesian efficient designs

It is strongly advised to first generate a non-Bayesian design with the MNL model. This allows problems to be much more quickly identified. For example, the priors might lead to extreme choice probabilities of zero and one, and may need to be adjusted. An MNL design should always be generated quickly, so if it is not, then you know there is a problem with your design specification. Random parameter and Bayesian models are much slower to generate, and it may not be clear for some time that there is a problem with the design. A good principle to follow is: start simple, and gradually add complexity to the design.

In the previously discussed efficient designs the parameter prior values are assumed known and fixed. Since there is always some uncertainty about the true parameter values, these priors are never known exactly, but only by approximation. In order to take into account the uncertainty about the parameter priors, Bayesian efficient designs have been developed which make use of random priors instead of fixed priors, described by random distributions. All previously described model types can be used in conjunction with random priors. All that is needed is to substitute the fixed prior values with random distributions in the *model* property. We will illustrate this using the example for the MNL and the MMNL models.

In the MNL model, assume that the prior value for parameter ‘b3’ is uncertain and that the prior distribution is a normal distribution with mean 0.5 and standard deviation 0.2. Then instead of having [0.5] as a fixed prior for ‘b3’ it is now a random prior denoted by [(n,0.5,0.2)]. Note that the round brackets within which the distribution is placed distinguishes a Bayesian parameter distribution from a random parameter distribution. That is, the round brackets around the prior value indicate that it is a Bayesian prior, which is not to be confused with a random parameter.

```
;eff = (mnl,d,mean)
;model:
U(alt1) = b1[-0.2] + b2[1.2] * A[0,1,2] + b3[(n,0.5,0.2)] * B[0,1]     /
U(alt2) =            b2      * A        + b4[1.1]         * C[0,1,2,3]
```

Observe that now in the efficiency method *eff* the term ‘mean’ is added as a third argument. When computing the Bayesian D-error over different random draws, one can choose to take the mean value, the minimum or maximum value (‘min’ or ‘max’), or the median value (‘median’) of the efficiency measure being optimised. An additional argument that can be used is ‘fixed’, in which fixed priors values are assumed (set to the mean values of the distribution) instead of Bayesian prior distributions.

In the MMNL model, the random parameters have prior values to describe the distribution, and these prior values can again be Bayesian by assuming a prior distribution. For example, assume that the ‘b2’ parameter is random, following a normal distribution with mean 1.2 and standard deviation 0.3. These two values are not known with certainty, so we could assume prior distributions for them, e.g., a normal distribution for the prior mean, and a uniform distribution for the prior standard deviation:

```
;eff = (rp,d,median)
;model:
U(alt1) = b1[-0.2] + b2[n,(n,1.2,0.2),(u,0.1,0.3)] * A[0,1,2] + b3[0.5]
* B[0,1]     /
U(alt2) =            b2                            * A        + b4[0.4]
* C[0,1,2,3]
```

In this case, the mean of the random parameter ‘b2’ follows a Bayesian normal distribution with mean 1.2 and standard deviation 0.2, while the standard deviation of this random parameter follows a uniform distribution from 0 to 0.3. Note that negative standard deviations should not occur, hence the Bayesian distribution for the standard deviation prior should be chosen with care.

Similarly, Bayesian prior parameter distributions can be used for EC models. For EC models however, the error component term represents a normally distributed random parameter with a mean of zero and an estimated standard deviation parameter. As such, only the Bayesian prior parameter distribution for the standard deviation parameter of the error component need be established. This is shown in the following syntax.

```
;eff = (ecpanel,s,mean)
;model:
U(alt1) = b11[-0.2] + b2[1.2] * A[0,1,2] + b3[(n,0.5,0.2)] * B[0,1]    +
s1[ec,(u,0.8,1.2)]  /
U(alt2) = b12[-0.3] + b2      * A        + b4[1.1]          * C[0,1,2,3]
+ s1                /
U(alt2) =             b2      * A        + b5[0.8]          * D[0,1,2,3]
```

Similar to the MMNL model, for the Bayesian efficient designs several random draws have to be taken from the Bayesian random distributions. This is defined by the property *bdraws* (Bayesian draws) and can have the same arguments as the *rdraws* property, e.g.,

```
;bdraws = halton(100)
;bdraws = gauss(3,4)
```

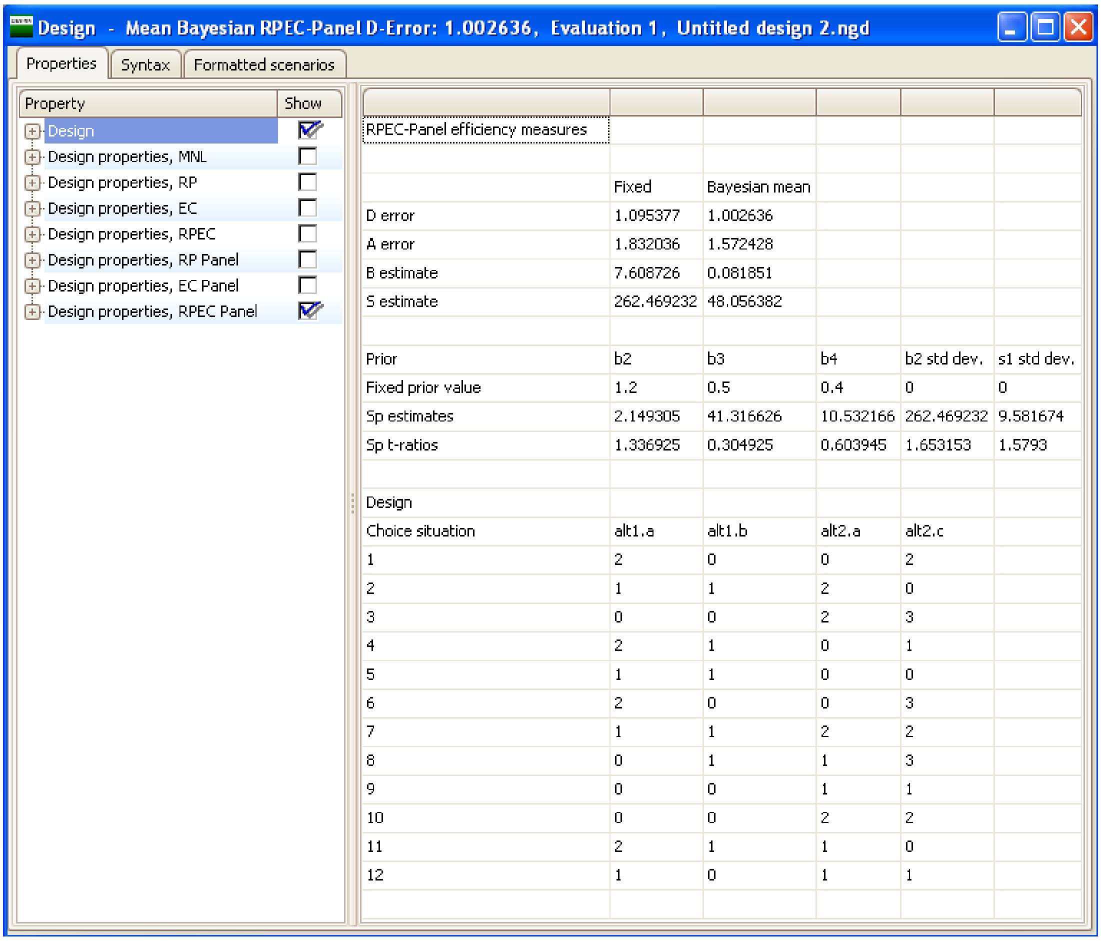
Note that generating designs for the MMNL or EC model with Bayesian priors can be very computationally intensive, even more so if a panel approach is applied. Therefore, the number of random parameters, error components, and Bayesian priors should preferably be limited.

Example syntax demonstrating the use of fixed parameter priors, Bayesian distributions for fixed parameters, Bayesian distributions for random parameter population moments and Bayesian prior parameter distributions for error components is given below.

```
Design
;alts = alt1, alt2,alt3
;rows = 12
;eff = (rpecpanel,d,mean)
;rep = 250
;rdraws = gauss(2)
;bdraws = gauss(2)
;model:
U(alt1)  = b1[-0.2]  + b2[n,(n,1.2,0.2),(u,0.1,0.3)]  *  A[0,1,2]  +  b3
[(n,0.5,0.1)] * B[0,1] + s1[ec,(u,0.8,1.2)]  /
U(alt2) =            b2                              * A         + b4[0.4]
* C[0,1,2,3]     + s1
$
```

In the above example, the ‘b4’ parameter prior is assumed to be fixed and known with exact certainty (i.e., 0.4). The ‘b3’ parameter is assumed to be a fixed (i.e., non-random) parameter but with a Bayesian prior parameter distribution assumed representing some uncertainty as to what the true population parameter will be once data is collected using the design. The ‘b2’ parameter (which is also generic across alternatives ‘alt1’ and ‘alt2’) is assumed to be a random parameter with Bayesian prior parameter distributions for both population moments. The design also allows for an error component with a standard deviation parameter that is not precisely known *a priori* and hence draws values also from a Bayesian prior parameter distribution.

Figure 7.14 shows an example design generated using the above syntax. The output for designs generated assuming Bayesian prior parameter distributions mirrors that given for designs constructed assuming fixed parameters with one exception. In addition to the efficiency measures assuming the fixed priors, Ngene reports the Bayesian efficiency measures used in the optimization routine. Note that when Gaussian quadrature is used, as in this example, only the mean values for each of the efficiency measures is reported. This is a byproduct of how Gaussian quadrature is calculated. When other draw types are used, such as Halton sequences, Ngene will report additional population moments for each of the efficiency measures, as shown in Figure 7.15.

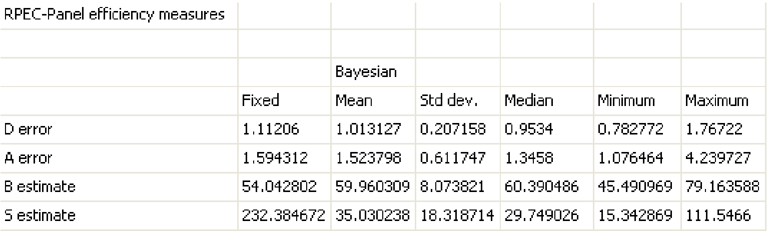


Design - Mean Bayesian RPEC-Panel D-Error: 1.002636, Evaluation 1, Untitled design 2.ngd

| Property | Show |
|---|---|
| ⊞ Design | ☑ |
| ⊞ Design properties, MNL | □ |
| ⊞ Design properties, RP | □ |
| ⊞ Design properties, EC | □ |
| ⊞ Design properties, RPEC | □ |
| ⊞ Design properties, RP Panel | □ |
| ⊞ Design properties, EC Panel | □ |
| ⊞ Design properties, RPEC Panel | ☑ |

| | | | | | |
|---|---|---|---|---|---|
| RPEC-Panel efficiency measures | | | | | |
| | | | | | |
| | Fixed | Bayesian mean | | | |
| D error | 1.095377 | 1.002636 | | | |
| A error | 1.832036 | 1.572428 | | | |
| B estimate | 7.608726 | 0.081851 | | | |
| S estimate | 262.469232 | 48.056382 | | | |
| | | | | | |
| Prior | b2 | b3 | b4 | b2 std dev. | s1 std dev. |
| Fixed prior value | 1.2 | 0.5 | 0.4 | 0 | 0 |
| Sp estimates | 2.149305 | 41.316626 | 10.532166 | 262.469232 | 9.581674 |
| Sp t-ratios | 1.336925 | 0.304925 | 0.603945 | 1.653153 | 1.5793 |
| | | | | | |
| Design | | | | | |
| Choice situation | alt1.a | alt1.b | alt2.a | alt2.c | |
| 1 | 2 | 0 | 0 | 2 | |
| 2 | 1 | 1 | 2 | 0 | |
| 3 | 0 | 0 | 2 | 3 | |
| 4 | 2 | 1 | 0 | 1 | |
| 5 | 1 | 1 | 0 | 0 | |
| 6 | 2 | 0 | 0 | 3 | |
| 7 | 1 | 1 | 2 | 2 | |
| 8 | 0 | 1 | 1 | 3 | |
| 9 | 0 | 0 | 1 | 1 | |
| 10 | 0 | 0 | 2 | 2 | |
| 11 | 2 | 1 | 1 | 0 | |
| 12 | 1 | 0 | 1 | 1 | |

**Figure 7.14: Bayesian efficient design output screen**



| | | | | | | |
|---|---|---|---|---|---|---|
| RPEC-Panel efficiency measures | | | | | | |
| | | | | | | |
| | | Bayesian | | | | |
| | Fixed | Mean | Std dev. | Median | Minimum | Maximum |
| D error | 1.11206 | 1.013127 | 0.207158 | 0.9534 | 0.782772 | 1.76722 |
| A error | 1.594312 | 1.523798 | 0.611747 | 1.3458 | 1.076464 | 4.239727 |
| B estimate | 54.042802 | 59.960309 | 8.073821 | 60.390486 | 45.490969 | 79.163588 |
| S estimate | 232.384672 | 35.030238 | 18.318714 | 29.749026 | 15.342869 | 111.5466 |

**Figure 7.15: Output for non-Gaussian quadrature efficient designs**

Finally, note that it is possible to combine Bayesian prior parameter estimates with dummy effects coded variables. For example,

```
b1.dummy[(n,-0.7,0.2)|(n,0.4,0.2)|0.8]*A[5,10,15,20]
```

will assign Bayesian prior parameters for dummy variables associated with levels ‘5’ and ‘10’, and a fixed prior parameter for the dummy variable associated with the attribute level ‘15’ (the attribute level ‘20’ will be the base level). Similar structures can be applied to effects coded variables.

Ngene also allows the user to optimise the design based on the generalised Asymptotic Fisher Information matrix (see Section 7.1.5). To do this, the command

```
;gfim
```

is added to the syntax. For example, the previous syntax becomes

```
Design
;alts = alt1, alt2,alt3
;rows = 12
;eff = (rpecpanel,d,mean)
;rep = 250
;gfim
;rdraws = gauss(2)
;bdraws = gauss(2)
;model:
U(alt1)  =  b1[-0.2]  +  b2[n,(n,1.2,0.2),(u,0.1,0.3)]  *  A[0,1,2]  +  b3
[(n,0.5,0.1)] * B[0,1] + s1[ec,(u,0.8,1.2)]  /
U(alt2) =           b2                          * A         + b4[0.4]
* C[0,1,2,3]     + s1
$
```

Note that use of the generalised Asymptotic Fisher Information matrix will not change the output generated by Ngene, however it will affect the AVC matrix that the design is being optimised for.

## 7.4 Model averaging of efficient designs

Not only the prior parameter values are uncertain, the precise model type that one is likely to estimate once data is collected using the design may also be uncertain. In order to provide greater flexibility, Ngene is capable of evaluating different models at the same time for a single design. These models may be of a different type, with different utility functions and different priors. Also, different efficiency measures may be used in conjunction and/or Bayesian and fixed priors can be taken into account. Since a single design is evaluated for different models, it is important that the attribute levels specified in each model specification are the same when referring to the same attribute. Not all attributes have to occur in each model specification, although the design generated will contain levels for all attributes (attributes not used in the model specification will simply be ignored when evaluating the efficiency of that model).

As described by Rose *et al.* (2009), Figure 7.16 schematically demonstrates the model average approach. Given a single design, different parameter priors associated with different model types will result in different AVC matrices. Based on these AVC matrices, a single combined AVC matrix can be constructed from which efficiency measures can be calculated. In constructing the combined AVC matrix, different weights can be attached to each of the model types assumed, thus giving different model types different degrees of emphasis in generating the overall design.

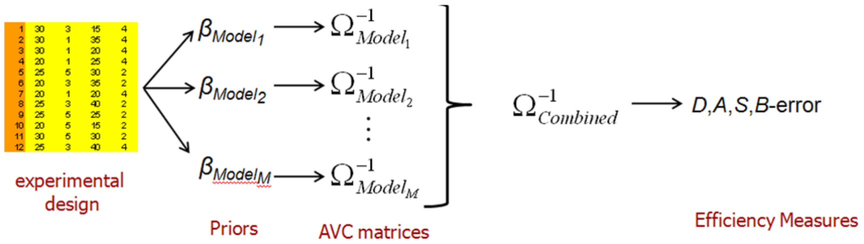


**Figure 7.16: Model average approach**

In Ngene, to describe different models, the *model* property will be set a number of times, and each model will be given a name. For example, below in the syntax we define five models and name them ‘M1’, ‘M2’, ‘M3’, ‘M4’, ‘M5’. Note that in separating the model types, no backslash (i.e., /) is used for the last utility function as would typically be the case.

```
;model(M1):
U(Alt1) = SP1[0.7] + b1[(n,-0.7,0.2)]*A[5,10,15,20] + b2[1.2]*B[0,1,2,3]
+ b3[1.8]*C[0,1,2,3] + b4[-0.6]*D[1,2,3,4] /
U(Alt2) = SP2[0.5] + b1*A                           + b2*B
 + b3*C               + b4*D

;model(M2):
U(Alt1) = SP1[0.6] + b1[(n,-0.6,0.2)]*A[5,10,15,20] + b2[1]*B[0,1,2,3] +
b3[1.5]*C[0,1,2,3] + b4[-0.5]*D[1,2,3,4]  + EC[EC,(U,1,2)] /
U(Alt2) = SP2[0.4] + b1*A                          + b2*B            +
b3*C                + b4*D                  + EC

;model(M3):
U(Alt1)  = SP1[0.8]  + b1[n,(n,-0.8,0.1),(u,0.1,0.2)]*A[5,10,15,20]  + b2
[n,1.2,0.2]*B[0,1,2,3] + b3[1.2]*C[0,1,2,3] + b4[-0.7]*D[1,2,3,4]  /
U(Alt2) = SP2[0.6] + b1*A                                          + b2*B
                      + b3*C                + b4*D

;model(M4):
U(Alt1)  = SP1[10.4]  + b1[n,(n,-1.2,0.1),(u,0.1,0.2)]*A[5,10,15,20]  + b2
[n,1.4,0.3]*B[0,1,2,3] + b3[1]*C[0,1,2,3] + b4[-0.6]*D[1,2,3,4]  /
U(Alt2)  = SP2[10.2]  + b1*A                                             +
b2*B                    + b3*C              + b4*D

;model(M5):
U(Alt1) = SP1[0.7] + b1[(n,-0.5,0.2)]*A[5,10,15,20] + b2[1.1]*B[0,1,2,3]
+ b3[1.2]*C[0,1,2,3] + b4[-0.4]*D[1,2,3,4]  + EC[EC,(U,2,3)] /
U(Alt2) = SP2[0.5] + b1*A                           + b2*B
 + b3*C               + b4*D                  + EC
```

The first model is a MNL model with some Bayesian priors, the second and fourth models are MMNL models whilst the third and fifth are EC models. Note that the priors for the unique models can be different, and although not shown here, not all parameters and attributes need appear in all model utility functions.

When generating an efficient design for multiple models at the same time, a weighted efficiency measure is computed and optimised on. The *eff* property has to be changed to compute this weighted efficiency measure, for example as follows:

```
;eff = M1(mnl,d,mean) + 2*M2(rppanel,d,mean) + 1.5*M3(ecpanel,d,mean) +
M4(rp,d,mean) + M5(ec,d,mean)
```

The efficiency in this example consists of the Bayesian D-errors for MNL model ‘M1’, MMNL models ‘M2’ and ‘M4’ and EC models ‘M3’ and ‘M5’. Whilst it is possible to mix different efficiency measures in this procedure Rose *et al*. (2009) suggest against this as each measure is based on a different metric which may cause one efficiency measure to dominate all the others. In the above example, note how we have suggested multiplying the efficiency measure for model ‘M2’ by two and ‘M3’ by 1.5. In this way, the efficiency measures for these models will be given these amounts of weight more than the efficiency measures of the remaining models. Note that if no weight is provided, then the efficiency measure for that model is automatically weighted by one.

In the above example, all alternatives in all model specifications are the same; each model uses ‘alt1’, ‘alt2’, and ‘alt3’. In general, these may be different for each model specification as well. In that case, the *alts* property has to be defined for each model, such as:

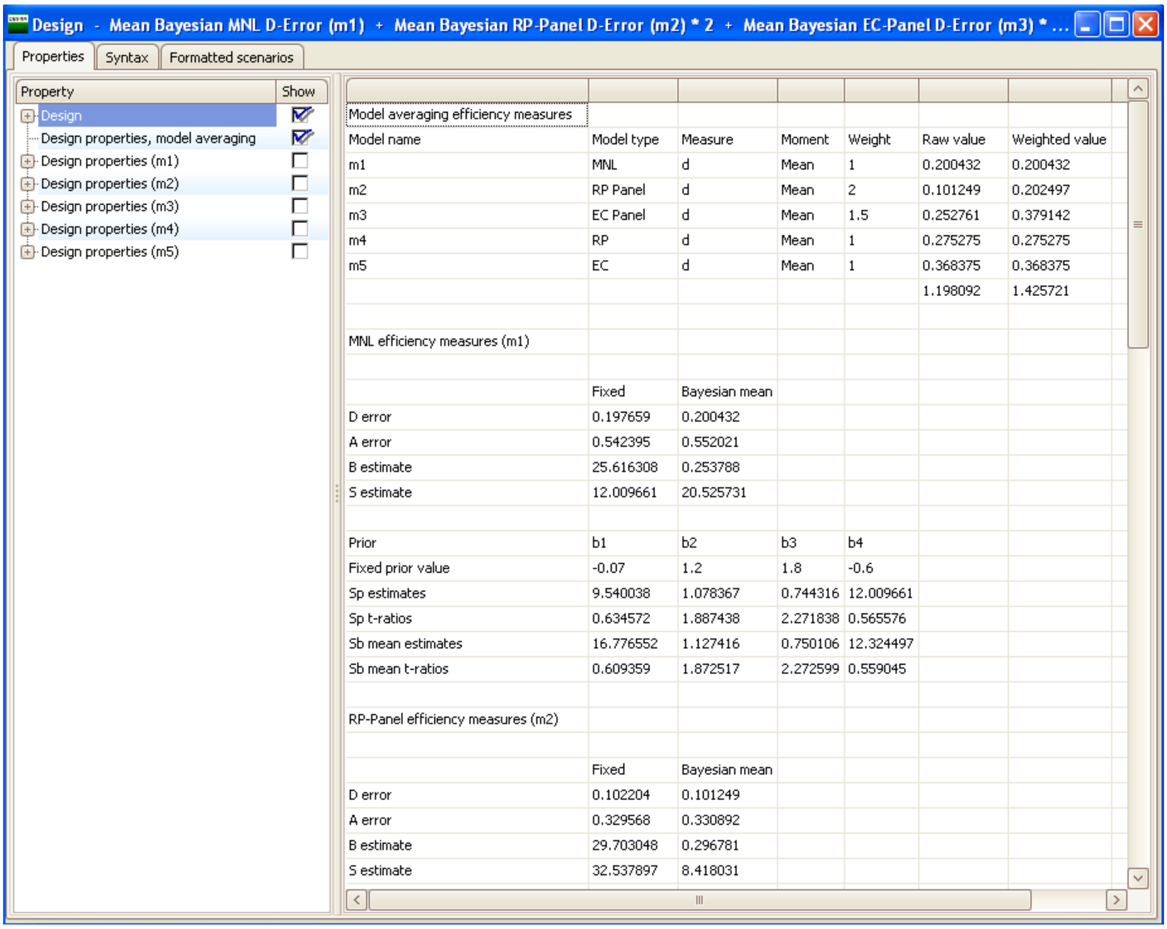
```
;alts(M1) = alt1, alt2, alt4
;alts(M2) = alt1, alt3, alt5, alt6
```

Other properties set in the syntax, such as *rdraws*, *bdraws*, and *rep*, apply to all models specified.

Example syntax using the model averaging approach to generating an efficient design is provided below.

```
Design
;alts(m1) = alt1, alt2, alt3
;alts(m2) = alt1, alt2, alt3
;alts(m3) = alt1, alt2, alt3
;alts(m4) = alt1, alt2, alt3
;alts(m5) = alt1, alt2, alt3
;rows = 16
;eff = M1(mnl,d,mean) + 2*M2(rppanel,d,mean) + 1.5*M3(ecpanel,d,mean) +
M4(rp,d,mean) + M5(ec,d,mean)
;rdraws=gauss(3)
;bdraws=gauss(3)
;rep=250

;model(M1):
U(Alt1) = SP1[-3.2] + b1[(n,-0.07,0.03)]*A[5,10,15,20] + b2[(n,1.2,0.2)]
*B[0,1,2,3] + b3[1.8]*C[0,1,2,3] + b4[-0.6]*D[1,0] /
U(Alt2) = SP2[-3.4] + b1*A                             + b2*B
            + b3*C               + b4*D

;model(M2):
U(Alt1) = SP1[-2.4] + b1[n,(n,-0.08,0.01),(u,0.02,0.04)]*A[5,10,15,20] +
b2[n,1.2,0.4]*B[0,1,2,3] + b3[1.2]*C[0,1,2,3] + b4[-0.7]*D[1,0]  /
U(Alt2) = SP2[-2.2] + b1*A                                               +
b2*B                    + b3*C                 + b4*D

;model(M3):
U(Alt1) = SP1[-3] + b1[(n,-0.06,0.02)]*A[5,10,15,20] + b2[1]*B[0,1,2,3]+
b3[1.5]*C[0,1,2,3] + b4[-0.5]*D[1,0]  + EC[EC,(U,1,2)] /
U(Alt2) = SP2[-2.8] + b1*A                          + b2*B          +
b3*C                +b4*D               + EC

;model(M4):
U(Alt1) = SP1[-3.2] + b1[n,(n,-0.02,0.01),(u,0.01,0.03)]*A[5,10,15,20] +
b2[n,1.4,0.3]*B[0,1,2,3] + b3[1]*C[0,1,2,3] + b4[-0.6]*D[1,0]  /
U(Alt2) = SP2[-3] + b1*A                                                 +
b2*B                    + b3*C               + b4*D

;model(m5):
U(Alt1)  =  SP1[-3.3]  +  b1[(n,-0.05,0.02)]*A[5,10,15,20]  +  b2[1.1]*B
[0,1,2,3]+ b3[1.2]*C[0,1,2,3] + b4[-0.4]*D[1,0]  + EC[EC,(U,1.5,2.5)] /
U(Alt2) = SP2[-3.2] + b1*A                             + b2*B
   + b3*C                + b4*D             + EC

$
```

Example output based on the above syntax is given in Figure 7.17. Note that the output provided is similar to that provided for non model average efficient designs, although Ngene now also reports the weights applied to the various model types in generating the design as well as the unweighted and weighted efficiency measure values.


Design - Mean Bayesian MNL D-Error (m1) + Mean Bayesian RP-Panel D-Error (m2) * 2 + Mean Bayesian EC-Panel D-Error (m3) * ...
Properties
Syntax
Formatted scenarios

| Property | Show |
|---|---|
| Design | ☑ |
| Design properties, model averaging | ☑ |
| Design properties (m1) | □ |
| Design properties (m2) | □ |
| Design properties (m3) | □ |
| Design properties (m4) | □ |
| Design properties (m5) | □ |

| | | | | | | |
|---|---|---|---|---|---|---|
| Model averaging efficiency measures | | | | | | |
| Model name | Model type | Measure | Moment | Weight | Raw value | Weighted value |
| m1 | MNL | d | Mean | 1 | 0.200432 | 0.200432 |
| m2 | RP Panel | d | Mean | 2 | 0.101249 | 0.202497 |
| m3 | EC Panel | d | Mean | 1.5 | 0.252761 | 0.379142 |
| m4 | RP | d | Mean | 1 | 0.275275 | 0.275275 |
| m5 | EC | d | Mean | 1 | 0.368375 | 0.368375 |
| | | | | | 1.198092 | 1.425721 |
| | | | | | | |
| MNL efficiency measures (m1) | | | | | | |
| | | | | | | |
| | Fixed | Bayesian mean | | | | |
| D error | 0.197659 | 0.200432 | | | | |
| A error | 0.542395 | 0.552021 | | | | |
| B estimate | 25.616308 | 0.253788 | | | | |
| S estimate | 12.009661 | 20.525731 | | | | |
| | | | | | | |
| Prior | b1 | b2 | b3 | b4 | | |
| Fixed prior value | -0.07 | 1.2 | 1.8 | -0.6 | | |
| Sp estimates | 9.540038 | 1.078367 | 0.744316 | 12.009661 | | |
| Sp t-ratios | 0.634572 | 1.887438 | 2.271838 | 0.565576 | | |
| Sb mean estimates | 16.776552 | 1.127416 | 0.750106 | 12.324497 | | |
| Sb mean t-ratios | 0.609359 | 1.872517 | 2.272599 | 0.559045 | | |
| | | | | | | |
| RP-Panel efficiency measures (m2) | | | | | | |
| | | | | | | |
| | Fixed | Bayesian mean | | | | |
| D error | 0.102204 | 0.101249 | | | | |
| A error | 0.329568 | 0.330892 | | | | |
| B estimate | 29.703048 | 0.296781 | | | | |
| S estimate | 32.537897 | 8.418031 | | | | |

**Figure 7.17: Model average output screen**

The model averaging approach outlined here may also be used to examine different possible utility specifications for the same model. For example, the analyst may be unsure as to whether they will use dummy or effects codes or not post data collection. In such a case, the analyst may utilize the same model form (e.g., MNL), but using the model averaging approach, specify linear in the attributes for one model and dummy and/or effects codes for another model. For example, the syntax below assumes a linear in the attributes specification for model ‘M1” but a nonlinear specification using dummy coding for model ‘M2’. Similarly, one can use the same process to average models with and without a no choice alternative if one is not sure what choice will be used in the final experiment, or if a dual choice process will be used.

```
Design
;alts(M1) = alt1, alt2, alt3
;alts(M2) = alt1, alt2, alt3
;rows = 20
;eff = M1(mnl,d,mean) + M2(mnl,d,mean)

;bdraws=halton(150)

;model(M1):
U(Alt1) = SP1[0.7] + b1[(n,-0.7,0.2)]*A[5,10,15,20] + b2[1.2]*B[0,1,2,3]
+ b3[1.8]*C[0,1,2,3] /
U(Alt2) = SP2[0.5] + b1*A                           + b2*B
 + b3*C

;model(M2):
U(Alt1)    =    SP1[1.2]    +    b1.dummy[(n,-0.7,0.2)|(n,0.4,0.2)|0.8]*A
[5,10,15,20] + b2[1.2]*B[0,1,2,3] + b3[1.8]*C[0,1,2,3] /
U(Alt2) = SP2[0.9] + b1*A
 + b2*B                + b3*C

$
```

## 7.5 Appendix 7A Discrete choice models

In this section, we outline the differences between the MNL, MMNL, and EC models. We begin by examining different conceptualizations of utility specifications that result in each of the different model formulations before discussing how these utility specifications impact upon the choice probabilities and log-likelihood functions of each of the models.

### 7.5.1 Utility specification

Let $U_{nsj}$ denote the utility of alternative *j* perceived by respondent *n* in choice situation s. $U_{nsj}$ may be partitioned into three separate components, an observed component of utility, $V_{nsj}$, an unobserved (or un-modeled) component of utility, $\eta_{nsj}$, and an unobserved (and un-modeled) component, $\varepsilon_{nsj}$, such that

$$U_{nsj} = V_{nsj} + \eta_{nsj} + \varepsilon_{nsj} \qquad (7A.1)$$

The observed component of utility is typically assumed to be a linear relationship of observed attribute levels of each alternative, *x*, and their corresponding weights (parameters), *β*. In the MNL model, the parameter weights for each attribute are invariant over respondents, such that the observed component of utility may be represented as

$$V_{nsj} = \sum_{k=1}^{K} \beta_{jk} x_{nsjk}. \qquad (7A.2)$$

Unlike the MNL model, some or all of the parameter weights of the MMNL model are assumed to vary with density *f*(*β* | Ω) over the sampled population. Assumptions as to how these parameter weights vary over the population have in the past resulted in two different formulations of the MMNL model. One version of the model, known as the cross sectional MMNL formulation, assumes that the parameter weights vary with density over both *n* and *s* suggesting that preference heterogeneity exists both within and between individuals, even when the same individual is observed to make *s* choices within a similar choice context. The second version of the model, known as the panel MMNL formulation, assumes that preferences vary between individuals but not within. The assumption that preferences vary between and not within respondents accounts for the pseudo panel nature of SP data (Ortúzar and Willumsen, 2001; Revelt and Train, 1998; Train, 2003). Equations (7A.3a) and (7A.3b) represent the observed components of utility under both the cross sectional and panel formulations of the MMNL model specifications.

$$V_{nsj} = \sum_{k=1}^{K} \beta_{nsk} x_{nsjk}, \qquad \text{(7A.3a)}$$

$$V_{nsj} = \sum_{k=1}^{K} \beta_{nk} x_{nsjk}. \qquad \text{(7A.3b)}$$

Like the MMNL model, the EC model involves estimation of one or more random parameters. Unlike the MMNL model however, the random parameter estimates of the EC model are associated with alternatives, *j*, not attributes, *x*. To estimate the model, the analyst first specifies a set of dummy variables, with each dummy variable able to appear in the utility specifications of up to *J*-1 alternatives. Next, generic normally distributed random parameters with means normalised to zero, represented as $\eta_{nsj}$ in Equation (7A.1), are estimated for each of the defined dummy variables. By associating each $\eta_{nsj}$ with different subsets of alternatives, the parameters (which represent standard deviations set around a mean of zero) capture different common error variances associated with those alternatives for which they are estimated for. Note that utility specifications with alternative specific constants and alternative specific error components will be equivalent to a MMNL model with normally distributed random constant terms. Also, as with the MMNL model, the random parameters of the EC model may be estimated with density over both *n* and *s* (cross sectional EC model) or only over *n* (panel EC model).

Assuming the analyst fails to specify error components as part of the utility functions of the model, then Equation (7A.1) will collapse to

$$U_{nsj} = V_{nsj} + \varepsilon_{nsj} \qquad \text{(7A.4)}$$

which represents the most common form of utility representation within the literature.

Finally, for all logit type models, the second unobserved component of utility, $\varepsilon_{nsj}$, are assumed to be identically and independently extreme value type 1 (EV1) distributed.

### 7.5.2 Model probabilities

Depending on the assumptions made about the utility specifications as outlined above, different functional forms of the logit model will be arrived at. We now outline in turn how the assumptions made about the different models influence the choice probabilities derived for each of the models.

#### The MNL Model

The choice probabilities of the MNL model are derived from a number of assumptions about the choice behaviour of respondents. In particular, aside from the assumption that $\varepsilon_{nsj}$ are IID EV1, the MNL model assumes that the marginal utilities for the attributes and variables specified within the system of utility equations are fixed for the sampled population and that $\eta_{nsj} = 0$. Under these assumptions, the probability, $P_{nsj}$, that respondent $n$ chooses alternative $j$ in choice situation $s$ is given by

$$P_{nsj} = \frac{\exp\left(V_{nsj}\right)}{\sum_{i \in J_{ns}} \exp\left(V_{nsi}\right)}. \qquad (7A.5)$$

**MMNL and EC Models**

Both the MMNL and EC models differ from the MNL model in that we now assume that (some of) the parameters (or error components) are random, following a certain probability distribution. The choice probabilities of the MMNL model therefore depend on the random parameters. Both models utilize the MNL probabilities given in Equation (7A.5), however rather than calculate a single probability for each alternative, both models calculate the choice probabilities for each random draw taken from the assumed probability distribution(s). In this way, multiple choice probabilities are obtained for each alternative, as opposed to a single set of probabilities as obtained from the MNL model. It is the expectation of these probabilities over the random draws which are calculated and used in the model estimation process. The expected choice probabilities for the MMNL logit and EC models are given in Equations (7A.6a) and (7A.7b) respectively.

$$E\left[P_{nsj}\right] = \int_{\beta} \frac{\exp\left(V_{nsj}\right)}{\sum_{i \in J_{ns}} \exp\left(V_{nsi}\right)} f\left(\beta \mid \theta\right) d\beta, \qquad (7A.6a)$$

$$E\left[P_{nsj}\right] = \int_{\eta} \frac{\exp\left(V_{nsj}\right)}{\sum_{i \in J_{ns}} \exp\left(V_{nsi}\right)} f\left(\eta \mid \theta\right) d\eta. \qquad (7A.6b)$$

Equations (7A.6a) and (7A.6b) provide the choice probabilities at the level of the alternatives. In the cross sectional formulations of the MMNL and EC models, it is these probabilities that are used directly in model estimation. In the panel formulations of the MMNL and EC models, the choice probabilities given in Equations (7A.7a) and (7A.7b), whilst calculated, are not of direct interest. Rather, what are of interest are the probabilities of observing the sequence of choices made by each respondent, not the probabilities that specific alternatives will be observed to be chosen. To this end, we define the probability $P_n^*$ that a certain respondent $n$ has made a certain sequence of choices $\{j \mid y_{nsj} = 1\}_{s \in S_n}$ with respect to the set of choice situations, $S_n$, by

$$P_n^* = \int_{\beta} \prod_{s \in S_n} \prod_{j \in J_{ns}} \left(P_{nsj}\right)^{y_{nsj}} f\left(\beta \mid \theta\right) d\beta, \qquad (7A.7a)$$

$$P_n^* = \int_{\eta} \prod_{s \in S_n} \prod_{j \in J_{ns}} \left(P_{nsj}\right)^{y_{nsj}} f\left(\eta \mid \theta\right) d\eta, \qquad (7A.7b)$$

for the MMNL and EC models respectively.

## 7.5.3 Model log-likelihood functions

Typically, the parameters $\beta$ contained within each $V_{nsi}$ are unknown and must be estimated from data. Let $y_{nsj}$ equal one if $j$ is the chosen alternative in choice situation $s$ shown to respondent $n$, and zero otherwise. Then the parameters can be estimated by maximizing the likelihood function $L$,

$$L = \prod_{n=1}^{N} \prod_{s \in S_n} \prod_{j \in J_{ns}} \left( P_{nsj} \right)^{y_{nsj}}. \qquad (7A.8)$$

where *N* denotes the total number of respondents and $S_n$ is the set of choice situations faced by respondent *n*.

Rather than maximize the likelihood function, it is more common to maximize the log of the likelihood function instead. This is because taking the product of a series of probabilities will typically produce values that are extremely small and which most computing software packages will be unable to adequately handle. By taking the logs of the probabilities first, large negative values will result, which when multiplied, produce even larger negative values. As such, the log-likelihood function of the model, shown below, is typically preferred.

$$LL = \ln \left[ \prod_{n=1}^{N} \prod_{s \in S_n} \prod_{j \in J_{ns}} \left( P_{nsj} \right)^{y_{nsj}} \right]. \qquad (7A.9)$$

In the sections that follow, we attempt to differentiate between the log-likelihood functions of the various models available in Ngene.

### The MNL Model

In order to derive the log-likelihood function of the MNL model, an assumption is made that all choice observations are independent of each other. That is, even in data where the same individual is observed to make multiple choices, the log-likelihood function of the MNL model treats the data as if the observed choices have been made by separate pseudo individuals. Using the mathematical properties ln($n_1n_2$) = ln($n_1$) + ln($n_2$) and $\ln(n_1)^{y_{njs}} = y_{njs}\ln(n_1)$, and applying the same mathematical rules to choice tasks, *s*, and alternatives, *j*, this independence of choice observations assumption results in Equation (7A.9) being rewritten in the more commonly known form of

$$LL = \sum_{n=1}^{N} \sum_{s \in S_n} \sum_{j \in J_{ns}} y_{nsj} \ln \left( P_{nsj} \right) \qquad (7A.10)$$

The Log-likelihood function of the MNL model given in Equation (7A.10) will be globally concave for linear in the parameters utility specifications (see McFadden 1974) suggesting that there should exist a single set of parameter estimates that will maximise this function.

### Cross Sectional MMNL and EC Models

The log-likelihood functions of the cross sectional MMNL and EC models are derived under the same assumptions of choice observation independence as made with the MNL model. The difference between these two models and the MNL model however is that the choice probabilities used for the MNL are replaced with the expected choice probabilities given in Equations (7A.6a) and (7A.6b). Using the same mathematical rules used to derive the MNL model log-likelihood function, and noting additionally that *E*($n_1n_2$) = *E*($n_1$)*E*($n_2$), the log-likelihood functions of the cross sectional MMNL and EC models may be represented as

$$LL = \sum_{n=1}^{N} \sum_{s \in S_n} \sum_{j \in J_{ns}} y_{nsj} \ln \left[ E \left( P_{nsj} \right) \right] \qquad (7A.11)$$

### Panel MMNL and EC Models

The derivation of the log-likelihood functions of the panel formulations of the MMNL and EC models differ to those of their equivalent cross sectional forms, as well as to that of the MNL model, in that the choice observations are no longer assumed to be independent within each respondent (although the independence across respondents assumption is maintained).

Mathematically, this means that $E(s_1s_2) \neq E(s_1)E(s_2)$, and hence we are no longer able to invoke the mathematical rule $\ln(s_1s_2) = \ln(s_1) + \ln(s_2)$. Given this, the log-likelihood functions of the panel MMNL and EC models may respectively be represented as

$$LL = \sum_{n=1}^{N} \ln \int_{\beta} \prod_{s \in S_n} \prod_{j \in J_{ns}} \left(P_{nsj}\right)^{y_{nsj}} f\left(\beta \,|\, \theta\right) d\beta \qquad (7A.12a)$$

$$LL = \sum_{n=1}^{N} \ln \int_{\eta} \prod_{s \in S_n} \prod_{j \in J_{ns}} \left(P_{nsj}\right)^{y_{nsj}} f\left(\eta \,|\, \theta\right) d\beta \qquad (7A.12a)$$

or

$$LL = \sum_{n=1}^{N} \ln\left(P_n^*\right) \qquad (7A.12c)$$

In the next section we outline the AVC matrices of each of the model types available in Ngene.

## 7.5.4 Model variance-covariance matrices

The generation of efficient SC experiments requires first an estimation of the AVC matrix of the design, $\Omega_N$.. The AVC matrix $\Omega_N$ can be determined as the inverse of the Fisher Information matrix, $I_N$, which in turn can be computed using the second derivatives of the log-likelihood function of the discrete choice model to be estimated (see Train, 2003). Mathematically, the AVC matrix for the MNL may be represented as

$$\Omega_N = I_N^{-1}, \quad \text{with } I_N = -E_N\left(\frac{\partial^2 \log LL}{\partial\beta\partial\beta'}\right) \qquad (7A.13a)$$

whilst the AVC matrix of the MMNL and EC models becomes

$$\Omega_N = I_N^{-1}, \quad \text{with } I_N = -E_N\left(\frac{\partial^2 \log E(LL)}{\partial\theta\partial\theta'}\right) \qquad (7A.13b)$$

where $E_N(.)$ is used to express the large sample population mean. Hence, the AVC matrix can be determined by calculating the Hessian matrix of the log-likelihood function for the specific model.

As was seen in Appendix 7A.3, different discrete choice models have different log-likelihood functions. Given that the AVC matrix of a discrete choice model is calculated as the inverse of the second derivatives of the log-likelihood function of that model, it is clear that each model will also yield a different AVC matrix. In this section, we reproduce the second derivatives of the log-likelihood functions for each of the models available in Ngene.

### The MNL Model

The second derivatives of the log-likelihood function of the MNL depend on whether the parameter estimates are generic or alternative specific (see Bliemer and Rose, 2005b). Let $x^*_{nsj}$ and $x_{nsj}$ represent attributes for which generic, given as $\beta^*$, and alternative specific, represented by $\beta_j$, parameters are to be estimated for respectively. Assuming that all respondents face the same choice situations, *s*, the second derivatives of the MNL log-likelihood function yields the following expressions (see Rose and Bliemer, 2005b)

$$\frac{\partial^2 LogL}{\partial\beta^*_{k_1}\partial\beta^*_{k_2}} = -\sum_{n=1}^{N}\sum_{s\in S_n}\sum_{j\in J_{ns}} x^*_{nsjk_1} P_{nsj}\left(x^*_{nsjk_2} - \sum_{i=1}^{J} P_{nsi} x^*_{nsik_2}\right) \quad \text{(7A.14a)}$$

$$\frac{\partial^2 LogL}{\partial\beta_{j_1k_1}\partial\beta^*_{k_2}} = -\sum_{n=1}^{N}\sum_{s\in S_n}\sum_{j\in J_{ns}} x_{j_1k_1s} P_{j_1s}\left(x^*_{j_1k_2s} - \sum_{i=1}^{J} x^*_{ik_2s} P_{is}\right) \quad \text{(7A.14b)}$$

$$\frac{\partial^2 LogL}{\partial\beta_{j_1k_1}\partial\beta_{j_2k_2}} = \begin{cases} \sum_{n=1}^{N}\sum_{s\in S_n}\sum_{j\in J_{ns}} x_{j_1k_1s} x_{j_2k_2s} P_{j_1s} P_{j_2s}, & \text{if } j_1 \neq j_2; \\ -\sum_{n=1}^{N}\sum_{s\in S_n}\sum_{j\in J_{ns}} x_{j_1k_1s} x_{j_2k_2s} P_{j_1s}\left(1 - P_{j_2s}\right), & \text{if } j_1 = j_2. \end{cases} \quad \text{(7A.14b)}$$

Note that the choice index, $y_{nsj}$, drops out of the second derivatives of the MNL log-likelihood function, with only the design, *x*, and choice probabilities remaining. Given this result, it is not necessary to know *a prior* what alternatives will be chosen in the sample data in order to calculate the expected AVC matrix of the model. All the analyst requires to know is the design, and the choice probabilities. Given that the choice probabilities are a function of the design as well as the parameter estimates (see Equation (7A.5)), in generating an efficient design, the analyst is required to make certain assumptions regarding the parameter estimates in advance.

### Cross Sectional MMNL and EC Models

The AVC matrix of the MMNL and EC models are somewhat more complicated than those of the MNL model given that the parameter and error component estimates are now assume to be randomly distributed. Let $M_k$ represent a vector of parameters related to the probability distributions of the *k* (either random or error component) parameters, $\beta_k$, denoted by $\Theta_k = [\Theta_{km}]$, where m = 1, ..., $M_k$. The second derivatives of this model is given as

$$\frac{\partial^2 \log E(L)}{\partial\theta_{k_1m_1}\partial\theta_{k_2m_2}} = \sum_{n=1}^{N}\sum_{s\in S_n}\sum_{j\in J_{ns}} y_{nsj}\left(\frac{1}{E\left(P_{nsj}\right)}E\left(\frac{\partial^2 P_{nsj}}{\partial\theta_{k_1m_1}\partial\theta_{k_2m_2}}\right) - \frac{1}{\left(E\left(P_{nsj}\right)\right)^2}E\left(\frac{\partial P_{nsj}}{\partial\theta_{k_1m_1}}\right)E\left(\frac{\partial P_{nsj}}{\partial\theta_{k_2m_2}}\right)\right). \quad \text{(7A.15)}$$

Unfortunately, unlike the MNL model, the choice index, $y_{nsj}$, does not drop out when taking the second derivatives of the log-likelihood function of this model. Thus, in order to derive Equation (7A.15), we are forced to rely on asymptotic theory and substitute $E_N(y_{nsj}) = E(P_{nsj})$, where $E_N(.)$ is again the large sample mean. In this way, Equation (7A.15) becomes equivalent to that given in Sándor and Wedel (2002).

### Panel MMNL and EC Models

Relative to the other models explored herein, the second derivatives of the log-likelihood functions of the panel MMNL and EC models are far more complex to compute as a result of the product terms resident in Equations (7A.12a) to (7A.12b). Nevertheless, such derivations are possible. Bliemer and Rose (2009) show that the second derivatives of Equation (7A.12c) is

$$\begin{aligned}\frac{\partial^2 \log E(L)}{\partial\theta_{k_1m_1}\partial\theta_{k_2m_2}} &= \sum_{n=1}^{N}\left(\frac{1}{E\left(P_n^*\right)}\frac{\partial^2 E\left(P_n^*\right)}{\partial\theta_{k_1m_1}\partial\theta_{k_2m_2}} - \frac{1}{\left(E\left(P_n^*\right)\right)^2}\frac{\partial E\left(P_n^*\right)}{\partial\theta_{k_1m_1}}\frac{\partial E\left(P_n^*\right)}{\partial\theta_{k_2m_2}}\right) \\ &= \sum_{n=1}^{N}\left(\frac{1}{E\left(P_n^*\right)}E\left(\frac{\partial^2 P_n^*}{\partial\theta_{k_1m_1}\partial\theta_{k_2m_2}}\right) - \frac{1}{\left(E\left(P_n^*\right)\right)^2}E\left(\frac{\partial P_n^*}{\partial\theta_{k_1m_1}}\right)E\left(\frac{\partial P_n^*}{\partial\theta_{k_2m_2}}\right)\right),\end{aligned} \tag{7A.16}$$

where

$$\frac{\partial P_n^*}{\partial\theta_{km}} = P_n^* \frac{\partial\beta_k}{\partial\theta_{km}} \sum_{s\in S_n}\sum_{j\in J_{ns}} \frac{y_{nsj}}{P_{nsj}}\frac{\partial P_{nsj}}{\partial\beta_k}, \tag{7A.17}$$

and

$$\begin{aligned}\frac{\partial^2 P_n^*}{\partial\theta_{k_1m_1}\partial\theta_{k_2m_2}} = \frac{1}{P_n^*}\frac{\partial P_n^*}{\partial\theta_{k_1m_1}}\frac{\partial P_n^*}{\partial\theta_{k_2m_2}} - P_n^*\frac{\partial\beta_{k_1}}{\partial\theta_{k_1m_1}}\frac{\partial\beta_{k_1}}{\partial\theta_{k_2m_2}}\sum_{s\in S_n}\sum_{j\in J_{ns}}\frac{\partial P_{nsj}}{\partial\beta_{k_2}}x_{nsjk_1} \\ + P_n^*\frac{\partial^2\beta_{k_1}}{\partial\theta_{k_1m_1}\partial\theta_{k_2m_2}}\sum_{s\in S_n}\sum_{j\in J_{ns}}\frac{y_{nsj}}{P_{nsj}}\frac{\partial P_{nsj}}{\partial\beta_{k_1}},\end{aligned} \tag{7A.18}$$

and where $\partial P_{nis} / \partial\beta_k$ is the first derivative of the MNL probability,

$$\frac{\partial P_{nsj}}{\partial\beta_k} = P_{nsj}\left(x_{nsjk} - \sum_{i\in J_{ns}} P_{nsi}x_{nsik}\right). \tag{7A.19}$$

As with the cross sectional MMNL and EC models, the choice index, $y_{nsj}$, does not drop out when taking the second derivatives of the log-likelihood function of this model. Nevertheless, it is possible once more to for the choice outcomes to be replaced by probabilities, since $E_N(y_{nsj}) = P_{nsj}$ ($y$ follows a multinomial distribution). However, $E_N(P_N^*)$ cannot be approximated that easily, as it describes a generalized multinomial distribution (Beaulieu, 1991). It is therefore necessary, unlike for designs generated specifically for the MNL and cross sectional MMNL and EC models, to simulate a sample based on the design $x$ in order to calculate the second derivatives of the model. To do this, for each respondent $n$, we first draw a random parameter $\beta_k$ from each given parameter distribution, then determine the observed utility $V_{nsj}$ for each choice situation $s$ based on design $x$. Next we separately draw random values for the unobserved component $\varepsilon_{nsj}$ for each alternative in each choice situation, and determine $y_{nsj}$ by selecting the alternative with the highest utility in each choice situation. Note that the same random draw for $\beta_k$ is used over all choice situations for each respondent, representing the panel formulation.

## 7.6 Appendix 7B Steps in generating efficient stated choice designs

Designs which attempt to minimise the elements contained within the AVC matrix are referred to as efficient choice designs. We now go on to discuss the generation process for efficient choice designs.

*Step 1*: Specify the utility specification for the likely final model to be estimated from data collected using the SC design. This involves determining (i) what parameters will be generic and alternative specific; (ii) whether attributes will enter the utility function as dummy/effects codes or some other format; (iii) whether main effects only or interaction terms will be estimated; (iv) the values of the parameter estimates likely to be obtained once the model is estimated; and (v) the precise econometric model that is likely to be estimated from data collected using the experimental design. Points (i) to (iii) impact directly upon the design matrix X, whereas point (iv) influences the AVC matrix via the choice probabilities and point (v) via the choice probabilities as well as influencing the dimensionality of the AVC matrix itself.

Point (iv) represents the most divisive aspect of generating efficient choice designs. In order to estimate the AVC matrix of a design, point (iv) suggests that the analyst is required to have a priori knowledge of the parameter estimates that will be achieved using the design, even though the design has not yet been constructed. Fortunately, the analyst does not have to assume exact knowledge of these *parameter priors* (e.g., the price parameter will be -0.4), but can use Bayesian methods to reflect imperfect knowledge of the exact parameter value (e.g., the price parameter may be drawn from a normal distribution with a mean of -0.4 and a standard deviation of 0.2, or from a uniform distribution with a range between -1 and zero; see for example Sándor and Wedel 2001). Independent of how the priors are treated, two methods, namely numerically by simulation or analytical derivation (discussed in step 4) can be used to approximate the AVC matrix.

Point (v), determining the econometric model influences the AVC matrix not via the X matrix, but in terms of the parameter estimates represented within the AVC matrix. For example, designs assuming MNL will require only parameters related to each of the design attributes whereas designs generated for NL models will require consideration of the scale parameters and designs constructed for MMNL models will require elements in the AVC to be associated with the standard deviation or spread parameters. Given interdependencies between the values that populate the AVC matrix of discrete choice models, one cannot simply assume that a design that minimises the elements contained within the AVC for one model form will necessarily minimise the AVC matrix for another model form.

*Step 2*: Randomly populate the design matrix, *X*, to create an initial design. Unlike OOD designs, the initial design need not be orthogonal, although if the analyst wishes to retain orthogonally it should be. The initial design, however, should incorporate all the constraints that the analyst wishes to impose upon the final design outcome. For example, if the analyst wishes to retain attribute level balance, then the initial design should display this property. The initial design can be constructed with the desired number of rows, however the number of rows should be greater than or equal to *K*/(*J*-1). The utility specification expressed in step 1 should act as a handy guide in determining the minimum number of choice situations to use. Similarly, step 1 should help determine the number of columns that make up the *X* matrix; one for each attribute (or attribute level minus one in terms of dummy or effects coded attributes). In constructing the *X* matrix, the precise levels that will likely be used later during estimation should be used. That is, if an attribute is likely to be dummy coded in estimation, then the *X* matrix should reflect this. Similarly, if a quantitative attribute is to be estimated exactly as shown to a respondent during the survey (e.g., a price attribute takes on the levels $2, $4 and $6), then these values should be used to populate the *X* matrix. Note that different attributes may take on different coding schemes. Typically, a single design would be constructed that will be applied to the entire sample population; however, multiple designs might be generated corresponding to different sub segments of the sampled population (see e.g., Sándor and Wedel 2005 and Rose and Bliemer 2006).

*Step 3*: For the design, calculate the choice probabilities for each alternative in the design. For the MNL and NL models calculating the choice probabilities is relatively straightforward when fixed parameter priors are used (e.g., the price parameter is -0.4). When parameter priors are drawn from Bayesian distributions, the analyst is required to take a number of draws from the given random distributions and calculate the choice probability for each set of draws. Unlike the estimation process of the MMNL model, the average probability is not calculated, but rather the average efficiency measure is used (as calculated in step 5).

For designs assuming a MMNL, EC or probit model form, draws must be taken using the same procedures as when estimating the parameters in order to calculate the choice probabilities at each draw. When draws are taken from a Bayesian distribution for such models however, different distributions may be required for each random parameter population moment (e.g., mean and standard deviation). Bliemer *et al*. (2008) examined the use of various types of draws when drawing from Bayesian parameter distributions. They conclude that the predominantly employed method of using pseudo Monte Carlo draws is unlikely to result in leading to truly Bayesian efficient SC designs and that quasi Monte Carlo methods (e.g., using Halton or Sobol draws), Modified Latin Hypercube Sampling, or polynomial cubature methods should be employed instead.

*Step 4*: Once the choice probabilities have been calculated, the next step is to construct the AVC matrix for the design. Let $\Omega_N$ denote the AVC matrix given a sample size of *N* respondents (each facing *S* choice situations). This AVC matrix depends in general on the experimental design, *X*, the parameter values, β, and the outcomes of the survey, *Y* = [$y_{jsn}$], where $y_{jsn}$ equals one if respondent *n* chooses alternative *j* in choice situation *s* and is zero otherwise. Since the parameter values β are unknown, prior parameter values $\tilde{\beta}$ are used as best guesses for the true parameters.

The AVC matrix is the negative inverse of the expected Fisher Information matrix (e.g., see Train, 2003), where the latter is equal to the second derivatives of the log-likelihood function:

$$\Omega_N(X,Y,\tilde{\beta}) = -\left[E\left(I_N(X,Y,\beta)\right)\right]^{-1} = -\left[\frac{\partial^2 L_N(X,Y,\tilde{\beta})}{\partial\beta\partial\beta'}\right]^{-1} \tag{7B.1}$$

where $I_N(X, Y, \beta)$ is the Fisher Information matrix with *N* respondents, and $L_N(X, Y, \tilde{\beta})$ is the log-likelihood function in case of *N* respondents defined by

$$L_N(X,Y,\tilde{\beta}) = \sum_{n=1}^{N}\sum_{s=1}^{S}\sum_{j=1}^{J} y_{jsn}\log P_{jsn}(X,\tilde{\beta}). \tag{7B.2}$$

This formulation holds for each model type (MNL, MMNL or EC), only the choice probabilities $P_{jsn}($ $X, \tilde{\beta})$ are different. There are two ways of determining the AVC matrix, either by Monte Carlo simulation, or analytically.

Most researchers have relied on *Monte Carlo simulation*. In this case, a sample of size *N* is generated and parameters are estimated based on simulated choices (by simply computing the observed utilities using some prior parameter estimates, adding random draws for the unobserved utilities, and then determine the chosen alternative by assuming that each respondent selects the alternative with the highest utility). Such an estimation also provides the results for the variance-covariance matrix. This procedure is repeated a large number of times and the average variance-covariance matrix gives the AVC matrix.

Many have not realized that the AVC matrix can be determined analytically, as suggested for MNL models with all generic parameters by McFadden (1974). In this case, the second derivative of the log-likelihood function in Equation (7B.2) is determined and evaluated analytically. A potential problem is, that the vector of outcomes, *Y*, is part of the log-likelihood function, the reason why most researchers perform Monte Carlo simulations. However, it can be shown that the outcomes *Y* drop out when taking the second derivatives in case of the MNL model. This has been shown by McFadden (1974) for models with all generic parameters, and in Rose and Bliemer (2005a) for models with alternative-specific parameters, or a combination. Furthermore, Bliemer *et al*. (2009) have also derived analytical expressions for the second derivatives for the NL model. The outcomes *Y* do not drop out, but as shown Bliemer *et al*. (2009), they can be replaced with probabilities leading to exactly the same AVC matrix, which has been confirmed by Monte Carlo simulation outcomes. Although more tedious, the second derivatives can also be derived for the MMNL model and a similar procedure holds for removing the outcome vector *Y*. Note that the MMNL model will always require some simulations, as the parameters are assumed to be random and therefore expected probabilities need to be approximated using simulation. However, these simulations have no connection with the simulations mentioned earlier for determining the AVC matrix. To conclude, $\Omega_N$ can be determined without knowing the simulated outcomes $Y$, hence, the dependency on $Y$ disappears in Equation (7B.1).

*Step 5*: The next step is to evaluate the statistical efficiency of the design. Efficiency measures have been proposed in the literature in order to calculate an efficiency value based on the AVC matrix, typically expressed as in efficiency 'error' (i.e., a measure for the inefficiency). The objective then becomes to minimize this efficiency error. The most widely used measure is called the *D-error* (not to be confused with the D-efficiency measure of OOD designs (equation (7B.3)), which takes the determinant of the AVC matrix $\Omega_1$, assuming only a single respondent[10]. Other measures exist, such as the *A*-error, which takes the trace (sum of the diagonal elements) of the AVC matrix. However, in contrast to the *D*-error, the *A*-error is sensitive to scaling of the parameters and attributes, hence here only the *D*-error will be discussed.

The D-errors are a function of the experimental design X and the prior values (or prior probability distributions) $\tilde{\beta}$, and can be mathematically formulated as:

$$D_z\text{-error} = \det\left(\Omega_1(X,0)\right)^{1/K}, \qquad (7B.3)$$

$$D_p\text{-error} = \det\left(\Omega_1(X,\tilde{\beta})\right)^{1/K}, \qquad (7B.4)$$

$$D_b\text{-error} = \int_{\tilde{\beta}} \det\left(\Omega_1(X,\tilde{\beta})\right)^{1/K} \phi(\tilde{\beta} \mid \theta)\, d\tilde{\beta}. \qquad (7B.5)$$

where *K* is the number of parameters to be estimated. It is common to normalize the D-error by taking the power 1/*K*. Within the literature, designs which are optimized without any information on the priors (i.e., assuming $\tilde{\beta}$=0) are referred to as $D_z$–optimal designs (Equation (7B.3), whereas designs optimized for specific fixed (non-zero) prior parameters are referred to as $D_p$–optimal designs (Equation (7B.4)). In (Bayesian) $D_b$–optimal designs (Equation (7B.5)), the priors $\tilde{\beta}$ are assumed to be random variables with a joint probability density function *Φ*(.) with given parameters Θ.

*Step 6*: In step 2, we began with a random start design. The next stage in generating efficient choice designs is to change the design(s) and repeat steps 3 to 5 up to *R* number of times, each time recoding the designs relative level of statistical efficiency. By changing the design *R* number of times, the analyst is in effect able to compare the efficiency of each of the *R* different design matrices. It is important to note that for only the smallest of designs will it be possible to search the full enumeration of possible designs[11]. As such, it is common to turn to algorithms to determine as many different designs with low efficiency errors as possible. A number of algorithms have been proposed and implemented within the literature for determining how best to change the attribute levels in locating efficient choice designs. Primarily, these consist of row based and column based algorithms. In a *row based algorithm* choice situations are selected from a predefined candidate set of choice situations (either a full factorial or a fractional factorial) in each iteration. *Column based algorithms* create a design by selecting attribute levels over all choice situations for each attribute. Row based algorithms can easily remove dominated choice situations from the canditure set at the beginning (e.g., by applying some utility balance criterion), but it is more difficult to satisfy attribute level balance. The opposite holds for column based algorithms, in which attribute level balance is easy to satisfy, but finding good combinations of attribute levels in each choice situation is more difficult. In general column based algorithms offer more flexibility and can deal with larger designs, but in some cases (e.g., for unlabelled designs and for specific designs such as constrained designs) row based algorithms are more suitable.

The *Modified Federov algorithm* (Cook and Nachtsheim, 1980) is the most widely used row based algorithm. The algorithm first constructs a candidature set of choice situations which may either be the full factorial (for small problems) or a fractional factorial (for large problems) drawn from the full enumeration of choice situations possible for the problem. Next, a (attribute level balanced) design is created by selecting choice situations from the candidature set, after which the efficiency error (e.g., D-error) is computed for the design. If this design has a lower efficiency error than the current best design, the design is stored as the most efficient design so far, and one continues with the next iteration repeating the whole process again. The algorithm terminates if all possible combinations of choice situations have been evaluated (which is in general not feasible), or after a predefined number of iterations.

*RSC (Relabeling, Swapping & Cycling)* algorithms (Huber and Zwerina, 1996; Sándor and Wedel, 2001) represent the predominant column based algorithms in use today. Each iteration of the algorithm creates different columns for each attribute, which together form a design. This design is evaluated and if it has a lower efficiency error than the current best design, then it is stored. The columns are not created randomly, but are generated in a structured way using *relabeling*, *swapping*, and *cycling* techniques. *Relabeling* involves switching all the attribute levels of an attribute. For example, if the attribute levels 1 and 3 are relabeled, then a column containing the levels (1,2,1,3,2,3) will become (3,2,3,1,2,1). Rather than switch all attribute levels within an attribute, *swapping* involves switching only a few attribute levels within an attribute at a time. For example, if the attribute levels in the first and fourth choice situation are swapped, then (1,2,1,3,2,3) would now become (3,2,1,1,2,3). Finally, *cycling* works by replacing all attribute levels in each choice situation at the same time by replacing the first attribute level with the second level, the second level with the third, etc. Since this impacts all columns, cycling can only be performed if all attributes have exactly the same sets of feasible levels (e.g., in case all variables are dummy coded). Note that it is not necessary to use all three methods simultaneously, such that only relabelling, swapping or cycling, or combinations thereof can be used.

# Chapter 8

# Advanced Features in Generating Efficient Designs

# 8 Advanced Features in Generating Efficient Designs

So far we discussed orthogonal designs which remain the mainstream design type used by practitioners, and efficient designs that have theoretical and practical advantages and are envisaged to be used more and more by practitioners. In this section, several advanced designs will be discussed. These designs are actually special efficient designs in which some of the assumptions are relaxed to allow more flexibility in the design, or in which more constraints are added, both for practical reasons. It is important to note that the designs discussed in this section are the current state-of-the-art and certainly not state-of-the-practice, although practitioners may be highly interested in these advanced designs. We note that there still remains a significant amount of research to be done in this area.

## 8.1 Attribute level balance and fractional factorial designs

As mentioned in Section 6.2.2, most designs in Ngene default to the property of attribute level balance, meaning that for each attribute, each level appears an equal number of times over the choice situations. This will guarantee an even distribution of the levels, such that not just primarily high or low levels are faced by respondents. However, like orthogonality, attribute level balance puts another restriction on the design, such that some efficiency may be lost. Letting go of attribute level balance typically produces more efficient designs, although in practice most people maintain attribute level balance in their design as a desired property.

Ngene allows three methods to overcome the attribute level balance restriction in the types of designs discussed to date, or otherwise control the degree of attribute level balance. We briefly discussed the first method in Section 6.2.2. This involved designs where the number of rows specified is greater than or equal to *K*/(*J*-1), but such that they do not allow for attribute level balance. An example of this is given in the following syntax where the number of rows specified is eight, but attribute A is specified with three levels.

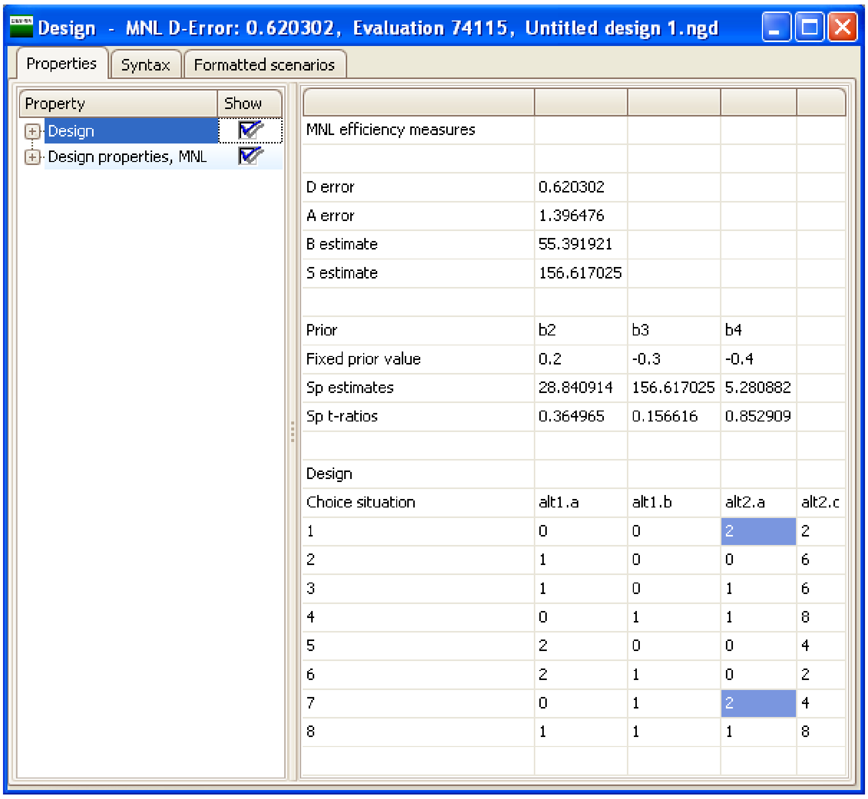
```
Design
;alts = alt1, alt2
;eff=(mnl,d)
;rows = 8
;model:
U(alt1) = b1[-0.2] + b2[0.2] * A[0,1,2] + b3[-0.3] * B[0,1]     /
U(alt2) =            b2      * A        + b4[-0.4] * C[2,4,6,8]
$
```

In this case, Ngene will generate a design but in doing so provide the following warning.

*"Warning: One or more attributes will not have level balance with the number of rows specified: alt1.a, alt2.a"*

In such cases, Ngene will generate a design while attempting to maintain attribute level balance as much as possible. This is shown in Figure 8.1. When the number of attribute levels specified is less than the number of rows, Ngene will ensure that each attribute level appears at least once in the design. If the number of attribute levels for any given attribute exceeds the number of rows specified, it becomes impossible for Ngene to ensure that each attribute level appears at least once over the course of the design, forcing Ngene to select those levels that will maximize the efficiency criteria selected. The following utility functions demonstrate this idea assuming the analyst maintained a desire to generate the design in eight rows.

```
;model:
U(alt1) = b1[-0.2] + b2[0.2] * A[0,1,2,3,4,5,6,7,8] + b3[-0.3] * B[0,1]
/
U(alt2) =           b2      * A           + b4[-0.4] * C[2,4,6,8]
$
```


Design - MNL D-Error: 0.620302, Evaluation 74115, Untitled design 1.ngd

| Property | Show |
|---|---|
| Design | ☑ |
| Design properties, MNL | ☑ |

| | | | | |
|---|---|---|---|---|
| MNL efficiency measures | | | | |
| | | | | |
| D error | 0.620302 | | | |
| A error | 1.396476 | | | |
| B estimate | 55.391921 | | | |
| S estimate | 156.617025 | | | |
| | | | | |
| Prior | b2 | b3 | b4 | |
| Fixed prior value | 0.2 | -0.3 | -0.4 | |
| Sp estimates | 28.840914 | 156.617025 | 5.280882 | |
| Sp t-ratios | 0.364965 | 0.156616 | 0.852909 | |
| | | | | |
| Design | | | | |
| Choice situation | alt1.a | alt1.b | alt2.a | alt2.c |
| 1 | 0 | 0 | 2 | 2 |
| 2 | 1 | 0 | 0 | 6 |
| 3 | 1 | 0 | 1 | 6 |
| 4 | 0 | 1 | 1 | 8 |
| 5 | 2 | 0 | 0 | 4 |
| 6 | 2 | 1 | 0 | 2 |
| 7 | 0 | 1 | 2 | 4 |
| 8 | 1 | 1 | 1 | 8 |

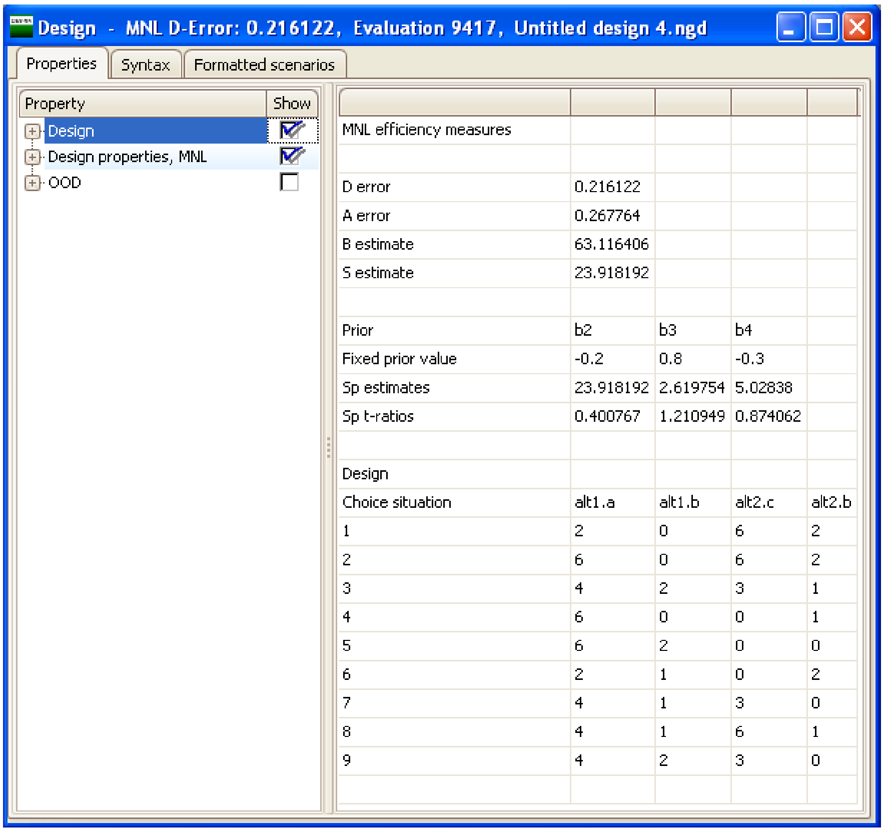
**Figure 8.1: A non attribute level balanced design generated using Method 1**

The second method for letting go of attribute level balance involves the user specifying attribute level count constraints, i.e. how many times each level needs to occur within the design, by indicating a minimum and maximum number. This is done by specifying this minimum and maximum in a range after the attribute levels (using round brackets, and a dash, '-', for indicating a range). This allows more flexibility than the first method of letting go of attribute level balance by allowing the user to specify the number of times an attribute level will appear (within some range) rather than have Ngene attempt to enforce attribute level balance as much as is possible. Example syntax of this method is shown below.

```
Design
;alts = alt1, alt2
;eff = (mnl,d)
;rows = 9
;model:
U(alt1)  = b1[1.1]  + b2[-0.2]  * A[2,4,6](1-4,4,2-4)     + b3[0.8]  * B
[0,1,2] /
U(alt2)  =          b4[-0.3] * C[0,3,6](0-9,2-9,0-9) + b3      * B

$
```

In this example, a design will be generated with nine choice situations, where the levels of attribute 'A' do not necessarily have to be attribute level balanced (i.e., each of the three levels does not have to appear exactly three times). In fact, the first level (2) has to appear 1 to 4 times, the second level (4) exactly 4 times, and the third level (6) has to appear 2 to 4 times. Attribute ‘C’ does not put any restrictions on the number of times each attribute level has to appear, indicated by a minimum of appearing not at all (0) to appearing in all choice situations (9).

Figure 8.2 shows a design generated using the above syntax. Although not always the case, we note that for designs with attributes with more than two levels, if the user allows non-end point levels to appear zero times (i.e., the minimum number of times the middle attribute levels are allowed to appear is set at zero; e.g., 0-9) then typically the most efficient design will be one that will have only the two end point levels. As stated above, this need not be the case, as it depends upon the attribute levels and priors assumed in generating the design, however our experience is that this will be the case in many instances.


Design - MNL D-Error: 0.216122, Evaluation 9417, Untitled design 4.ngd
Properties
Syntax
Formatted scenarios

| Property | Show |
|---|---|
| Design | ☑ |
| Design properties, MNL | ☑ |
| OOD | □ |

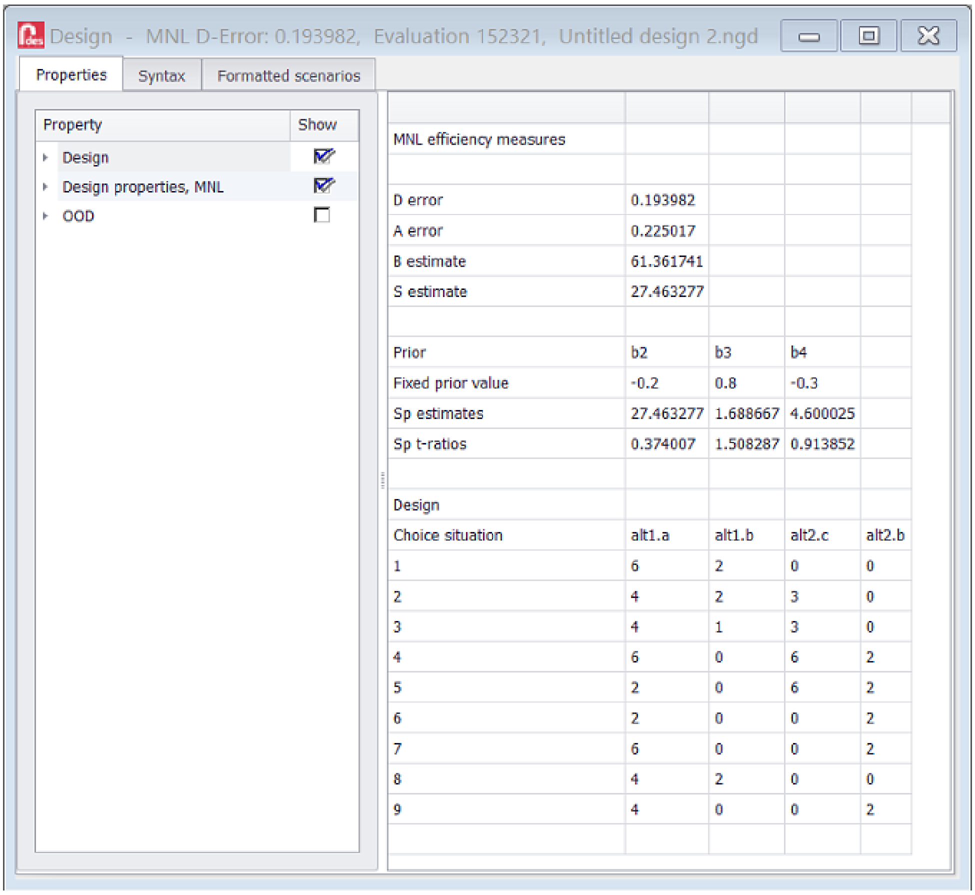
| | | | | |
|---|---|---|---|---|
| MNL efficiency measures | | | | |
| | | | | |
| D error | 0.216122 | | | |
| A error | 0.267764 | | | |
| B estimate | 63.116406 | | | |
| S estimate | 23.918192 | | | |
| | | | | |
| Prior | b2 | b3 | b4 | |
| Fixed prior value | -0.2 | 0.8 | -0.3 | |
| Sp estimates | 23.918192 | 2.619754 | 5.02838 | |
| Sp t-ratios | 0.400767 | 1.210949 | 0.874062 | |
| | | | | |
| Design | | | | |
| Choice situation | alt1.a | alt1.b | alt2.c | alt2.b |
| 1 | 2 | 0 | 6 | 2 |
| 2 | 6 | 0 | 6 | 2 |
| 3 | 4 | 2 | 3 | 1 |
| 4 | 6 | 0 | 0 | 1 |
| 5 | 6 | 2 | 0 | 0 |
| 6 | 2 | 1 | 0 | 2 |
| 7 | 4 | 1 | 3 | 0 |
| 8 | 4 | 1 | 6 | 1 |
| 9 | 4 | 2 | 3 | 0 |

**Figure 8.2: A non attribute level balanced design generated using Method 2**

This second approach, attribute level count constraints, is particularly useful when utilising the Modified Federov algorithm to generate efficient designs (see Section 8.6 for details on the various algorithms available). This is because the Modified Federov algorithm does not preserve attribute level balance by default. By contrast, the swapping algorithm, that is the default in Ngene, will maximise the attribute level balance, so long as attribute level count constraints are not specified. Despite a natural tendency to have attribute level imbalance, the Modified Federov algorithm can be useful in a number of situations, and allows for very flexible constraints to be imposed on the design (see Section 8.2 for more on these constraints). The following is the same syntax as above, but utilising the Modified Federov algorithm.

```
Design
;alts = alt1, alt2
;eff = (mnl,d)
;rows = 9
;alg = mfederov
;model:
U(alt1)  = b1[1.1]  + b2[-0.2]  *  A[2,4,6](1-4,4,2-4)     + b3[0.8]  *  B
[0,1,2] /
U(alt2) =          b4[-0.3] * C[0,3,6](0-9,2-9,0-9) + b3       * B

$
```

Figure 8.3 shows a design generated with the Modified Federov algorithm and attribute level count constraints imposed on some of the attributes. Note how only the end point levels are in the design for attribute 'B'.


Design - MNL D-Error: 0.193982, Evaluation 152321, Untitled design 2.ngd
Properties
Syntax
Formatted scenarios

| Property | Show |
|---|---|
| Design | ☑ |
| Design properties, MNL | ☑ |
| OOD | □ |

| | | | | |
|---|---|---|---|---|
| MNL efficiency measures | | | | |
| | | | | |
| D error | 0.193982 | | | |
| A error | 0.225017 | | | |
| B estimate | 61.361741 | | | |
| S estimate | 27.463277 | | | |
| | | | | |
| Prior | b2 | b3 | b4 | |
| Fixed prior value | -0.2 | 0.8 | -0.3 | |
| Sp estimates | 27.463277 | 1.688667 | 4.600025 | |
| Sp t-ratios | 0.374007 | 1.508287 | 0.913852 | |
| | | | | |
| Design | | | | |
| Choice situation | alt1.a | alt1.b | alt2.c | alt2.b |
| 1 | 6 | 2 | 0 | 0 |
| 2 | 4 | 2 | 3 | 0 |
| 3 | 4 | 1 | 3 | 0 |
| 4 | 6 | 0 | 6 | 2 |
| 5 | 2 | 0 | 6 | 2 |
| 6 | 2 | 0 | 0 | 2 |
| 7 | 6 | 0 | 0 | 2 |
| 8 | 4 | 2 | 0 | 0 |
| 9 | 4 | 0 | 0 | 2 |

**Figure 8.3: A non attribute level balanced design generated with the Modified Federov algorithm using Method 2**

The third method for controlling the amount of attribute level balance is through the imposition of a soft level balance constraint. Unlike the second approach, which imposes hard constraints on how many times each attribute level can appear, this approach calculates an overall measure of attribute level imbalance, which can be added to efficiency measures such as d-error, allowing the design to be optimised on a combination of statistical efficiency and attribute level balance.

Consider an experimental design *x*, which contains *S* choice tasks. Each attribute *k* has $L_k$ levels.

Collins et al. (2014) calculate the attribute level imbalance (LIB) as:

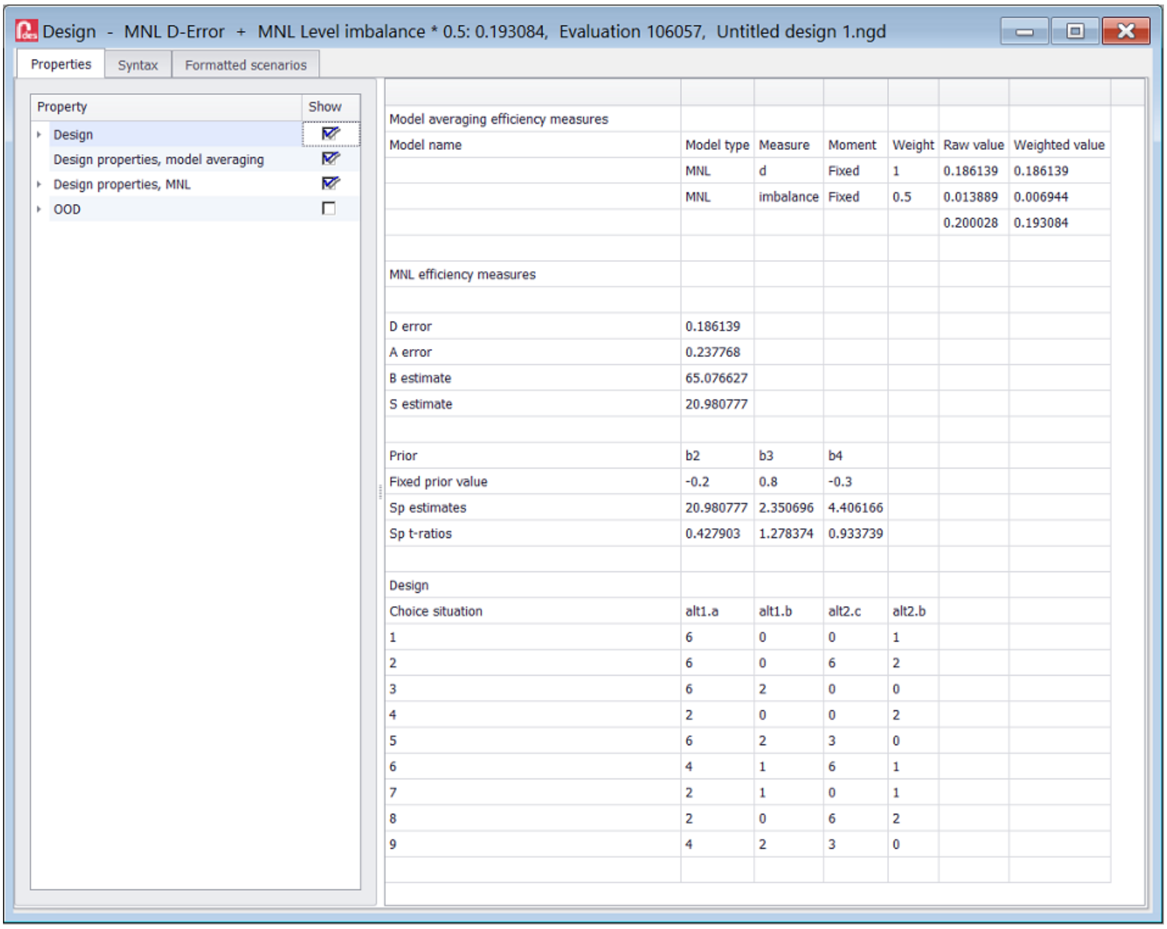
$$LIB = \frac{1}{K}\sum_{k}^{K}\left[\frac{\sum_{l}^{L_k}\left(\frac{S}{L_k}-\sum_{s}^{S}1_{\{x_{ks}=l\}}\right)^2}{\left(\frac{S}{L_k}-S\right)^2+\left(L_k-1\right).\left(\frac{S}{L_k}\right)^2}\right] \tag{8.1}$$

For each attribute, the difference is calculated between the number of times each attribute level occurs, and the number of times it should occur under level balance. Imbalance of greater magnitude is penalised by squaring the difference. The denominator represents the balance measure in the worst case. The final level imbalance measure lies between 0 (full balance) and 1 (full imbalance), and is useful for combining with an efficiency measure in the optimisation function (Collins et al., 2014).

The attribute level imbalance measure can readily be combined with other efficiency measures in Ngene using the *eff* property. The analyst can adjust the relative weights of the imbalance measure and other measures, and may need to do so iteratively to find an appropriate compromise between statistical efficiency and attribute level balance.

```
Design
;alts = alt1, alt2
;eff = (mnl,d) + 0.5*(imbalance)
;rows = 9
;alg = mfederov
;model:
U(alt1) = b1[1.1] + b2[-0.2] * A[2,4,6] + b3[0.8] * B[0,1,2] /
U(alt2) =           b4[-0.3] * C[0,3,6] + b3      * B
$
```

Figure 8.4 shows a design generated with the soft level balance constraint approach specified in the above syntax. Note how level balance is nearly but not fully achieved.


Design - MNL D-Error + MNL Level imbalance * 0.5: 0.193084, Evaluation 106057, Untitled design 1.ngd
Properties
Syntax
Formatted scenarios
Property
Show
Design ☑
Design properties, model averaging ☑
Design properties, MNL ☑
OOD □

| | | | | | | |
|---|---|---|---|---|---|---|
| Model averaging efficiency measures | | | | | | |
| Model name | Model type | Measure | Moment | Weight | Raw value | Weighted value |
| | MNL | d | Fixed | 1 | 0.186139 | 0.186139 |
| | MNL | imbalance | Fixed | 0.5 | 0.013889 | 0.006944 |
| | | | | | 0.200028 | 0.193084 |
| | | | | | | |
| MNL efficiency measures | | | | | | |
| | | | | | | |
| D error | 0.186139 | | | | | |
| A error | 0.237768 | | | | | |
| B estimate | 65.076627 | | | | | |
| S estimate | 20.980777 | | | | | |
| | | | | | | |
| Prior | b2 | b3 | b4 | | | |
| Fixed prior value | -0.2 | 0.8 | -0.3 | | | |
| Sp estimates | 20.980777 | 2.350696 | 4.406166 | | | |
| Sp t-ratios | 0.427903 | 1.278374 | 0.933739 | | | |
| | | | | | | |
| Design | | | | | | |
| Choice situation | alt1.a | alt1.b | alt2.c | alt2.b | | |
| 1 | 6 | 0 | 0 | 1 | | |
| 2 | 6 | 0 | 6 | 2 | | |
| 3 | 6 | 2 | 0 | 0 | | |
| 4 | 2 | 0 | 0 | 2 | | |
| 5 | 6 | 2 | 3 | 0 | | |
| 6 | 4 | 1 | 6 | 1 | | |
| 7 | 2 | 1 | 0 | 1 | | |
| 8 | 2 | 0 | 6 | 2 | | |
| 9 | 4 | 2 | 3 | 0 | | |

**Figure 8.4: A design generated utilising soft level balance constraints**

## 8.2 Constraints and fractional factorial designs

### 8.2.1 Constrained designs

Sometimes certain combinations of attribute levels in a choice situation are not feasible. These infeasible choice situations need to be avoided by adding constraints.

Level constrained designs are most apparent in applications in health economics. For example, consider two alternatives, treating and not treating a patient. Then the attribute ‘age of death’ in these alternatives should be such that in each choice situation this age for the treating alternative is never lower than the non-treating alternative, and the attribute ‘current age’ cannot be higher than the ‘age of death’. In transportation, one could think of route alternatives with different departure times, free-flow travel times, and arrival times. Clearly, the arrival times should be later than the departure times, and the difference between the arrival and departure time should be greater than or equal to the free-flow travel time.

There are different ways of including these constraints. A straightforward way, implemented in Ngene, is using an extended version of the modified Federov algorithm. After having determined the candidate set, choice situations that do not satisfy the constraints are removed from this set. This ensures that all designs generated from this candidate set will be feasible.

Note that it may be hard or even impossible to find an attribute level balanced design satisfying the constraints, especially when the constraints impose many restrictions. Also note that in theory RSC algorithms can also be used, but that after each relabeling, swapping, or cycling all choice situations need to be checked for feasibility. Ensuring that all choice situations are feasible could be difficult, hence RSC algorithms may not be suitable.

### 8.2.2 Constrained designs in Ngene

In order to avoid designs with choice situations that are not feasible, Ngene allows constraints to be put on the attribute levels. Constraints can be included by specifying conditions for the attribute levels. The cond property can be used to include these conditions, which are basically if-then statements.

A first type of constraint is called ‘nesting’. If a specific attribute has a certain level, then another attribute has to have a certain level as well (or perhaps is limited to a set of levels). For example,

```
;cond:
if(alt1.A = 0, alt2.B = 1) ,
if(alt1.A = [1,2], alt2.B = [2,3])
```

Each line contains a condition, and the conditions are separated with a comma, ‘,’. The first condition states that if the attribute level of attribute ‘A’ of alternative ‘alt1’ equals zero, than the attribute level of attribute ‘B’ of alternative ‘alt2’ should be equal to one. The second condition states that if the level of attribute ‘A’ in alternative ‘alt1’ is either one *or* two, than the allowed levels of attribute ‘B’ in alternative ‘alt2’ are two or three. Note that nesting will overrule the attribute levels (and also possible ranges) defined in the model property.

Besides nesting constraints, more general constraints can be included in the *cond* property. Some examples are:

```
;cond:
if(alt1.A + alt1.B > alt1.C, alt2.A = alt1.A) ,
if(alt1.A = alt2.A and alt1.B < 3, alt2.B = [2,3]) ,
if(alt1.A <> alt2.A or alt1.A = 0, alt2.A > 3)
```

Note that the operations ‘=’ (equal to), ‘>’ (greater than), ‘<’ (less than), ‘<=” (less than or equal to), ‘>=” (greater than or equal to), ‘<>’ (not equal to), ‘and’ (logical and), ‘or’ (logical or) can be used in order to make logical expressions. This offers great flexibility in dealing with almost any constraints.

Important to keep in mind is that if the constraints are too strong, Ngene may not be able to find a design that satisfies all constraints. If this happens, in most cases Ngene will report one of a number of error messages. However, if you are having problems generating a design that includes constraints, removing some of these constraints may overcome the problem. It is recommended that you start with an unconstrained design, and progressively add constraints, checking the properties of the design after each addition.

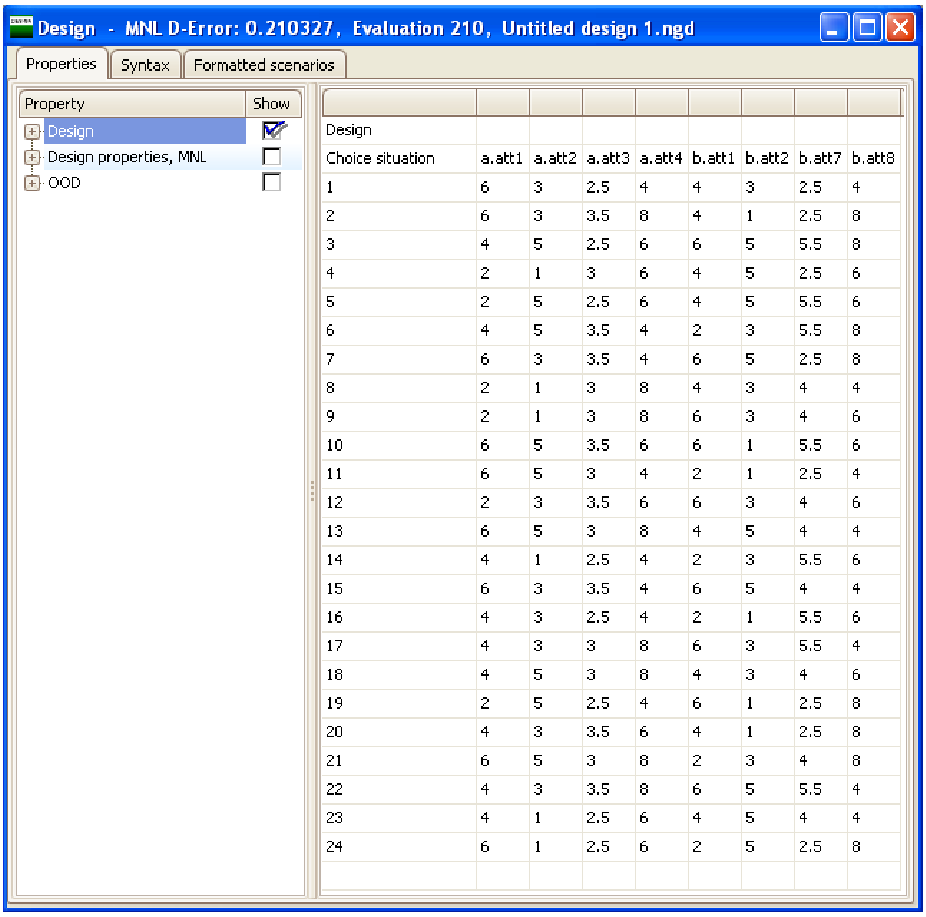
Furthermore, it is very difficult (and often even impossible) to find an attribute level balanced design when constraints are specified, such that Ngene aims to find a design that is as much attribute level balanced as possible. Note that in case of nesting constraints, only the nested attributes will not be attribute level balanced (such as ‘alt2.B’ in the example mentioned above); all other attribute will be. Also note that the *cond* property will only work if a swapping algorithm is used. It will not work for example if a Modified Federov algorithm is applied to the design.

A further complication can arise if a large number of attributes are 'related' through multiple conditions that 'overlap'. Ngene will attempt to generate a full factorial of all combinations of levels from the related attributes that do not violate the conditions. This can lead to memory problems. A warning will be provided if this problem is likely to occur. The solution is to add to the comma separated list in the *cond* property the following: `fractional=X%`. A sufficiently low value of X will solve the memory problem, although several attempts may be required to find a suitable value.

A fully complete example of syntax employing two constraints is shown below.

```
Design
;alts = A, B, C
;rows = 24
;eff = (mnl, d)
;cond:
if(a.att1=2, b.att1=[4,6]),
if(a.att2<3, b.att2=[3,5])
;model:
U(A) = A0[-0.1] + G1[-0.4] * att1[2,4,6]   + G2[-0.3] * att2[1,3,5] + A1
[0.7]  * att3[2.5,3,3.5]  + A2[0.6]  * att4[4,6,8] /
U(B) = B0[-0.2] + G1       * att1         + G2       * att2         + B1
[-0.4] * att7[2.5,4,5.5]  + B2[0.7]  * att8[4,6,8]
$
```

A screenshot of an example design generated using the above syntax is given in Figure 8.5. We leave it to the reader to verify that the conditions specified have actually been meet as well as the degree of attribute level balance of the design shown (a good starting point would be to examine attribute b.att2).



| Design | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Choice situation | a.att1 | a.att2 | a.att3 | a.att4 | b.att1 | b.att2 | b.att7 | b.att8 |
| 1 | 6 | 3 | 2.5 | 4 | 4 | 3 | 2.5 | 4 |
| 2 | 6 | 3 | 3.5 | 8 | 4 | 1 | 2.5 | 8 |
| 3 | 4 | 5 | 2.5 | 6 | 6 | 5 | 5.5 | 8 |
| 4 | 2 | 1 | 3 | 6 | 4 | 5 | 2.5 | 6 |
| 5 | 2 | 5 | 2.5 | 6 | 4 | 5 | 5.5 | 6 |
| 6 | 4 | 5 | 3.5 | 4 | 2 | 3 | 5.5 | 8 |
| 7 | 6 | 3 | 3.5 | 4 | 6 | 5 | 2.5 | 8 |
| 8 | 2 | 1 | 3 | 8 | 4 | 3 | 4 | 4 |
| 9 | 2 | 1 | 3 | 8 | 6 | 3 | 4 | 6 |
| 10 | 6 | 5 | 3.5 | 6 | 6 | 1 | 5.5 | 6 |
| 11 | 6 | 5 | 3 | 4 | 2 | 1 | 2.5 | 4 |
| 12 | 2 | 3 | 3.5 | 6 | 6 | 3 | 4 | 6 |
| 13 | 6 | 5 | 3 | 8 | 4 | 5 | 4 | 4 |
| 14 | 4 | 1 | 2.5 | 4 | 2 | 3 | 5.5 | 6 |
| 15 | 6 | 3 | 3.5 | 4 | 6 | 5 | 4 | 4 |
| 16 | 4 | 3 | 2.5 | 4 | 2 | 1 | 5.5 | 6 |
| 17 | 4 | 3 | 3 | 8 | 6 | 3 | 5.5 | 4 |
| 18 | 4 | 5 | 3 | 8 | 4 | 3 | 4 | 6 |
| 19 | 2 | 5 | 2.5 | 4 | 6 | 1 | 2.5 | 8 |
| 20 | 4 | 3 | 3.5 | 6 | 4 | 1 | 2.5 | 8 |
| 21 | 6 | 5 | 3 | 8 | 2 | 3 | 4 | 8 |
| 22 | 4 | 3 | 3.5 | 8 | 6 | 5 | 5.5 | 4 |
| 23 | 4 | 1 | 2.5 | 6 | 4 | 5 | 4 | 4 |
| 24 | 6 | 1 | 2.5 | 6 | 2 | 5 | 2.5 | 8 |

**Figure 8.5: Example design with constraints**

Similar to the *cond* property, users may use the *require* property to force certain attribute level combinations to be present within the design within all choice situations. The *require* property constructs a set of candidate choice situations that meet some criteria in terms of the attribute level combinations allowed within each choice situation. All other choice situations that do not meet the required choice criteria are then rejected from the design. Note that this method, unlike the *cond* property, cannot be used in conjunction with any form of swapping algorithm but rather requires use of the Modified Federov algorithm or factorial design. Note also that the *require* property will also likely not display the attribute level balance property for the generated design unless the user specifically restricts the number of times each level appears within the design in a manner similar to that discussed in Section 8.1. However, a combination of these restrictions may result in an inability to locate a design, or even if a design can be located, the efficiency level of the design is likely to be poor.

An example of the *require* property is shown below. In this property, the design would require that attribute a.att1 be greater than or equal to that of attribute b.att1 for all choice situations in the design.

```
;require:
a.att1 >= b.att1
```

Note that the operations ‘=’ (equal to), ‘>’ (greater than), ‘<’ (less than), ‘<=” (less than or equal to), ‘>=” (greater than or equal to), ‘<>’ (not equal to), ‘and’ (logical and), ‘or’ (logical or) can be used in order to make logical expressions, as per the *cond* property. Note also that unlike the *cond* property, the require property does not use if statements.

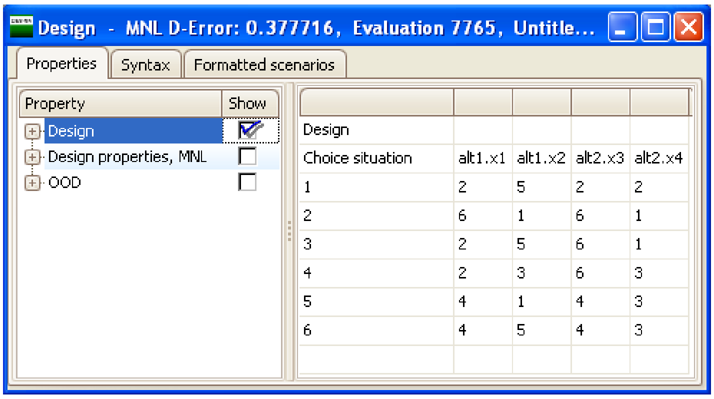
```
Design
;alts = A, B, C
;rows = 24
;eff = (mnl, d)
;alg = mfederov
;require:
a.att1 >= b.att1
;model:
U(A) =  A0[-0.1] + G1[-0.4] * att1[2,4,6]  + G2[-0.3] * att2[1,3,5] + A1
[0.7]  * att3[2,3,4] /
U(B) =  B0[-0.2] + G1       * att1         + G2       * att2         + B1
[-0.4] * att7[3,4,5]
$
```

In addition to using the *cond* and *require* properties, Ngene also allows users to use the *reject* property to force attribute level constraints within a design. Whereas the *cond* and *require* properties force the attributes within the design to meet certain criteria, the reject property disallows a design from having choice situations in which the attributes can take on certain combinations of levels. Unlike the *cond* property, but as with the *require* property, the reject property does not allow the use if statements.

```
;reject:
a.att1 > a.att2
```

Example syntax using the reject property is given below. Figure 8.6 provides a screen capture of a design generated using this syntax, demonstrating that the attribute levels of the design met the required restrictions set.

```
Design
;alts = Alt1, Alt2
;rows = 6
;eff = (mnl, d)
;alg = mfederov
;reject:
Alt1.X1 > Alt2.X3
;model:
U(Alt1)  = b1[-0.2]  + b2[0.3]  *  X1[2,4,6](1-3,1-3,1-3)  + b3[0.4]  *  X2
[1,3,5](1-3,1-3,1-3) /
U(Alt2)  =             b2       *  X3[2,4,6](1-3,1-3,1-3)  + b4[0.3]  *  X4
[1,2,3](1-3,1-3,1-3)
$
```



| Property | Show |
|---|---|
| Design | ☑ |
| Design properties, MNL | □ |
| OOD | □ |

| Design | | | | |
|---|---|---|---|---|
| Choice situation | alt1.x1 | alt1.x2 | alt2.x3 | alt2.x4 |
| 1 | 2 | 5 | 2 | 2 |
| 2 | 6 | 1 | 6 | 1 |
| 3 | 2 | 5 | 6 | 1 |
| 4 | 2 | 3 | 6 | 3 |
| 5 | 4 | 1 | 4 | 3 |
| 6 | 4 | 5 | 4 | 3 |

**Figure 8.6: Example design with constraints using the *reject* property**

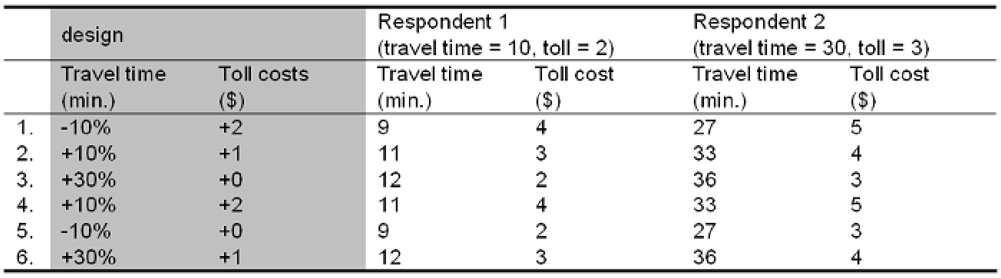
## 8.3 Reference or pivot (customized) designs

### 8.3.1 Pivot designs

So far we have assumed that all respondents face the same choice situations. From a cognitive and contextual point of view, this may not be optimal. The use of a respondent's *knowledge base* to derive the attribute levels of the experiment has come about in recognition of a number of supporting theories in behavioral and cognitive psychology, and economics, such as prospect theory, case-based decisions theory and minimum-regret theory. This leads to the notion of so-called *reference alternatives*, which may be different for each respondent. As Starmer (2000, p. 353) remarks: "While some economists might be tempted to think that questions about how reference points are determined sound more like psychological than economic issues, recent research is showing that understanding the role of reference points may be an important step in explaining real economic behavior in the field." Reference alternatives in stated choice experiments act to frame the decision context of the choice task within some existing memory schema of the individual respondents and hence make preference-revelation more meaningful at the level of the individual.

In a pivot design the attribute levels shown to the respondents are pivoted from reference alternatives of each respondent. In Table 8.1 an example is shown, where for compactness only the first alternative is presented. The actual underlying design is shown in grey, where the attributes are either a relative pivot (as in the travel time), or an absolute pivot (as in the toll cost). The attribute levels shown in the stated choice experiment are based on the reference alternative of the respondents. For example, suppose that respondent 1 has answered in an earlier question in the survey that he or she currently has a travel time of 10 minutes and pays $2 toll, then the attribute levels for the first alternative in the first choice situation will be determined as 10-1 = 9 minutes (10 percent less travel time), and a toll cost of 2+2 = $4 ($2 extra). Therefore, this choice situation will be different from the choice situation presented to respondent 2 (facing a travel time of 27 minutes and a toll of $5 for the first alternative in the first choice situation).

**Table 8.1: Designs pivoted from a reference alternative**



| | design | | Respondent 1<br>(travel time = 10, toll = 2) | | Respondent 2<br>(travel time = 30, toll = 3) | |
|---|---|---|---|---|---|---|
| | Travel time<br>(min.) | Toll costs<br>($) | Travel time<br>(min.) | Toll cost<br>($) | Travel time<br>(min.) | Toll cost<br>($) |
| 1. | -10% | +2 | 9 | 4 | 27 | 5 |
| 2. | +10% | +1 | 11 | 3 | 33 | 4 |
| 3. | +30% | +0 | 12 | 2 | 36 | 3 |
| 4. | +10% | +2 | 11 | 4 | 33 | 5 |
| 5. | -10% | +0 | 9 | 2 | 27 | 3 |
| 6. | +30% | +1 | 12 | 3 | 36 | 4 |

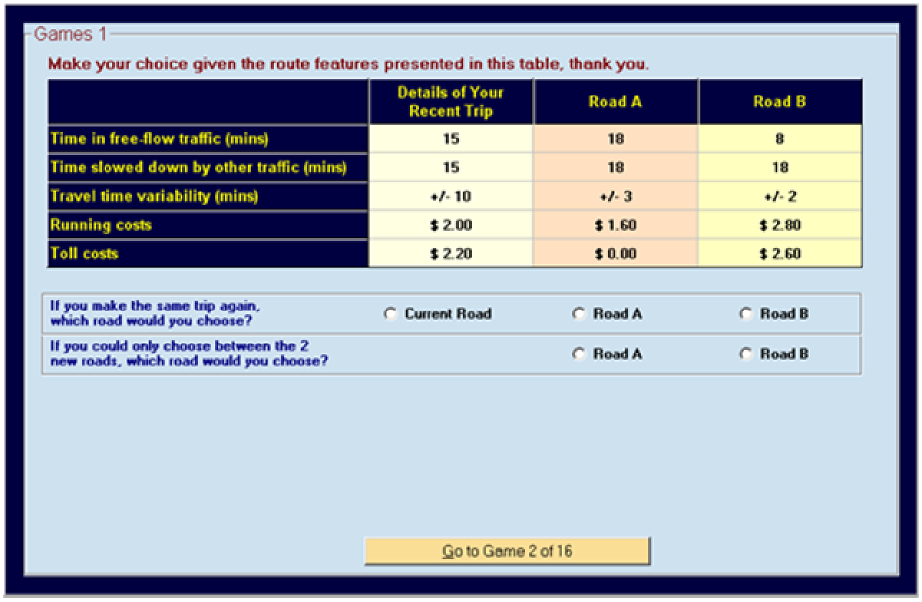
Hence, instead of creating a design with the actual attribute levels, a pivot design is created with relative or absolute deviations from references. Suppose that a single pivot design is created. The efficiency of this design depends on the references of the respondents, as these determine the actual attribute levels in the choice situations and therefore the AVC matrix. However, in advance the references of the respondents are typically not available. Rose *et al*. (2008) have compared several different approaches for finding efficient pivot designs:

(a) Use the population average as the reference (yields a single design);
(b) Segment the population based on a finite set of different references (yields multiple designs);
(c) Determine an efficient design on the fly (yields a separate design for each respondent); and
(d) Use a two-stage process in which the references are captured in the first stage and the design is created in the second stage (yields a single design).

Intuitively, approach (a) should give the lowest efficiency (individual reference alternatives may differ widely from those assumed in generating the design), while the last approach should yield the highest efficiency (likely to produce truly efficient data). This was also the outcome of the study. Approach (a) worked relatively well, and approach (b) only performed marginally better. Approach (c) and (d) performed best. The outcomes were also compared with an orthogonal design, which performed poorly. Pivot designs for approaches (a) and (b) are relatively easy to generate, for approaches (c) and (d) more effort is needed. Approach (c) requires a CAPI or internet survey, and an efficient design is generated while the respondent is answering other questions. Approach (d) is sensitive to drop-outs, as the design will only be optimal if all respondents in the second stage participate again in the survey.

## 8.3.2 Pivot designs in Ngene

Instead of a traditional no-choice alternative, one may want to generate a design with a reference (or status-quo) alternative. Similar to the traditional no-choice alternative, the reference alternative has a fixed utility across choice situations (at least fixed within all choice situations for a single respondent). However, unlike the traditional no-choice alternative, the attribute levels of the alternative need not be absent and hence the utility need not be equal to zero. Figure 8.7 shows an example of a SC questionnaire involving a reference alternative, where the attributes of the first alternative are non-zero and fixed across choice situations.

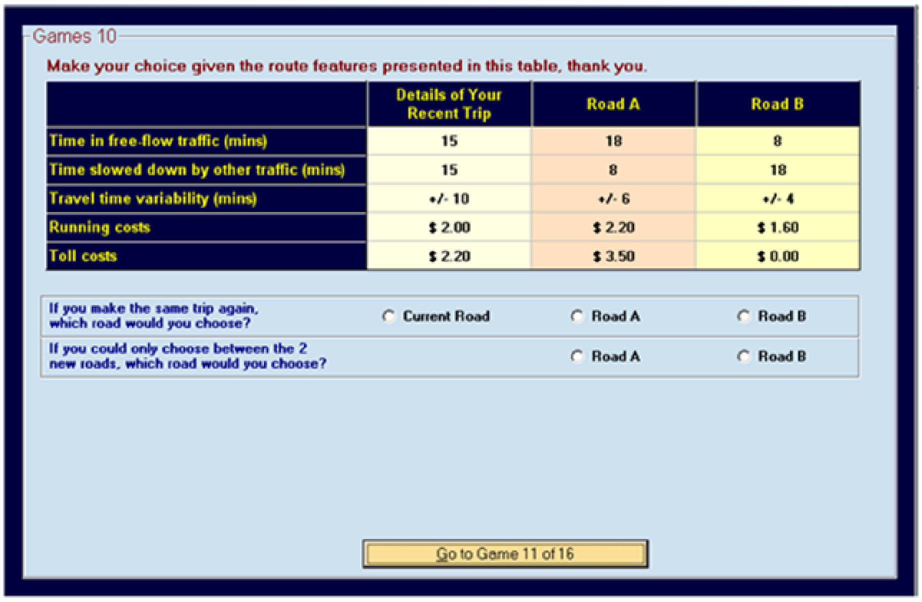


Games 1

Make your choice given the route features presented in this table, thank you.

| | Details of Your Recent Trip | Road A | Road B |
|---|---|---|---|
| Time in free-flow traffic (mins) | 15 | 18 | 8 |
| Time slowed down by other traffic (mins) | 15 | 18 | 18 |
| Travel time variability (mins) | +/- 10 | +/- 3 | +/- 2 |
| Running costs | $ 2.00 | $ 1.60 | $ 2.80 |
| Toll costs | $ 2.20 | $ 0.00 | $ 2.60 |

| | | | |
|---|---|---|---|
| If you make the same trip again, which road would you choose? | ○ Current Road | ○ Road A | ○ Road B |
| If you could only choose between the 2 new roads, which road would you choose? | | ○ Road A | ○ Road B |



Games 10

Make your choice given the route features presented in this table, thank you.

| | Details of Your Recent Trip | Road A | Road B |
|---|---|---|---|
| Time in free-flow traffic (mins) | 15 | 18 | 8 |
| Time slowed down by other traffic (mins) | 15 | 8 | 18 |
| Travel time variability (mins) | +/- 10 | +/- 6 | +/- 4 |
| Running costs | $ 2.00 | $ 2.20 | $ 1.60 |
| Toll costs | $ 2.20 | $ 3.50 | $ 0.00 |

| | | | |
|---|---|---|---|
| If you make the same trip again, which road would you choose? | ○ Current Road | ○ Road A | ○ Road B |
| If you could only choose between the 2 new roads, which road would you choose? | | ○ Road A | ○ Road B |

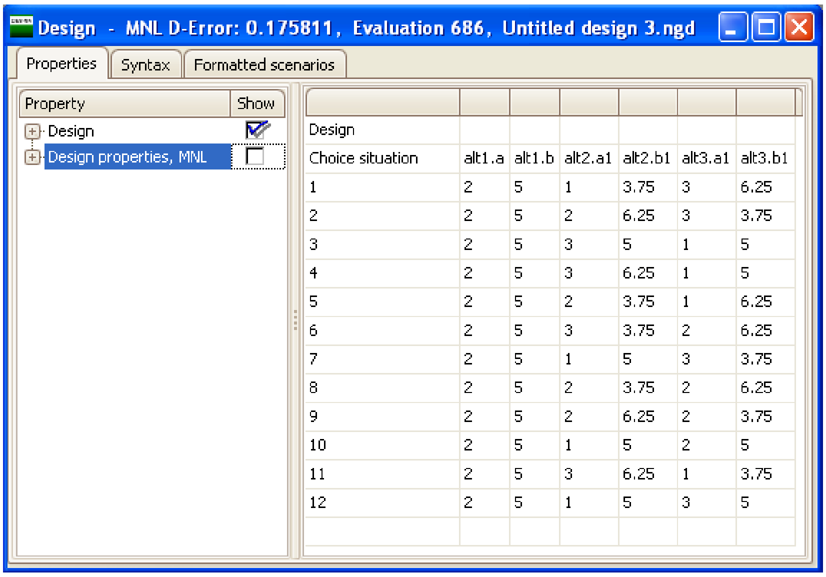
**Figure 8.7: Example choice situations based on a pivot design**

In the simplest case, where all respondents observe the same reference or status quo alternative, Ngene is able to construct pivot style designs quite easily via the utility specifications of the model. For example, the attribute levels of the reference alternative can be assigned a single attribute level rather than multiple levels as in

```
b2[-0.1] * B[5]
```

Next, the attribute levels of the non-reference alternatives can then be chosen so that they vary either by some absolute value from the reference alternative, or by some percentage. For example, if the non-reference alternative levels are to vary by 0% and ±25% from the reference level, then assuming the reference level is 5 (as above), then the levels 3.75, 5 and 6.25 could be assigned to the common attribute of the non-reference alternative. The syntax below demonstrates this concept with the attribute A1 varying by fixed amounts of -1, 0 and 1 around the reference attribute A (= 2) and attribute B1 varying by -25%, 0% and 25% around the reference attribute B (=5).

```
Design
;alts = alt1, alt2, alt3
;rows = 12
;eff = (mnl,d)
;model:
U(alt1) = b1[0.6] * A[2]       + b2[-0.1] * B[5] /
U(alt2) = b1      * A1[1,2,3] + b2 * B1[3.75,5,6.25] /
U(alt3) = b1      * A1[1,2,3] + b2 * B1[3.75,5,6.25]
$
```



Design

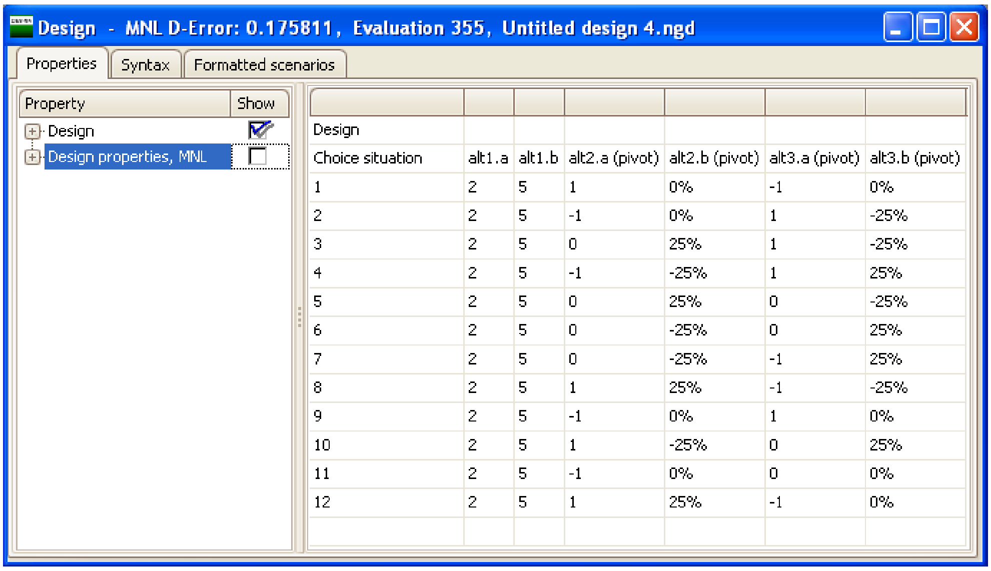
| Choice situation | alt1.a | alt1.b | alt2.a1 | alt2.b1 | alt3.a1 | alt3.b1 |
|---|---|---|---|---|---|---|
| 1 | 2 | 5 | 1 | 3.75 | 3 | 6.25 |
| 2 | 2 | 5 | 2 | 6.25 | 3 | 3.75 |
| 3 | 2 | 5 | 3 | 5 | 1 | 5 |
| 4 | 2 | 5 | 3 | 6.25 | 1 | 5 |
| 5 | 2 | 5 | 2 | 3.75 | 1 | 6.25 |
| 6 | 2 | 5 | 3 | 3.75 | 2 | 6.25 |
| 7 | 2 | 5 | 1 | 5 | 3 | 3.75 |
| 8 | 2 | 5 | 2 | 3.75 | 2 | 6.25 |
| 9 | 2 | 5 | 2 | 6.25 | 2 | 3.75 |
| 10 | 2 | 5 | 1 | 5 | 2 | 5 |
| 11 | 2 | 5 | 3 | 6.25 | 1 | 3.75 |
| 12 | 2 | 5 | 1 | 5 | 3 | 5 |

**Figure 8.8: Example pivot design assuming everyone observes the same reference alternative (method 1)**

Rather than having to calculate the attribute levels of the non-reference alternatives manually (e. g., -25% of 5 is 3.75) and insert these as the attribute levels of the non-reference alternatives, Ngene has available syntax that will automatically do this for you. This first requires the user to specify what attributes represent a reference attribute and which represent those which should be pivoted around the reference attribute. This is handled by adding either the suffix .ref or .piv after an attributes name. For example `B.ref` is used to specify attribute 'B' as a reference alternative, whereas `B.piv` would be used to specify the same attribute (but for another alternative) as an attribute that will be pivoted around the previously specified reference attribute. In specifying the reference alternative, only a single attribute level is required. (e.g., `B.ref[5]`). For the pivoted attribute, the analyst may specify either absolute pivot levels or percentage pivot levels. For absolute pivot levels, the analyst simply places the levels, + or -, that are to be pivoted around the reference attribute (e.g., `B.piv[-2,0,1]`). For pivoted attributes which are to be a percentage change from the reference attribute level, the analyst simply specifies the percentages, + or -, that are required (e.g., `B.piv[-25%,0%,25%]`). Example syntax showing the use of both absolute and percentage change pivot levels is given below.

```
Design
;alts = alt1, alt2, alt3
;rows = 12
;eff = (mnl,d)
;model:
U(alt1) = b1[0.6] * A.ref[2]         + b2[-0.1] * B.ref[5]           /
U(alt2) = b1         * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%] /
U(alt3) = b1         * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%]
$
```

Figure 8.9 provides example output produced using the above syntax. Note that the output differs to that shown previously, in that the actual levels for the non-reference attributes are not given, but rather the absolute or percentage changes. This may be useful where the attribute levels of the reference alternative are not fixed over respondents, but in generating the design a ‘sample average’ is assumed to generate a design that will be applied to all individuals irrespective of their real reference alternative.


Design - MNL D-Error: 0.175811, Evaluation 355, Untitled design 4.ngd

| Design | | | | | | |
|---|---|---|---|---|---|---|
| Choice situation | alt1.a | alt1.b | alt2.a (pivot) | alt2.b (pivot) | alt3.a (pivot) | alt3.b (pivot) |
| 1 | 2 | 5 | 1 | 0% | -1 | 0% |
| 2 | 2 | 5 | -1 | 0% | 1 | -25% |
| 3 | 2 | 5 | 0 | 25% | 1 | -25% |
| 4 | 2 | 5 | -1 | -25% | 1 | 25% |
| 5 | 2 | 5 | 0 | 25% | 0 | -25% |
| 6 | 2 | 5 | 0 | -25% | 0 | 25% |
| 7 | 2 | 5 | 0 | -25% | -1 | 25% |
| 8 | 2 | 5 | 1 | 25% | -1 | -25% |
| 9 | 2 | 5 | -1 | 0% | 1 | 0% |
| 10 | 2 | 5 | 1 | -25% | 0 | 25% |
| 11 | 2 | 5 | -1 | 0% | 0 | 0% |
| 12 | 2 | 5 | 1 | 25% | -1 | 0% |

**Figure 8.9: Example pivot design assuming everyone observes the same reference alternative (method 2)**

The above two methods generate designs assuming all respondents have the same reference alternative in terms of the attribute levels shown. In many cases, different respondents will have reference alternatives with different attribute levels. In Ngene, the analyst is able to generate i) a single design that can be applied to different respondent segments, despite the segments having different attribute levels for their reference alternatives, or ii) different designs for different respondent segments based on the fact the different segments face attributes with different reference attributes. We call the first type of design a ‘homogenous pivot design’ and the second type of design a ‘heterogeneous pivot design’. Both types of designs require additional syntax to generate the required design.

To demonstrate the syntax requirements for these two types of pivot designs, assume that there exist three different respondent segments. To generate both homogenous and heterogeneous pivot designs, syntax for the utility specifications is employed similar that used to generate model averaging designs as described in Section 7.4. That is, separate utility specifications are required for each data segment. For example, assuming three segments, small, medium and large, the following utility specifications might be used. Note that as with the model averaging approach, each segment must be given a unique name, and that the last utility specification for all but the last data segment does not end in a / or $.

```
;model(small):
U(alt1) = b1[0.6] * A.ref[2]      + b2[-0.1] * B.ref[5]           /
U(alt2) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%] /
U(alt3) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%]

;model(Medium):
U(alt1) = b1[0.6] * A.ref[4]      + b2[-0.1] * B.ref[10]          /
U(alt2) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%] /
U(alt3) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%]

;model(Large):
U(alt1) = b1[0.6] * A.ref[6]      + b2[-0.1] * B.ref[15]          /
U(alt2) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%] /
U(alt3) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%]
$
```

Next the user is required to use the *fisher* property to specify i) homogenous or heterogeneous pivot designs is required, and ii) how much weight each data segment should be given in calculating the overall Fisher Information matrix (and hence AVC matrix) of the design. In generating the design, only a single Fisher Information matrix (and hence AVC matrix) is constructed to represent the fact that the data segments are to be combined into a single data set post data collection. If the different segments are to be treated separately in data estimation, then separate designs should be generated as shown at the beginning of this section.

The *fisher* property requires several items of information in order to function properly. Firstly, the user is required to give the Fisher Information matrix a name (n.b., any name can be used). Next, in the case of a homogenous pivot designs, the user is required to provide a name for the design that is to be generated. In the case of heterogeneous pivot design, separate names must be provided for each data segment specific design required. In either design type, the user may use any name to designate the designs. Finally, the analyst is required to specify the weights that each segment is to have in calculating the Fisher Information matrix in generating the designs.

In order to generate a homogenous pivot design, each segment name is associated with a single design, separated by commas. This is done by placing all segment names and attached weights in round brackets after the design name. This is shown below for up to *k* data segments.

```
;fisher(<Fisher Information matrix name>) = <design name>(<model 1 name>
[<model 1 weight>], <model 2 name>[<model 2 weight>], ..., <model k
name>[<model k weight>])
```

In specifying the segment weights, the model names must be those provided in the utility specifications. Also, it is important to note that the weights must sum to one. Thus, given the above system of utility functions, the *fisher* property might look something like

```
;fisher(fish) = design1(small[0.33], medium[0.33], large[0.34])
```

In the above syntax, we have called the Fisher Information matrix 'fish' and the design 'design1'. All designs have been associated with design1 as they are included in the round brackets linked to this design. For the first segment, represented by the utility specifications given in the 'small' model segment, we have assigned the segment a 0.33 weight in calculating the overall Fisher Information matrix. Similarly, we have applied the same weight to the second data segment 'medium'. In order to make the weights sum to one, we have assigned a weight of 0.34 to the last data segment, 'large'.

To construct a heterogeneous pivot design, the different model data segments are linked to designs with different names. Rather than separate the different model data segments with a comma, + signs are used. This is shown below.

```
;fisher(<Fisher Information matrix name>) = <design name>(<model 1 name>[<model 1 weight>]) + <model 2 name>([<model 2 weight>]) + ... + <model k name>([<model k weight>])
```

An example *fisher* property for generating a heterogeneous pivot design for our three segment example is given below.

```
;fisher(Fish) = des1(small[0.33]) + des2(medium[0.33]) + des3(large[0.34])
```

In the above syntax, the small model data segment is linked to a design which we have designated ‘des1’, whereas the medium and large model data segments are linked to different designs, ‘des2’ and ‘des3’ respectively. As such, different designs will be generated for each of the model data segments. As per the homogenous pivot designs, each model data segment must be given a weight in calculating the overall design Fisher Information matrix.

Note that it is also possible to generate designs which both specify that different data segments be generated with both homogenous and heterogeneous pivot designs over different subsets of data segments, as in

```
;fisher(Fish) = des1(small[0.33]) + des2(medium[0.33], large[0.34])
```

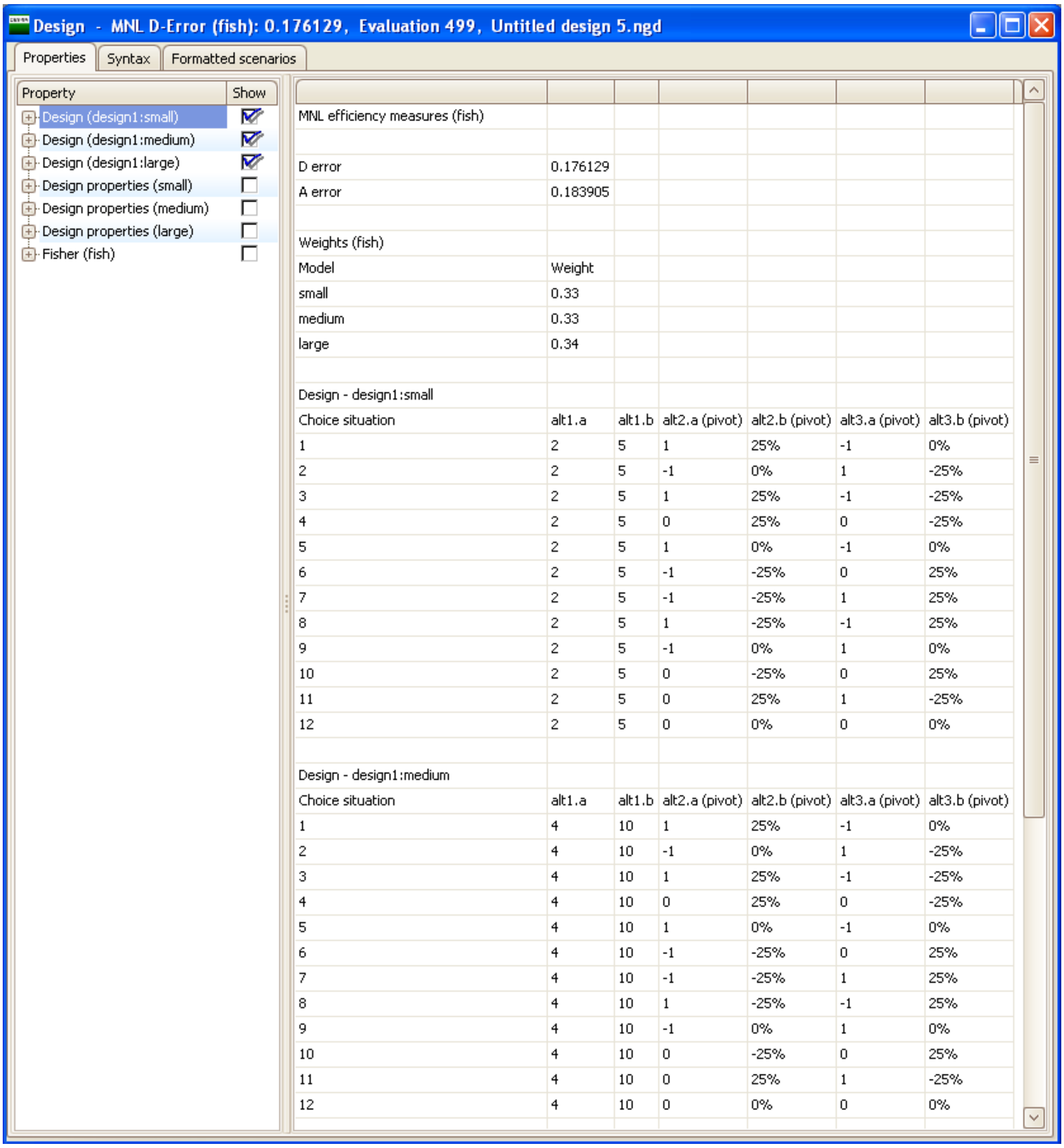
In addition to the *fisher* property, additional syntax is required for the efficiency measure. Rather than optimize on a single Fisher Information matrix (the inverse of the AVC matrix), the design is now to be optimized based on the weighted average Fisher Information matrix named in the fisher property. To handle this, the name of the Fisher Information matrix is added to the *eff* property, much like different models are added to the eff property in the model averaging process. For the above example, the *eff* property would look as follows.

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

Although we show a design specifically generated for an MNL model, the pivot design syntax can be applied to any model type available in Ngene. The complete syntax for a homogeneous pivot design is given below.

```
Design
;alts(small)  = alt1, alt2, alt3
;alts(medium) = alt1, alt2, alt3
;alts(large)  = alt1, alt2, alt3
;rows = 12
;eff = fish(mnl,d)
;fisher(fish) = design1(small[0.33], medium[0.33], large[0.34])

;model(small):
U(alt1) = b1[0.6] * A.ref[2]      + b2[-0.1] * B.ref[5]           /
U(alt2) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%] /
U(alt3) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%]

;model(Medium):
U(alt1) = b1[0.6] * A.ref[4]      + b2[-0.1] * B.ref[10]          /
U(alt2) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%] /
U(alt3) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%]

;model(Large):
U(alt1) = b1[0.6] * A.ref[6]      + b2[-0.1] * B.ref[15]          /
U(alt2) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%] /
U(alt3) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%]
$
```

Figure 8.10 shows output generated for the above syntax. From the output screen, it can be clearly seen that despite the reference alternatives taking on different attribute levels, the design itself has been constrained to be the same across each segment. Although not shown, the analyst is also able to examine the design properties as related to each data segment by clicking on the relevant click boxes located on the left of the output screen.



Design - MNL D-Error (fish): 0.176129, Evaluation 499, Untitled design 5.ngd

| Property | Show |
|---|---|
| Design (design1:small) | ☑ |
| Design (design1:medium) | ☑ |
| Design (design1:large) | ☑ |
| Design properties (small) | □ |
| Design properties (medium) | □ |
| Design properties (large) | □ |
| Fisher (fish) | □ |

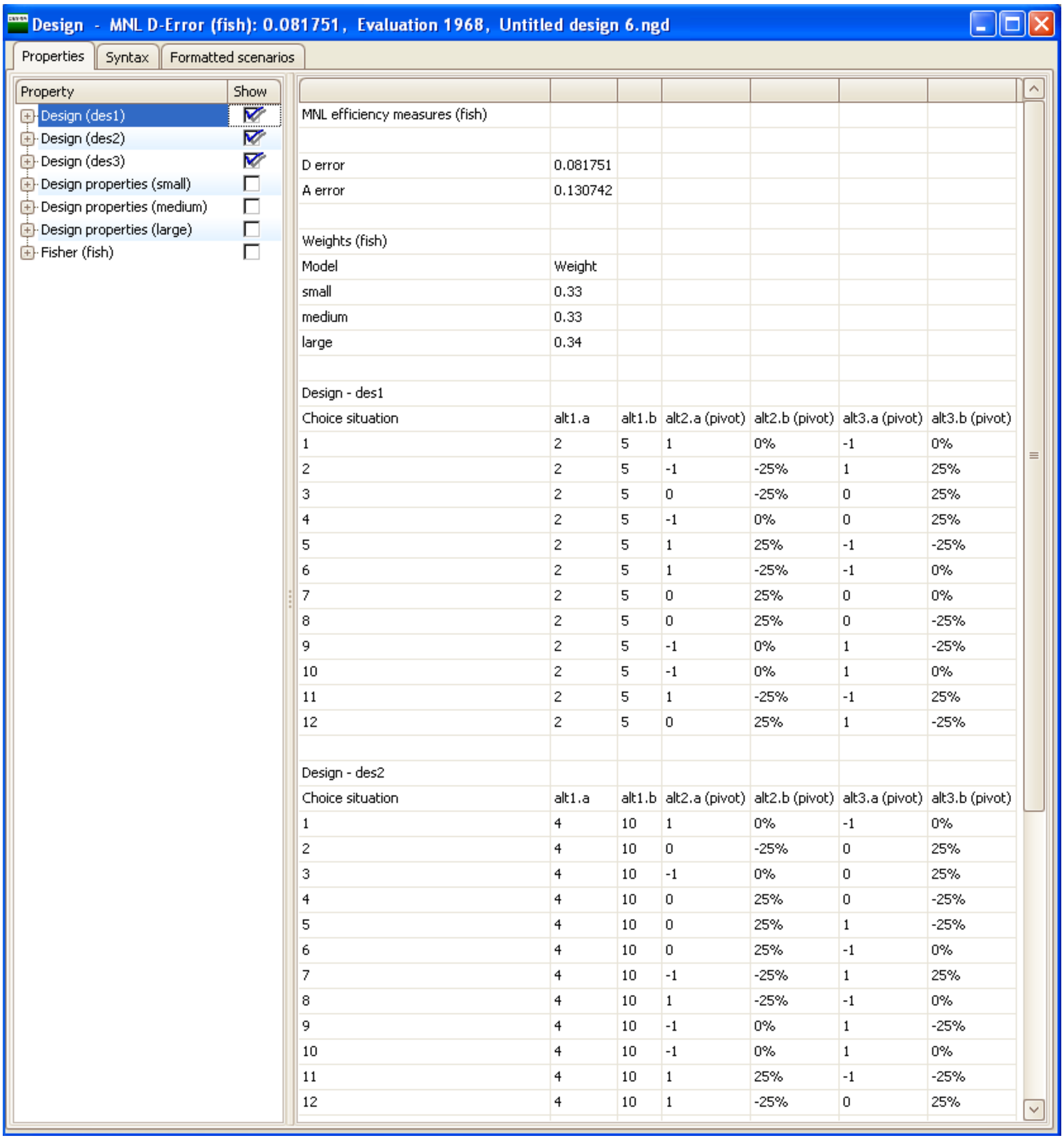
| | | | | | | |
|---|---|---|---|---|---|---|
| MNL efficiency measures (fish) | | | | | | |
| | | | | | | |
| D error | 0.176129 | | | | | |
| A error | 0.183905 | | | | | |
| | | | | | | |
| Weights (fish) | | | | | | |
| Model | Weight | | | | | |
| small | 0.33 | | | | | |
| medium | 0.33 | | | | | |
| large | 0.34 | | | | | |
| | | | | | | |
| Design - design1:small | | | | | | |
| Choice situation | alt1.a | alt1.b | alt2.a (pivot) | alt2.b (pivot) | alt3.a (pivot) | alt3.b (pivot) |
| 1 | 2 | 5 | 1 | 25% | -1 | 0% |
| 2 | 2 | 5 | -1 | 0% | 1 | -25% |
| 3 | 2 | 5 | 1 | 25% | -1 | -25% |
| 4 | 2 | 5 | 0 | 25% | 0 | -25% |
| 5 | 2 | 5 | 1 | 0% | -1 | 0% |
| 6 | 2 | 5 | -1 | -25% | 0 | 25% |
| 7 | 2 | 5 | -1 | -25% | 1 | 25% |
| 8 | 2 | 5 | 1 | -25% | -1 | 25% |
| 9 | 2 | 5 | -1 | 0% | 1 | 0% |
| 10 | 2 | 5 | 0 | -25% | 0 | 25% |
| 11 | 2 | 5 | 0 | 25% | 1 | -25% |
| 12 | 2 | 5 | 0 | 0% | 0 | 0% |
| | | | | | | |
| Design - design1:medium | | | | | | |
| Choice situation | alt1.a | alt1.b | alt2.a (pivot) | alt2.b (pivot) | alt3.a (pivot) | alt3.b (pivot) |
| 1 | 4 | 10 | 1 | 25% | -1 | 0% |
| 2 | 4 | 10 | -1 | 0% | 1 | -25% |
| 3 | 4 | 10 | 1 | 25% | -1 | -25% |
| 4 | 4 | 10 | 0 | 25% | 0 | -25% |
| 5 | 4 | 10 | 1 | 0% | -1 | 0% |
| 6 | 4 | 10 | -1 | -25% | 0 | 25% |
| 7 | 4 | 10 | -1 | -25% | 1 | 25% |
| 8 | 4 | 10 | 1 | -25% | -1 | 25% |
| 9 | 4 | 10 | -1 | 0% | 1 | 0% |
| 10 | 4 | 10 | 0 | -25% | 0 | 25% |
| 11 | 4 | 10 | 0 | 25% | 1 | -25% |
| 12 | 4 | 10 | 0 | 0% | 0 | 0% |

**Figure 8.10: Homogeneous pivot design output screen**

For the same data segments, the following syntax will generate a heterogeneous design.

```
Design
;alts(small)  = alt1, alt2, alt3
;alts(medium) = alt1, alt2, alt3
;alts(large)  = alt1, alt2, alt3
;rows = 12
;eff = fish(mnl,d)
;fisher(Fish)  =  des1(small[0.33])  +  des2(medium[0.33])  +  des3(large
[0.34])

;model(small):
U(alt1) = b1[0.6] * A.ref[2]      + b2[-0.1] * B.ref[5]            /
U(alt2) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%] /
U(alt3) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%]

;model(Medium):
U(alt1) = b1[0.6] * A.ref[4]      + b2[-0.1] * B.ref[10]           /
U(alt2) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%] /
U(alt3) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%]

;model(Large):
U(alt1) = b1[0.6] * A.ref[6]      + b2[-0.1] * B.ref[15]           /
U(alt2) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%] /
U(alt3) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%]
$
```

Output generated using the above syntax is shown in Figure 8.11. Examination of the output demonstrates that the non-reference alternatives are indeed different across the different data segments, meaning that each data segment has its own unique design.



Design - MNL D-Error (fish): 0.081751, Evaluation 1968, Untitled design 6.ngd

| Property | Show |
|---|---|
| Design (des1) | ☑ |
| Design (des2) | ☑ |
| Design (des3) | ☑ |
| Design properties (small) | □ |
| Design properties (medium) | □ |
| Design properties (large) | □ |
| Fisher (fish) | □ |

| | | | | | | |
|---|---|---|---|---|---|---|
| MNL efficiency measures (fish) | | | | | | |
| | | | | | | |
| D error | 0.081751 | | | | | |
| A error | 0.130742 | | | | | |
| | | | | | | |
| Weights (fish) | | | | | | |
| Model | Weight | | | | | |
| small | 0.33 | | | | | |
| medium | 0.33 | | | | | |
| large | 0.34 | | | | | |
| | | | | | | |
| Design - des1 | | | | | | |
| Choice situation | alt1.a | alt1.b | alt2.a (pivot) | alt2.b (pivot) | alt3.a (pivot) | alt3.b (pivot) |
| 1 | 2 | 5 | 1 | 0% | -1 | 0% |
| 2 | 2 | 5 | -1 | -25% | 1 | 25% |
| 3 | 2 | 5 | 0 | -25% | 0 | 25% |
| 4 | 2 | 5 | -1 | 0% | 0 | 25% |
| 5 | 2 | 5 | 1 | 25% | -1 | -25% |
| 6 | 2 | 5 | 1 | -25% | -1 | 0% |
| 7 | 2 | 5 | 0 | 25% | 0 | 0% |
| 8 | 2 | 5 | 0 | 25% | 0 | -25% |
| 9 | 2 | 5 | -1 | 0% | 1 | -25% |
| 10 | 2 | 5 | -1 | 0% | 1 | 0% |
| 11 | 2 | 5 | 1 | -25% | -1 | 25% |
| 12 | 2 | 5 | 0 | 25% | 1 | -25% |
| | | | | | | |
| Design - des2 | | | | | | |
| Choice situation | alt1.a | alt1.b | alt2.a (pivot) | alt2.b (pivot) | alt3.a (pivot) | alt3.b (pivot) |
| 1 | 4 | 10 | 1 | 0% | -1 | 0% |
| 2 | 4 | 10 | 0 | -25% | 0 | 25% |
| 3 | 4 | 10 | -1 | 0% | 0 | 25% |
| 4 | 4 | 10 | 0 | 25% | 0 | -25% |
| 5 | 4 | 10 | 0 | 25% | 1 | -25% |
| 6 | 4 | 10 | 0 | 25% | -1 | 0% |
| 7 | 4 | 10 | -1 | -25% | 1 | 25% |
| 8 | 4 | 10 | 1 | -25% | -1 | 0% |
| 9 | 4 | 10 | -1 | 0% | 1 | -25% |
| 10 | 4 | 10 | -1 | 0% | 1 | 0% |
| 11 | 4 | 10 | 1 | 25% | -1 | -25% |
| 12 | 4 | 10 | 1 | -25% | 0 | 25% |

**Figure 8.11: Heterogeneous pivot design output screen**

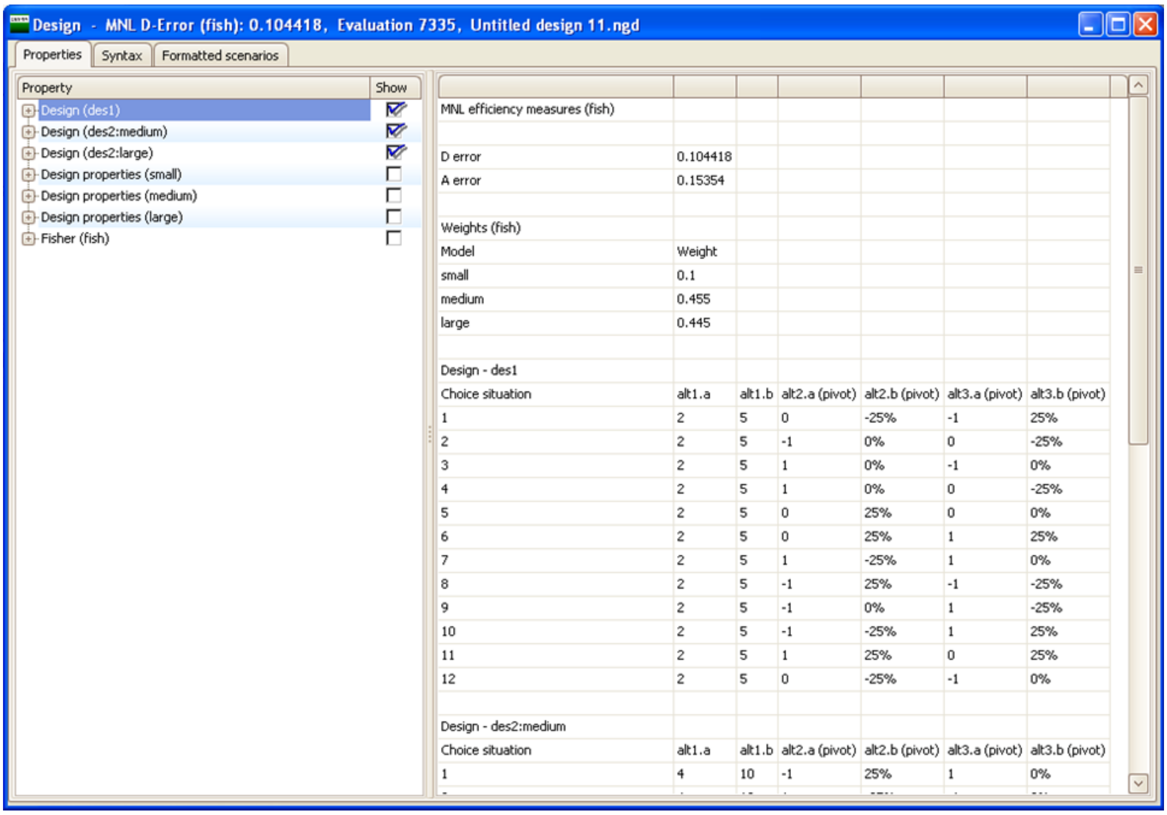
The optimization process for generating pivot designs requires that each data segment be given a weight in calculating the overall design Fisher Information matrix. The previous syntax assumed that the analyst knew *a priori* the proportions that each segment will appear within the sample. It is possible however to not only optimize the efficiency of a design, but also simultaneously the proportions of various segments that should be exposed to the design. As described in Rose and Bliemer (2006), the optimization routine first generates a random design (using whatever algorithm), and then searches over different segment proportions to determine if the overall efficiency level can be improved. If the efficiency level cannot be improved beyond the current best level, then another design is then examined.

This is handled in the weighting section of the *fisher* property. Rather than assign a single weight to each data segment, the analyst may specify a range of weights. This is done by separating a lower weight bound from an upper weight bound by a colon. For example, the syntax `small`

[0.1:0.6] will allow the small data segment to have a weight anywhere between 0.1 and 0.6 in the optimization of the overall design Fisher Information matrix. Note that for this to work, the upper weight bounds provided must sum to one or more. Note also, that this function may be applied to both homogenous and heterogeneous pivot designs. Example syntax for this is given below.

```
Design
;alts(small)  = alt1, alt2, alt3
;alts(medium) = alt1, alt2, alt3
;alts(large)  = alt1, alt2, alt3

;rows = 12
;eff = Fish(mnl,d)
;fisher(Fish)  =  des1(small[0.1:0.6])  +  des2(medium[0.1:0.6],  large
[0.1:0.6])
;model(small):
U(alt1) = b1[0.6] * A.ref[2]      + b2[-0.1] * B.ref[5]            /
U(alt2) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%] /
U(alt3) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%]

;model(Medium):
U(alt1) = b1[0.6] * A.ref[4]      + b2[-0.1] * B.ref[10]           /
U(alt2) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%] /
U(alt3) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%]

;model(Large):
U(alt1) = b1[0.6] * A.ref[6]      + b2[-0.1] * B.ref[15]           /
U(alt2) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%] /
U(alt3) = b1      * A.piv[-1,0,1] + b2[-0.2] * B.piv[-25%,0%,25%]
$
```

Figure 8.12 shows design generated using the above syntax. Based on this design, the output suggests that the most efficient results will be obtained if the first segment is represented by 10% of the final sample, whereas the remaining two segments should make up 45.5% and 44.5% of the final sample. In this way, if there exists only enough budget to collect data from 200 respondents, then 20 respondents should be sampled from segment 1 (i.e., the ‘small’ segment), 91 from segment 2 (i.e., the ‘medium’ segment), and the remaining 89 from segment 3 (i.e., the ‘large’ segment).


Design - MNL D-Error (fish): 0.104418, Evaluation 7335, Untitled design 11.ngd
Properties
Syntax
Formatted scenarios

| Property | Show |
|---|---|
| Design (des1) | ☑ |
| Design (des2:medium) | ☑ |
| Design (des2:large) | ☑ |
| Design properties (small) | □ |
| Design properties (medium) | □ |
| Design properties (large) | □ |
| Fisher (fish) | □ |

| | | | | | | |
|---|---|---|---|---|---|---|
| MNL efficiency measures (fish) | | | | | | |
| | | | | | | |
| D error | 0.104418 | | | | | |
| A error | 0.15354 | | | | | |
| | | | | | | |
| Weights (fish) | | | | | | |
| Model | Weight | | | | | |
| small | 0.1 | | | | | |
| medium | 0.455 | | | | | |
| large | 0.445 | | | | | |
| | | | | | | |
| Design - des1 | | | | | | |
| Choice situation | alt1.a | alt1.b | alt2.a (pivot) | alt2.b (pivot) | alt3.a (pivot) | alt3.b (pivot) |
| 1 | 2 | 5 | 0 | -25% | -1 | 25% |
| 2 | 2 | 5 | -1 | 0% | 0 | -25% |
| 3 | 2 | 5 | 1 | 0% | -1 | 0% |
| 4 | 2 | 5 | 1 | 0% | 0 | -25% |
| 5 | 2 | 5 | 0 | 25% | 0 | 0% |
| 6 | 2 | 5 | 0 | 25% | 1 | 25% |
| 7 | 2 | 5 | 1 | -25% | 1 | 0% |
| 8 | 2 | 5 | -1 | 25% | -1 | -25% |
| 9 | 2 | 5 | -1 | 0% | 1 | -25% |
| 10 | 2 | 5 | -1 | -25% | 1 | 25% |
| 11 | 2 | 5 | 1 | 25% | 0 | 25% |
| 12 | 2 | 5 | 0 | -25% | -1 | 0% |
| | | | | | | |
| Design - des2:medium | | | | | | |
| Choice situation | alt1.a | alt1.b | alt2.a (pivot) | alt2.b (pivot) | alt3.a (pivot) | alt3.b (pivot) |
| 1 | 4 | 10 | -1 | 25% | 1 | 0% |

**Figure 8.12: Heterogeneous pivot design output screen**

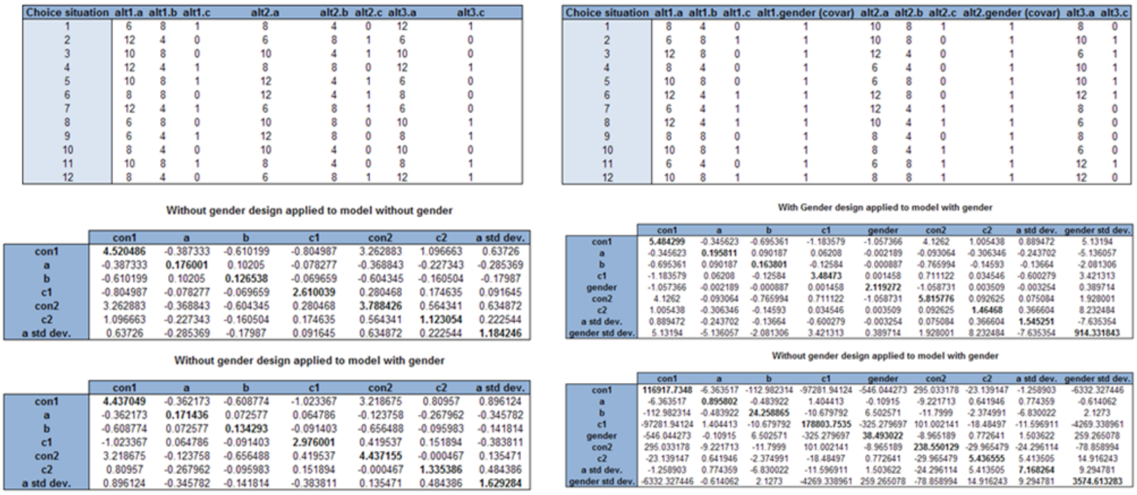
# 8.4 Including covariates in generating efficient designs

## 8.4.1 Designs with covariates

Including covariates (e.g., socio-economic data such as income, gender, car-ownership, etc.) in the model estimation may result in loss of efficiency when the design was generated ignoring these covariates. So far, only attributes have been considered in the model specification, but it is common to include covariates in the estimation process. Analysts should primarily be interested in the efficiency of the SC data collected rather than being concerned about the efficiency of the underlying SC design. Designs should be constructing in a manner that will reflect the final data to be collected, including any possible covariates.

Rose and Bliemer (2006) demonstrate how efficient SC experiments may be constructed to account for covariates, and how minimum quotas may be established in order to retain a fixed level of efficiency. The procedures for doing this are not much different for constructing efficient designs without considering any covariates. Assuming categorical covariates (or continuous covariates coded categorically), it is possible to calculate the AVC matrix for a SC study by constructing a set of segments based on combinations of covariates, and assigning to each segment one or more SC designs. If multiple covariates are to be analyzed, the analyst may wish to construct a full factorial or fractional factorial of the possible combinations formed by the covariates and assign to each the generated design. Next the analyst may generate segment specific efficient designs that minimize the AVC matrix for the pooled data. Procedures similar to those discussed here may be used to do this, however, rather than having one design, the analyst now has to deal with multiple ‘stacked’ or pooled designs.

Figure 8.13 shows two different designs; the one on the left generated without a gender covariate and the one on the right with. Below each of the design are the AVC matrices for the two designs assuming that gender either is or is not included in the model utility function during the estimation process. Examination of the different AVC matrices highlights the fact that an efficient design generated without accounting for possible covariates may potentially lose efficiency when the covariate is included in the estimation process. This is because any covariate will impact upon the choice probabilities of the design and hence will impact upon the elements contained within the AVC matrix.



| Choice situation | alt1.a | alt1.b | alt1.c | alt2.a | alt2.b | alt2.c | alt3.a | alt3.c |
|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 8 | 1 | 8 | 4 | 0 | 12 | 1 |
| 2 | 12 | 4 | 0 | 6 | 8 | 1 | 6 | 0 |
| 3 | 10 | 8 | 0 | 10 | 4 | 1 | 10 | 0 |
| 4 | 12 | 4 | 1 | 8 | 8 | 0 | 12 | 1 |
| 5 | 10 | 8 | 1 | 12 | 4 | 1 | 6 | 0 |
| 6 | 8 | 8 | 0 | 12 | 4 | 1 | 8 | 0 |
| 7 | 12 | 4 | 1 | 6 | 8 | 1 | 6 | 0 |
| 8 | 6 | 8 | 0 | 10 | 8 | 0 | 10 | 1 |
| 9 | 6 | 4 | 1 | 12 | 8 | 0 | 8 | 1 |
| 10 | 8 | 4 | 0 | 10 | 4 | 0 | 10 | 0 |
| 11 | 10 | 8 | 1 | 8 | 4 | 0 | 8 | 1 |
| 12 | 8 | 4 | 0 | 6 | 8 | 1 | 12 | 1 |

| Choice situation | alt1.a | alt1.b | alt1.c | alt1.gender (covar) | alt2.a | alt2.b | alt2.c | alt2.gender (covar) | alt3.a | alt3.c |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 8 | 4 | 0 | 1 | 10 | 8 | 1 | 1 | 8 | 0 |
| 2 | 6 | 8 | 1 | 1 | 10 | 8 | 0 | 1 | 10 | 1 |
| 3 | 12 | 8 | 0 | 1 | 12 | 4 | 0 | 1 | 6 | 1 |
| 4 | 8 | 4 | 0 | 1 | 6 | 4 | 0 | 1 | 10 | 1 |
| 5 | 10 | 8 | 0 | 1 | 6 | 8 | 0 | 1 | 10 | 1 |
| 6 | 12 | 4 | 1 | 1 | 12 | 8 | 0 | 1 | 12 | 1 |
| 7 | 6 | 4 | 1 | 1 | 12 | 4 | 1 | 1 | 8 | 0 |
| 8 | 12 | 4 | 1 | 1 | 10 | 4 | 1 | 1 | 6 | 0 |
| 9 | 8 | 8 | 0 | 1 | 8 | 4 | 0 | 1 | 8 | 0 |
| 10 | 10 | 8 | 1 | 1 | 8 | 4 | 1 | 1 | 6 | 0 |
| 11 | 6 | 4 | 0 | 1 | 6 | 8 | 1 | 1 | 12 | 1 |
| 12 | 10 | 8 | 1 | 1 | 8 | 8 | 1 | 1 | 12 | 0 |

Without gender design applied to model without gender

| | con1 | a | b | c1 | con2 | c2 | a std dev. |
|---|---|---|---|---|---|---|---|
| con1 | **4.520486** | -0.387333 | -0.610199 | -0.804987 | 3.262883 | 1.096663 | 0.63726 |
| a | -0.387333 | **0.176001** | 0.10205 | -0.078277 | -0.368843 | -0.227343 | -0.285369 |
| b | -0.610199 | 0.10205 | **0.126538** | -0.069659 | -0.604345 | -0.160504 | -0.17987 |
| c1 | -0.804987 | -0.078277 | -0.069659 | **2.610039** | 0.280468 | 0.174635 | 0.091645 |
| con2 | 3.262883 | -0.368843 | -0.604345 | 0.280468 | **3.788426** | 0.564341 | 0.634872 |
| c2 | 1.096663 | -0.227343 | -0.160504 | 0.174635 | 0.564341 | **1.123054** | 0.222544 |
| a std dev. | 0.63726 | -0.285369 | -0.17987 | 0.091645 | 0.634872 | 0.222544 | **1.184246** |

With Gender design applied to model with gender

| | con1 | a | b | c1 | gender | con2 | c2 | a std dev. | gender std dev. |
|---|---|---|---|---|---|---|---|---|---|
| con1 | **5.484299** | -0.345623 | -0.695361 | -1.183579 | -1.057366 | 4.1262 | 1.005438 | 0.889472 | 5.13194 |
| a | -0.345623 | **0.195811** | 0.090187 | 0.06208 | -0.002189 | -0.093064 | -0.306346 | -0.243702 | -5.136057 |
| b | -0.695361 | 0.090187 | **0.163801** | -0.12584 | -0.000887 | -0.765994 | -0.14593 | -0.13664 | -2.081306 |
| c1 | -1.183579 | 0.06208 | -0.12584 | **3.48473** | 0.001458 | 0.711122 | 0.034546 | -0.600279 | 3.421313 |
| gender | -1.057366 | -0.002189 | -0.000887 | 0.001458 | **2.119272** | -1.058731 | 0.003509 | -0.003254 | 0.389714 |
| con2 | 4.1262 | -0.093064 | -0.765994 | 0.711122 | -1.058731 | **5.815776** | 0.092625 | 0.075084 | 1.928001 |
| c2 | 1.005438 | -0.306346 | -0.14593 | 0.034546 | 0.003509 | 0.092625 | **1.46468** | 0.366604 | 8.232484 |
| a std dev. | 0.889472 | -0.243702 | -0.13664 | -0.600279 | -0.003254 | 0.075084 | 0.366604 | **1.545251** | -7.635354 |
| gender std dev. | 5.13194 | -5.136057 | -2.081306 | 3.421313 | 0.389714 | 1.928001 | 8.232484 | -7.635354 | **914.331843** |

Without gender design applied to model with gender

| | con1 | a | b | c1 | con2 | c2 | a std dev. |
|---|---|---|---|---|---|---|---|
| con1 | **4.437049** | -0.362173 | -0.608774 | -1.023367 | 3.218675 | 0.80957 | 0.896124 |
| a | -0.362173 | **0.171436** | 0.072577 | 0.064786 | -0.123758 | -0.267962 | -0.345782 |
| b | -0.608774 | 0.072577 | **0.134293** | -0.091403 | -0.656488 | -0.095983 | -0.141814 |
| c1 | -1.023367 | 0.064786 | -0.091403 | **2.976001** | 0.419537 | 0.151894 | -0.383811 |
| con2 | 3.218675 | -0.123758 | -0.656488 | 0.419537 | **4.437155** | -0.000467 | 0.135471 |
| c2 | 0.80957 | -0.267962 | -0.095983 | 0.151894 | -0.000467 | **1.335386** | 0.484386 |
| a std dev. | 0.896124 | -0.345782 | -0.141814 | -0.383811 | 0.135471 | 0.484386 | **1.629284** |

Without gender design applied to model with gender

| | con1 | a | b | c1 | gender | con2 | c2 | a std dev. | gender std dev. |
|---|---|---|---|---|---|---|---|---|---|
| con1 | **116917.7348** | -6.363517 | -112.982314 | -97281.94124 | -546.044273 | 295.033178 | -23.139147 | -1.258903 | -6332.327446 |
| a | -6.363517 | **0.895802** | -0.483922 | 1.404413 | -0.10915 | -9.221713 | 0.641946 | 0.774359 | -0.614062 |
| b | -112.982314 | -0.483922 | **24.258865** | -10.679792 | 6.502571 | -11.7999 | -2.374991 | -6.830022 | 2.1273 |
| c1 | -97281.94124 | 1.404413 | -10.679792 | **178803.7535** | -325.279697 | 101.002141 | -18.48497 | -11.596911 | -4269.338961 |
| gender | -546.044273 | -0.10915 | 6.502571 | -325.279697 | **38.493022** | -8.965189 | 0.772641 | 1.503622 | 259.265078 |
| con2 | 295.033178 | -9.221713 | -11.7999 | 101.002141 | -8.965189 | **238.550129** | -29.965479 | -24.296114 | -78.858994 |
| c2 | -23.139147 | 0.641946 | -2.374991 | -18.48497 | 0.772641 | -29.965479 | **5.436555** | 5.413505 | 14.916243 |
| a std dev. | -1.258903 | 0.774359 | -6.830022 | -11.596911 | 1.503622 | -24.296114 | 5.413505 | **7.168264** | 9.294781 |
| gender std dev. | -6332.327446 | -0.614062 | 2.1273 | -4269.338961 | 259.265078 | -78.858994 | 14.916243 | 9.294781 | **3574.613283** |

**Figure 8.13: Comparison of efficient design with and without accounting for covariates**

If the covariates are continuous in nature, then the above methods cannot be handled easily. If the above procedure is to be employed, then the number of segments that can be formed may be so large as to not be computationally possible to handle. If this is the case, then the analyst may have to resort to Monte Carlo simulations to simulate the likely data that is expected to be collected. Whilst this will generally take much longer to locate an efficient design than when using the true analytical AVC matrix, given the full factorial of possible covariate combinations that may possibly be formed by combining certain covariates, the use of Monte Carlo simulations may actually require much less time in this instance.

In Ngene, covariates are handled similarly to pivot type designs in that they require the use of the fisher property. As with pivot style designs, the fisher property may be used to generate homogenous or heterogeneous covariate style designs. Also, similar to pivot designs, the analyst is required to nominate a weight representing the proportion that each covariate will appear in the final sample. For example, assuming that the analyst wishes to construct a design allowing for a gender covariate (male = 1) and assuming that the analyst believes that males and females should be sampled equally, the fisher property for a homogeneous covariate design might look thus

```
;fisher(F1) = des1(Male[0.5], Female[0.5])
```

Similarly, the fisher property for a heterogeneous covariate design for the same example might look

```
;fisher(F1) = des1(Male[0.5]) + des2(Female[0.5])
```

Also similar to pivot designs, the analyst is required to inform Ngene what variables in the utility specification are covariates. For covariates, the suffix .covar is added after an attributes name (e.g., `gender.covar`).

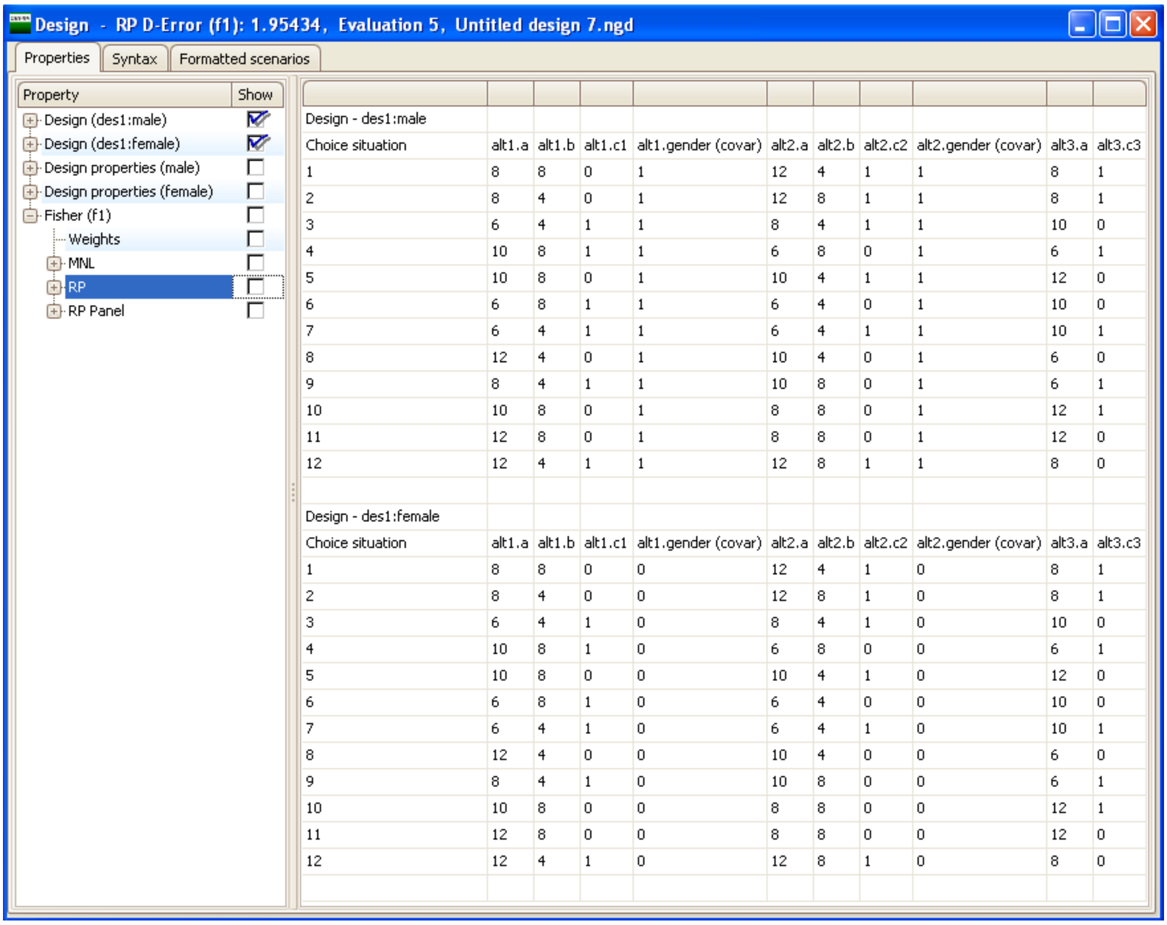
Within a single model, a number of rules exist related to the specification of covariates. Firstly, a covariate can only have one level per model (e.g., `gender.covar[1]`). As such, different levels of the covariate design must be assigned over models. Secondly, covariates can only be assigned to *J*-1 alternatives.

Example syntax for a homogeneous covariate design is represented below. Note that in setting out the utility specifications, both of the above mentioned rules are met. For example, the gender covariate appears in only 2 of the three utility functions within each model. Secondly, the two levels of the gender variable are spread over the two models ('male' and 'female').

```
Design
;alts(Male) = Alt1, Alt2, Alt3
;alts(Female) = Alt1, Alt2, Alt3
;rows = 12
;eff = F1(rp,d)
;fisher(F1) = des1(Male[0.5], Female[0.5])
;rdraws = Halton(150)
;con

;model(Male):
U(Alt1) = Con1[1.2] + A[n,-0.6,0.1] * A[6,8,10,12] + B[-0.4] * B[4,8] +
C1[0.3] * C1[0,1] + gender[n,-0.6,0.1]*gender.covar[1] /
U(Alt2) = Con2[0.8] + A             * A            + B       * B      +
C2[0.8] * C2[0,1] + gender[n,-0.6,0.1]*gender.covar[1] /
U(Alt3) =             A             * A                               +
C3[-1.0] * C3[0,1]

;model(Female):
U(Alt1) = Con1[1.2] + A[n,-0.6,0.1] * A[6,8,10,12] + B[-0.4] * B[4,8] +
C1[0.3] * C1[0,1] + gender[n,-0.6,0.1]*gender.covar[0] /
U(Alt2) = Con2[0.8] + A             * A            + B       * B      +
C2[0.8] * C2[0,1] + gender[n,-0.6,0.1]*gender.covar[0] /
U(Alt3) =             A             * A                               +
C3[-1.0] * C3[0,1]
$
```

Figure 8.14 shows a design generated using the above syntax. In presenting the output, design for the gender equal to zero (female) can be seen to be shown second, with the associated covariate columns taking the value zero for all choice situations. Similarly, the design for 'male' is shown taking the value 1 for the associated covariate columns for all choice situations.

Also, as with pivot designs, Ngene allows for the simultaneous optimization of the design and the proportions of the covariates required within the final sample collected. Once more, this is handled in the weighting section of the fisher property. Rather than assign a single weight to each level of a covariate, the analyst may specify a range of weights. This is done by separating a lower weight bound from an upper weight bound by a colon. For example, the syntax `female[0.1:0.6]` will allow the proportion of females in the final data set to have a weight anywhere between 0.1 and 0.6 in the optimization of the overall design Fisher Information matrix. Note that for this to work, the upper weight bounds provided must sum to one or more. Note also, that this function may be applied to both homogenous and heterogeneous covariate designs. For more information on this, see the earlier discussion on pivot designs (Section 8.3.2).


Design - RP D-Error (f1): 1.95434, Evaluation 5, Untitled design 7.ngd
Properties
Syntax
Formatted scenarios

| Property | Show |
|---|---|
| ⊞ Design (des1:male) | ☑ |
| ⊞ Design (des1:female) | ☑ |
| ⊞ Design properties (male) | □ |
| ⊞ Design properties (female) | □ |
| ⊟ Fisher (f1) | □ |
| Weights | □ |
| ⊞ MNL | □ |
| ⊞ RP | □ |
| ⊞ RP Panel | □ |

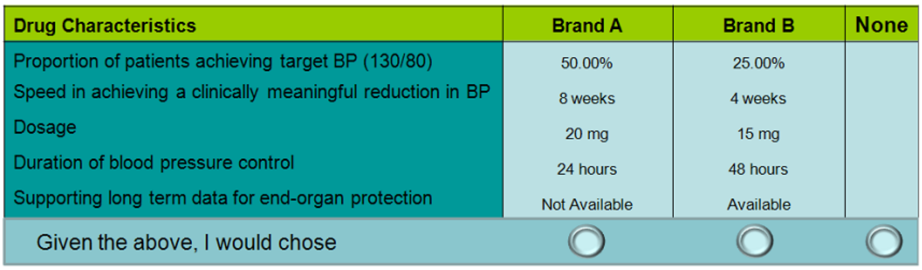
| Design - des1:male | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Choice situation | alt1.a | alt1.b | alt1.c1 | alt1.gender (covar) | alt2.a | alt2.b | alt2.c2 | alt2.gender (covar) | alt3.a | alt3.c3 |
| 1 | 8 | 8 | 0 | 1 | 12 | 4 | 1 | 1 | 8 | 1 |
| 2 | 8 | 4 | 0 | 1 | 12 | 8 | 1 | 1 | 8 | 1 |
| 3 | 6 | 4 | 1 | 1 | 8 | 4 | 1 | 1 | 10 | 0 |
| 4 | 10 | 8 | 1 | 1 | 6 | 8 | 0 | 1 | 6 | 1 |
| 5 | 10 | 8 | 0 | 1 | 10 | 4 | 1 | 1 | 12 | 0 |
| 6 | 6 | 8 | 1 | 1 | 6 | 4 | 0 | 1 | 10 | 0 |
| 7 | 6 | 4 | 1 | 1 | 6 | 4 | 1 | 1 | 10 | 1 |
| 8 | 12 | 4 | 0 | 1 | 10 | 4 | 0 | 1 | 6 | 0 |
| 9 | 8 | 4 | 1 | 1 | 10 | 8 | 0 | 1 | 6 | 1 |
| 10 | 10 | 8 | 0 | 1 | 8 | 8 | 0 | 1 | 12 | 1 |
| 11 | 12 | 8 | 0 | 1 | 8 | 8 | 0 | 1 | 12 | 0 |
| 12 | 12 | 4 | 1 | 1 | 12 | 8 | 1 | 1 | 8 | 0 |

| Design - des1:female | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Choice situation | alt1.a | alt1.b | alt1.c1 | alt1.gender (covar) | alt2.a | alt2.b | alt2.c2 | alt2.gender (covar) | alt3.a | alt3.c3 |
| 1 | 8 | 8 | 0 | 0 | 12 | 4 | 1 | 0 | 8 | 1 |
| 2 | 8 | 4 | 0 | 0 | 12 | 8 | 1 | 0 | 8 | 1 |
| 3 | 6 | 4 | 1 | 0 | 8 | 4 | 1 | 0 | 10 | 0 |
| 4 | 10 | 8 | 1 | 0 | 6 | 8 | 0 | 0 | 6 | 1 |
| 5 | 10 | 8 | 0 | 0 | 10 | 4 | 1 | 0 | 12 | 0 |
| 6 | 6 | 8 | 1 | 0 | 6 | 4 | 0 | 0 | 10 | 0 |
| 7 | 6 | 4 | 1 | 0 | 6 | 4 | 1 | 0 | 10 | 1 |
| 8 | 12 | 4 | 0 | 0 | 10 | 4 | 0 | 0 | 6 | 0 |
| 9 | 8 | 4 | 1 | 0 | 10 | 8 | 0 | 0 | 6 | 1 |
| 10 | 10 | 8 | 0 | 0 | 8 | 8 | 0 | 0 | 12 | 1 |
| 11 | 12 | 8 | 0 | 0 | 8 | 8 | 0 | 0 | 12 | 0 |
| 12 | 12 | 4 | 1 | 0 | 12 | 8 | 1 | 0 | 8 | 0 |

**Figure 8.14: An example covariate design**

## 8.5 Designs within designs: Designs with scenarios in Ngene

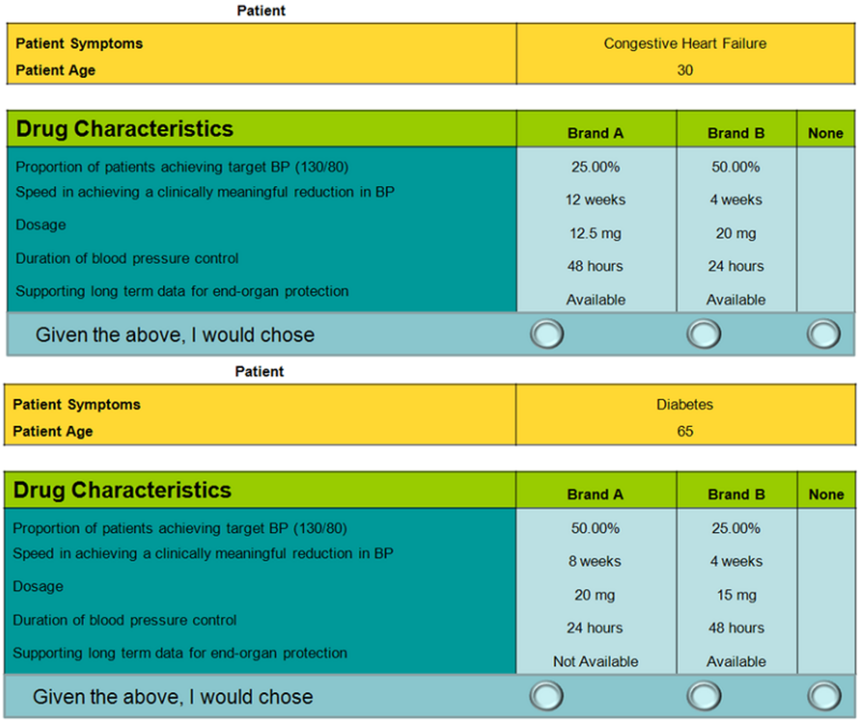
Typically, in presenting SC experiments to respondents, the analyst must first construct a scenario to frame the experiment. In most studies, the constructed scenarios are fixed over choice situations and respondents. Figure 8.15 shows an example choice situation for a health study. Above the choice situation, respondents are presented with a scenario of confronting a 30 year old patient with congenital heart disease. In most studies, this scenario would be replicated (i.e., it would not vary) over repeated choice situations.



| Drug Characteristics | Brand A | Brand B | None |
|---|---|---|---|
| Proportion of patients achieving target BP (130/80) | 50.00% | 25.00% | |
| Speed in achieving a clinically meaningful reduction in BP | 8 weeks | 4 weeks | |
| Dosage | 20 mg | 15 mg | |
| Duration of blood pressure control | 24 hours | 48 hours | |
| Supporting long term data for end-organ protection | Not Available | Available | |
| Given the above, I would chose | ○ | ○ | ○ |

**Figure 8.15: Example choice situation with a fixed scenario**

Often however, the analyst may wish to vary the scenario from one choice situation to the next. An example of this is presented in Figure 8.16 where the characteristics of the patient are varied over two different choice tasks. In varying the characteristics of the scenario over the different choice tasks, it should be noted that within any given choice task, the levels shown in the scenario are constants across all *J* alternatives. That is, the patient remains 30 years of age for the ‘Brand A’, ‘Brand B’ and ‘none’ alternatives. In this way, when setting up the experimental design, scenario characteristics should be treated in the same manner as covariate attributes in terms of being entered into only *J*-1 utility functions (alternatively, one could interact them with design attributes).



Patient

| | |
|---|---|
| Patient Symptoms | Congestive Heart Failure |
| Patient Age | 30 |

| Drug Characteristics | Brand A | Brand B | None |
|---|---|---|---|
| Proportion of patients achieving target BP (130/80) | 25.00% | 50.00% | |
| Speed in achieving a clinically meaningful reduction in BP | 12 weeks | 4 weeks | |
| Dosage | 12.5 mg | 20 mg | |
| Duration of blood pressure control | 48 hours | 24 hours | |
| Supporting long term data for end-organ protection | Available | Available | |
| Given the above, I would chose | ○ | ○ | ○ |

Patient

| | |
|---|---|
| Patient Symptoms | Diabetes |
| Patient Age | 65 |

| Drug Characteristics | Brand A | Brand B | None |
|---|---|---|---|
| Proportion of patients achieving target BP (130/80) | 50.00% | 25.00% | |
| Speed in achieving a clinically meaningful reduction in BP | 8 weeks | 4 weeks | |
| Dosage | 20 mg | 15 mg | |
| Duration of blood pressure control | 24 hours | 48 hours | |
| Supporting long term data for end-organ protection | Not Available | Available | |
| Given the above, I would chose | ○ | ○ | ○ |

**Figure 8.16: Example choice situation with changing scenarios**

In Ngene, it is possible to force an attribute level to be the same for two or more different alternatives via a slight variation in the usual utility function specification. Typically, in a utility function, the values supplied after an attribute’s name will represent the levels that that attribute may take. For example, `age[20,30,40,50]`, suggests that the age variable may take the values 20, 30, 40 or 50. In Ngene, if the age variable appears in a second alternative, it is possible to reference the original attribute and constrain the value that the attribute level in the second alternative takes to be the same as the level in the first alternative. This is done by specifying the name of the original attribute rather than providing attribute levels when writing out the utility function for the second attribute. For example,

```
U(Alt1) = A[-0.6] * age[20,30,40,50]
U(Alt2) = A       * age[age]
```

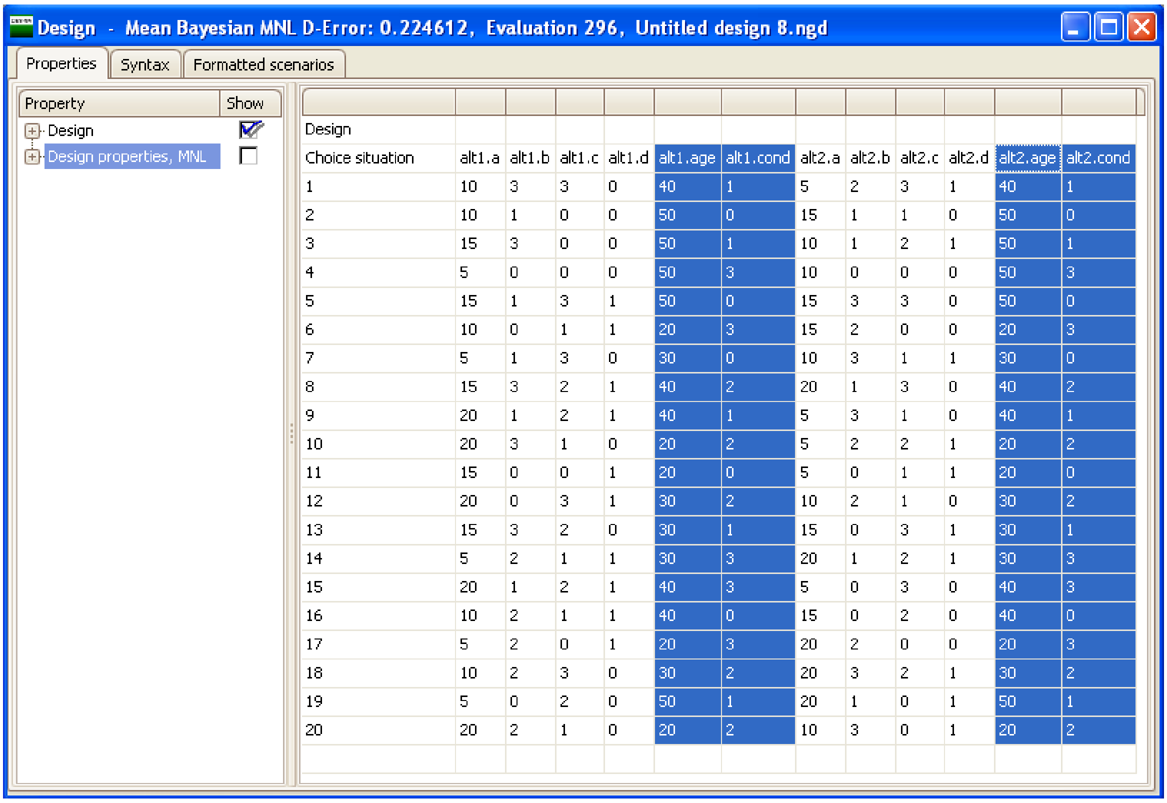
In the second alternative, the age attribute references the level provided in the first alternative and will constrain the level to be the same across the two alternatives. This is precisely what is required for designs where the analyst wishes to vary levels in the scenarios presented to respondents. Note that in setting out the syntax in this manner, the age characteristics will vary from one choice situation to the next (taking the values 20, 30, 40 or 50), but take the same value for the commonly named attribute across the two (or more) alternatives within the same choice situation. Note however, as previously mentioned, such variables may only be entered into *J*-1 utility functions unless they are to be specified as interaction effects. Thus, whilst the attribute may appear in any choice situation presented to a respondent across all *J* alternatives (as per Figure

8.16), in the actual modeling process, the variable will need to be treated similar to any socio-demographic variable. Thus in the example given in Figure 8.16, the patient characteristics could be treated as factors differentiating between a respondent choosing to prescribe medication (either ‘Brand A’ or ‘Brand B’) relative to not prescribing any medication.

Example syntax showing the full set of syntax for the above example is given below. In this syntax, we have two such scenario attributes.

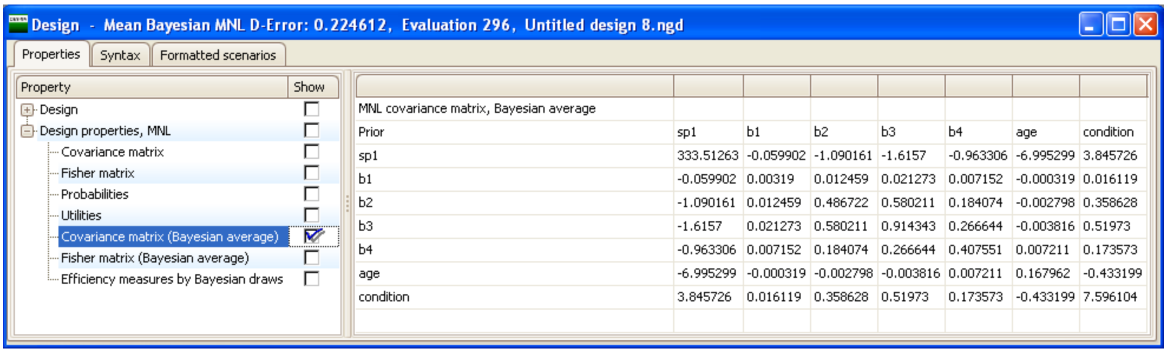
```
Design
;alts = alt1, alt2, alt3
;rows = 20
;eff = (mnl,d,mean)
;bdraws = halton(150)
;model:
U(Alt1) = SP1[3.2] +
          b1[(n,0.07,0.03)] * A[5,10,15,20]    +
          b2[(n,1.2,0.3)]   * B[0,1,2,3]       +
          b3[1.8]           * C[0,1,2,3]       +
          b4[0.6]           * D[0,1]           +
          age[-0.06]        * age[20,30,40,50] +
          condition[0.4]    * cond[0,1,2,3]    /
U(Alt2) = SP1[3.4]                             +
          b1                * A                +
          b2                * B                +
          b3                * C                +
          b4                * D                +
          age               * age[age]         +
          condition         * cond[cond]
$
```

Figure 8.17 shows output based on the above output. In the figure, we have highlighted the ‘age’ and ‘cond’ variables as they appear in the design to demonstrate that they are indeed constrained to take the same levels across alternatives. Figure 8.18 shows the AVC matrix for the design given in Figure 8.17. Note that the ‘age’ and ‘cond’ variables are represented in this matrix.


Design - Mean Bayesian MNL D-Error: 0.224612, Evaluation 296, Untitled design 8.ngd
Properties
Syntax
Formatted scenarios
Property
Show
Design ☑
Design properties, MNL □

| Design | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Choice situation | alt1.a | alt1.b | alt1.c | alt1.d | alt1.age | alt1.cond | alt2.a | alt2.b | alt2.c | alt2.d | alt2.age | alt2.cond |
| 1 | 10 | 3 | 3 | 0 | 40 | 1 | 5 | 2 | 3 | 1 | 40 | 1 |
| 2 | 10 | 1 | 0 | 0 | 50 | 0 | 15 | 1 | 1 | 0 | 50 | 0 |
| 3 | 15 | 3 | 0 | 0 | 50 | 1 | 10 | 1 | 2 | 1 | 50 | 1 |
| 4 | 5 | 0 | 0 | 0 | 50 | 3 | 10 | 0 | 0 | 0 | 50 | 3 |
| 5 | 15 | 1 | 3 | 1 | 50 | 0 | 15 | 3 | 3 | 0 | 50 | 0 |
| 6 | 10 | 0 | 1 | 1 | 20 | 3 | 15 | 2 | 0 | 0 | 20 | 3 |
| 7 | 5 | 1 | 3 | 0 | 30 | 0 | 10 | 3 | 1 | 1 | 30 | 0 |
| 8 | 15 | 3 | 2 | 1 | 40 | 2 | 20 | 1 | 3 | 0 | 40 | 2 |
| 9 | 20 | 1 | 2 | 1 | 40 | 1 | 5 | 3 | 1 | 0 | 40 | 1 |
| 10 | 20 | 3 | 1 | 0 | 20 | 2 | 5 | 2 | 2 | 1 | 20 | 2 |
| 11 | 15 | 0 | 0 | 1 | 20 | 0 | 5 | 0 | 1 | 1 | 20 | 0 |
| 12 | 20 | 0 | 3 | 1 | 30 | 2 | 10 | 2 | 1 | 0 | 30 | 2 |
| 13 | 15 | 3 | 2 | 0 | 30 | 1 | 15 | 0 | 3 | 1 | 30 | 1 |
| 14 | 5 | 2 | 1 | 1 | 30 | 3 | 20 | 1 | 2 | 1 | 30 | 3 |
| 15 | 20 | 1 | 2 | 1 | 40 | 3 | 5 | 0 | 3 | 0 | 40 | 3 |
| 16 | 10 | 2 | 1 | 1 | 40 | 0 | 15 | 0 | 2 | 0 | 40 | 0 |
| 17 | 5 | 2 | 0 | 1 | 20 | 3 | 20 | 2 | 0 | 0 | 20 | 3 |
| 18 | 10 | 2 | 3 | 0 | 30 | 2 | 20 | 3 | 2 | 1 | 30 | 2 |
| 19 | 5 | 0 | 2 | 0 | 50 | 1 | 20 | 1 | 0 | 1 | 50 | 1 |
| 20 | 20 | 2 | 1 | 0 | 20 | 2 | 10 | 3 | 0 | 1 | 20 | 2 |

**Figure 8.17: Example design output with constrained attribute levels**


Design - Mean Bayesian MNL D-Error: 0.224612, Evaluation 296, Untitled design 8.ngd
Properties
Syntax
Formatted scenarios
Property
Show
Design □
Design properties, MNL □
Covariance matrix □
Fisher matrix □
Probabilities □
Utilities □
Covariance matrix (Bayesian average) ☑
Fisher matrix (Bayesian average) □
Efficiency measures by Bayesian draws □

| MNL covariance matrix, Bayesian average | | | | | | | |
|---|---|---|---|---|---|---|---|
| Prior | sp1 | b1 | b2 | b3 | b4 | age | condition |
| sp1 | 333.51263 | -0.059902 | -1.090161 | -1.6157 | -0.963306 | -6.995299 | 3.845726 |
| b1 | -0.059902 | 0.00319 | 0.012459 | 0.021273 | 0.007152 | -0.000319 | 0.016119 |
| b2 | -1.090161 | 0.012459 | 0.486722 | 0.580211 | 0.184074 | -0.002798 | 0.358628 |
| b3 | -1.6157 | 0.021273 | 0.580211 | 0.914343 | 0.266644 | -0.003816 | 0.51973 |
| b4 | -0.963306 | 0.007152 | 0.184074 | 0.266644 | 0.407551 | 0.007211 | 0.173573 |
| age | -6.995299 | -0.000319 | -0.002798 | -0.003816 | 0.007211 | 0.167962 | -0.433199 |
| condition | 3.845726 | 0.016119 | 0.358628 | 0.51973 | 0.173573 | -0.433199 | 7.596104 |

**Figure 8.18: Example design output with constrained attribute levels**

## 8.6 Algorithms for generating designs in Ngene

When executing the syntax (see Chapter 4), Ngene will generate a design according to the specified properties. Different search algorithms have been implemented in order to generate a design. Depending on the properties set, different algorithms will be defaulted by Ngene. If using the default, no algorithm has to be specified in the syntax. However, if one would like to overrule the default, or change settings of the algorithms, then one could add the *alg* property in the syntax.

For different types of designs (e.g., orthogonal, efficient, orthogonal efficient, with constraints, etc.) different algorithms are used, including RSC (relabelling-swapping-cycling) algorithms, swapping algorithms, and Modified Federov algorithms. For efficient designs, the swapping algorithm is the default. The parameters for this algorithm can be changed, or even a different algorithm can be selected.

If one would like to use a specific algorithm, one can specify this in the *alg* property, by choosing one of the following:

```
;alg = swap
;alg = rsc
;alg = mfederov
```

Note that for the RSC algorithm, different combinations of the individual aspects of the algorithm can be employed. For example, one could employ only the relabeling and cycling methods by specifying only the appropriate letters in the *alg* property. This is shown below.

```
;alg = rc
```

For example

```
Design
;alts = alt1, alt2, alt3
;rows = 12
;eff = (mnl,d)
;alg = rc
;model:
U(alt1) = b11[-0.2] + b2[1.2] * A[0,1,2] + b3[2.5] * B[0,1]     /
U(alt2) = b12[0.3]  + b2      * A        + b4[1.1] * C[2,4,6,8]
$
```

will use only relabeling and cycling in searching for an efficient design.

Each algorithm also has a number of default settings. One can overrule these default settings by changing the settings in the *alg* property. For example,

```
;alg = swap(random = 50, swap = 10, swaponimprov = 20, reset = 200,
resetinc = 50)
```

For a more detailed explanation of all the algorithm settings we refer to the alg property in the Syntax Reference.

It is also possible to use an existing design as the initial starting design for the algorithm (which for example can be used as a starting point for the swapping algorithm), one can add the start property to the syntax, defining the filename of the initial design. First, the design should be present in the project, either by importing a Microsoft Excel file (*.xls, *.xlsx, *.xlsm) or importing an Ngene design file (*.ngd), see Section 3.3. Then it can be used as an initial design in the algorithm, for example:

```
;start = efficient design.xls
```

or

```
;start = design.ngd
```

Note that spaces are allowed in the filename.

By default, the Modified Federov algorithm attempts to generate a candidate set of 2000 choice sets. The candidate set size can be altered. For example, to generate a candidate set with 5000 choice tasks, specify:

```
;alg = mfederov(candidates=5000)
```

There are a number of reasons why the default or specified candidate set size cannot be achieved. The full factorial of choice tasks might be smaller in size, especially once dominance or row repetition checks are performed (see Section 8.8), or if constraints are specified (see Section 8.2). If this is the case, Ngene will automatically utilise a smaller candidate set size, and provide some statistics on why the default or specified candidate set size cannot be achieved.

In some cases, it may be desirable to have complete control over the composition of the choice tasks in the candidate set. Examples include partial profiles designs, in which only a subset of attributes are varied in each choice task (see Section 8.10), and partial choice set/availability designs, in which only a subset of alternatives are shown in each choice task (see Section 8.11).

A candidate set can be loaded in from the complete set of choice tasks in an existing 'design'. The number of choice tasks in this design is arbitrarily large and can be controlled by the analyst. As with the *start* syntax, the design should be present in the project, either by importing a Microsoft Excel file (*.xls, *.xlsx, *.xlsm) or importing an Ngene design file (*.ngd), see Section 3.3. For example:

```
;alg = mfederov(candidates=designForCandidates.xls)
```

or

```
;alg = mfederov(candidates=designForCandidates.ngd)
```

Most algorithms will keep running indefinitely. It is possible to force an algorithm to stop after a certain amount of time, or a certain number of iterations. For example,

```
;alg = swap(stop=total(10 mins))
```

will run the swapping algorithm for a total of 10 minutes, and

```
;alg = mfederov(stop=total(100000 iterations))
```

will run the Modified Federov algorithm for a total of 100000 iterations. An algorithm can also be instructed to stop after a specified amount of time or number of iterations since the latest improvement was found. For example

```
;alg = swap(stop=noimprov(80 secs))
```

will run the swapping algorithm until 80 seconds have elapsed since an improvement was found.

It is also possible to run several algorithms one after the other, so long as all but the last algorithm have stopping criteria. The best design found from the previous algorithm will be used as the starting design of the current algorithm. Specify a single alg property, and place a comma between the algorithms you wish to run. For example:

```
;alg = mfederov(stop=total(10 secs)), swap
```

Finally, for very small designs, it may be possible to sequentially evaluate all possible designs. This can be achieved by specifying ;alg=all . However it is only feasible for very small designs. The percentage of all possible designs evaluated so far is shown below the trace in the output window, in addition to the current evaluation.

```
;alg = all
```

## 8.7 Evaluating existing designs in Ngene

Instead of generating a new design, one may be interested in evaluating an existing design, for example to check the efficiency under certain model assumptions. Similar as to using a start design in an algorithm, we can read in an existing design that is currently in the project (importing again a Microsoft Excel file or an Ngene design file), and refer to this file in the *eval* property. This property overrules the *alg* property, in the sense that it will not search for a better design, but merely evaluates the design and then finishes. For example,

```
;eval = efficient design.ngd
```

Instead of just evaluating the design, it is also possible to block an existing design (independent of whether it was originally blocked or not). If the *block* property has been specified in the syntax, Ngene will use this to block the design that is being evaluated. If a blocking column already exists (possibly with a different number of blocks), it will be replaced with a new blocking column. For example,

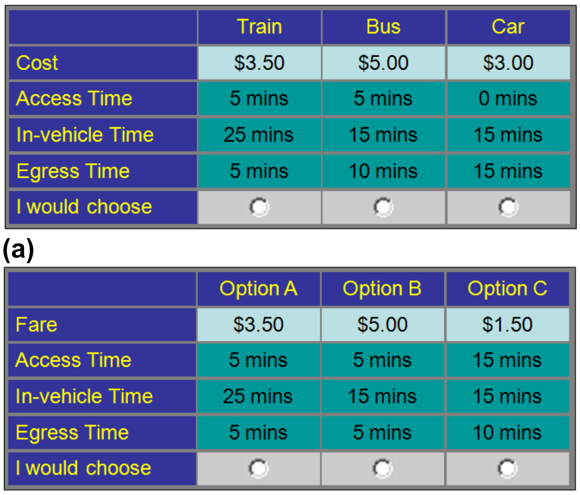
```
;block = 3
;eval = efficient design.ngd
```

Ngene will read in the design, evaluate it (with whatever model and efficiency measure specified in the syntax), and also block the design in 3 blocks by adding (or replacing) a blocking column in the design.

Note that if the design was saved using an evaluation version of Ngene, the number 0 was inserted in place of all design levels, and so you will be unable to use the *eval* property.

## 8.8 Handling unlabeled alternatives

Stated choice experiments can contain either labeled or unlabeled alternatives. Labeled alternatives occur where a heading conveys some meaning to the respondent beyond the order of the alternative shown, for example, bus, car and train (see Figure 8.19a). Unlabeled alternatives occur where headings convey no pertinent meaning beyond the order of the alternatives shown, for example, Option A, Option B, etc (see Figure 8.19b for an example). Labeled and unlabeled choice experiments are typically used for different purposes and to achieve different outcomes. For forecasting purposes where brand may influence preference, labeled alternatives may be preferred. Labeled experiments may also be preferred when one wishes to generate brand specific willingness to pay values. Where forecasting is not the main objective of the study, but where understanding preferences is, unlabeled experiments may be preferred as such experiments remove brand influences from the choice and hence focus the trade-offs upon the attributes in the study. There exist advantages and disadvantages for each type of experiment and the interested reader is referred to Hensher et al. (2005) for a full discussion.



| | Train | Bus | Car |
|---|---|---|---|
| Cost | $3.50 | $5.00 | $3.00 |
| Access Time | 5 mins | 5 mins | 0 mins |
| In-vehicle Time | 25 mins | 15 mins | 15 mins |
| Egress Time | 5 mins | 10 mins | 15 mins |
| I would choose | ○ | ○ | ○ |

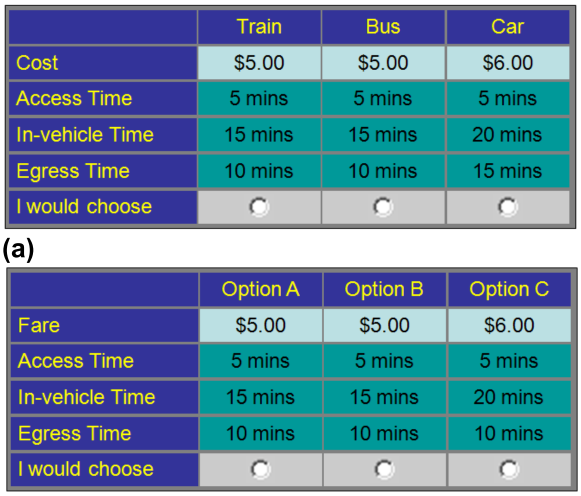
(a)

| | Option A | Option B | Option C |
|---|---|---|---|
| Fare | $3.50 | $5.00 | $1.50 |
| Access Time | 5 mins | 5 mins | 15 mins |
| In-vehicle Time | 25 mins | 15 mins | 15 mins |
| Egress Time | 5 mins | 5 mins | 10 mins |
| I would choose | ○ | ○ | ○ |

(b)

**Figure 8.19: Example labeled and unlabeled stated choice tasks**

When generating an experimental design, the distinction between labeled and unlabeled experiments may be an important one. To understand why, consider the labeled and unlabeled choice tasks shown in Figure 8.20. In both cases we have used the same attributes and attribute levels, with the only difference being the headings of the alternatives. Two things stand out in terms of the attribute levels we have chosen. Firstly, the attribute levels of the first two alternatives are exactly the same. Secondly, the attribute levels of the last alternative are always the same, or worse than the first two alternatives. When we consider the labeled experiment example, the fact that the train and bus alternatives have the same attribute levels is not too problematic in that respondents may still differentiate between the two alternatives based on the fact that one is a train and one is a bus. As such, any respondent observed to choose the train alternative is revealing a preference for train over bus, all other things being constant. Similarly, a respondent observed to choose the bus alternative is revealing a preference for bus over train, all other things being constant. Consider now the fact that the attribute levels of the car alternative are never better than the other alternatives. Such a situation does not preclude the possibility of a respondent rationally selecting the car alternative if both the unobserved and observed effects combined (i.e., the overall utility) associated with car is greater than that of both train and bus. In terms of generating the experimental design, this situation may manifest itself via a larger positive alternative specific constant associated with the car alternative than for the train or bus alternatives.



| | Train | Bus | Car |
|---|---|---|---|
| Cost | $5.00 | $5.00 | $6.00 |
| Access Time | 5 mins | 5 mins | 5 mins |
| In-vehicle Time | 15 mins | 15 mins | 20 mins |
| Egress Time | 10 mins | 10 mins | 15 mins |
| I would choose | ○ | ○ | ○ |

**(a)**

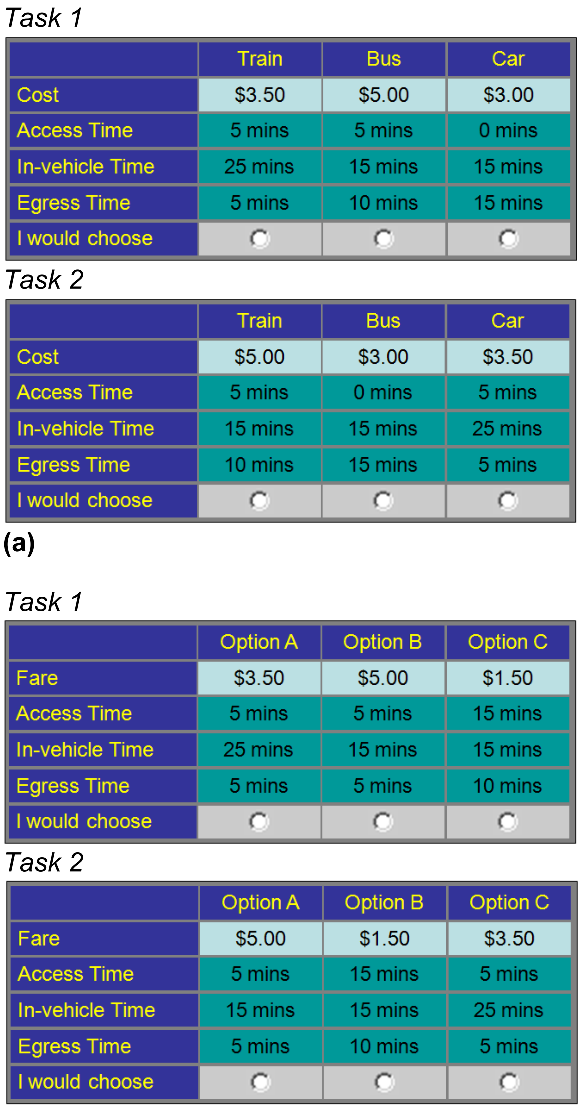
| | Option A | Option B | Option C |
|---|---|---|---|
| Fare | $5.00 | $5.00 | $6.00 |
| Access Time | 5 mins | 5 mins | 5 mins |
| In-vehicle Time | 15 mins | 15 mins | 20 mins |
| Egress Time | 10 mins | 10 mins | 10 mins |
| I would choose | ○ | ○ | ○ |

**(b)**

**Figure 8.20: Examples of problematic labeled and unlabeled stated choice tasks**

Now consider the unlabeled choice task. The two issues discussed above, that is, two alternatives taking the same attribute levels, and the fact that one alternative is never better on any attribute, will now have a different impact upon how respondents would be expected to react to the experiment. Taking the case of the first two alternatives being the same, any respondent facing this situation will not be able to distinguish between the two alternatives and hence the choice becomes purely random (however this may strictly not be true, as most people read left to right and hence the left most alternative, option A in this case, is more likely to be selected). Thinking about the second issue presented in the choice task, that is the fact that the last alternative is never better on any attribute, then there exists no rational explanation for a respondent to select this alternative (i.e., other than left to right bias in answering the question, there is no reason that the unobserved effects of the option should be any better or worse than the other alternatives, that is unless the respondent has a fetish for the words option C, a highly improbable circumstance). We call such alternatives dominated alternatives. As such, issues of alternatives being dominated and the repetition of all attribute levels across alternatives may have a larger bearing on generating unlabeled choice experiments than when generating labeled ones. This is not to suggest that dominance and attribute level repetition may not be important for labeled choice experiments. Indeed, labeled choice experiments may have dominated alternatives, however the dominance occurs purely as a result of preferences for the labeled alternatives and not purely as an artefact of the attribute levels being dominated.

An additional concern typically associated with unlabeled choice experiments relates to the fact that the order of combination of the attributes associated with alternatives matters over the experimental design, much more so than with labeled choice experiments. To see why, consider the series of choice tasks shown in Figure 8.21. Assume that we were to present the two labeled choice tasks given in Figure 8.21a to a respondent. Examination of the two choice tasks reveals that the bundles of attribute levels we have used are the same, however the alternatives that we have assigned these bundles of attributes to are different across the two choice tasks. In this instance, rotating the entire bundle of attributes across the alternatives has not impacted upon how sensible the overall survey would be to any given respondent. As such, the order of bundles of attribute levels is not likely to have a behavioural impact upon the design (it might have a statistical impact however depending upon the parameter priors).



*Task 1*

| | Train | Bus | Car |
|---|---|---|---|
| Cost | $3.50 | $5.00 | $3.00 |
| Access Time | 5 mins | 5 mins | 0 mins |
| In-vehicle Time | 25 mins | 15 mins | 15 mins |
| Egress Time | 5 mins | 10 mins | 15 mins |
| I would choose | ○ | ○ | ○ |

*Task 2*

| | Train | Bus | Car |
|---|---|---|---|
| Cost | $5.00 | $3.00 | $3.50 |
| Access Time | 5 mins | 0 mins | 5 mins |
| In-vehicle Time | 15 mins | 15 mins | 25 mins |
| Egress Time | 10 mins | 15 mins | 5 mins |
| I would choose | ○ | ○ | ○ |

**(a)**

*Task 1*

| | Option A | Option B | Option C |
|---|---|---|---|
| Fare | $3.50 | $5.00 | $1.50 |
| Access Time | 5 mins | 5 mins | 15 mins |
| In-vehicle Time | 25 mins | 15 mins | 15 mins |
| Egress Time | 5 mins | 5 mins | 10 mins |
| I would choose | ○ | ○ | ○ |

*Task 2*

| | Option A | Option B | Option C |
|---|---|---|---|
| Fare | $5.00 | $1.50 | $3.50 |
| Access Time | 5 mins | 15 mins | 5 mins |
| In-vehicle Time | 15 mins | 15 mins | 25 mins |
| Egress Time | 5 mins | 10 mins | 5 mins |
| I would choose | ○ | ○ | ○ |

**(b)**

**Figure 8.21: Example of choice tasks with repeated alternatives**

Now consider the two unlabeled choice tasks shown in Figure 8.21b. As with the two labeled choice tasks, we have simply rotated the bundles of attribute levels that make up the alternatives of the first choice task to make up the new choice task. Now, given the unlabeled nature of the experiment, the order that the attribute level bundles appear do matter. This is because if the respondent demonstrated a preference for the bundle of attribute levels associated with Option B in the first choice task, then clearly they should prefer Option A in the second choice task (again, ignoring any preference the respondent may have for the words ‘Option B’). As such, when a particular combination of attribute levels is repeated in an unlabeled choice experiment, even if the attribute level bundles are associated with different alternatives, no additional information is theoretically obtained from the respondent.

It is possible to prevent these problems from occurring in Ngene. This is achieved via the *alts* property, by placing an asterisk next to the names of the alternatives that one wants to prevent from having
i) within choice task alternative repetition,
ii) strict attribute level dominance and
iii) choice task repetition given attribute bundle ordering.

Whilst this may apply to labeled choice experiments, it is more likely to prove useful in generating unlabeled choice experiments. To demonstrate the property, consider

```
;alts = alt1*, alt2*, alt3*
```

Note that several other conditions must be met in the specification of the utility expressions to allow the checks to take place. To prevent within choice task alternative repetition (i), and to prevent row repetition in unlabeled choice situations (iii), all attribute names must be identical in the alternatives that are to be compared. Every attribute specified in one alternative must be specified in the other, and vice versa. Failure to do this for any alternative pair will result in the alternative repetition check not being performed for that alternative pair, and a warning being issued. The order in which the attributes are specified must not vary across alternatives. The presence of an alternative specific constant, while unusual for unlabeled alternatives, will not affect the check for repeated alternatives or row repetitions.

This design would be checked for alternative repetition:

```
Design
;alts = alt1*, alt2*
;rows=9
;eff=(mnl,d)
;model:
U(alt1) = a[-0.1]*A[96,114,126,144] + b[-0.5]*B[25,28,31,34] + c[0.1]*C
[20,40,60,80] /
U(alt2) = a*A                       + b*B                    + c*C

$
```

while this would not:

```
Design
;alts = alt1*, alt2*
;rows=9
;eff=(mnl,d)
;model:
U(alt1) = a[-0.1]*A[96,114,126,144] + b[-0.5]*B[25,28,31,34] + c[0.1]*C
[20,40,60,80] /
U(alt2) = a*A                       + b*B                    + c*D
[25,45,65,85]
$
```

To prevent dominance, all prior names must be identical in the alternatives that are to be compared. Every prior specified in one alternative must be specified in the other, and vice versa. Failure to do this for any alternative pair will result in the dominance check not being performed for that alternative pair, and a warning being issued. The order in which the priors are specified must not vary across alternatives. If multiple model specifications are provided for the same underlying design, the dominance check will be performed for each model specification. Failure of the dominance check by a choice situation on *any* of the model specifications will result in the design being rejected.

## 8.9 Handling probabilities and other attributes that must sum to a number

In some situations, it is necessary to ensure that the attribute levels of multiple attributes sum to a certain number, within each choice alternative. A key application is when probabilities are attached to various outcomes.

Consider for example an SC choice scenario which contains two alternative travel routes. The travel times via these routes vary from one trip to the next, resulting in what could broadly be called early, on time, and late trips, where each of these times may be experienced with a certain probability. The travel times will be attributes in the choice scenario, but so too will the probabilities. The challenge then is to constrain the probabilities to sum to one.

In Ngene, such a constraint cannot readily be achieved with mechanisms such as the ;cond and ; reject properties. An alternative approach is to specify levels for all probabilities bar one, then define the final probability as one minus the sum of all other probabilities. This can be achieved using the attribute level function feature, by placing 'fcn()' within the square brackets that define the attribute levels, and placing an expression within these round brackets. Syntax for the above example is provided below:

```
Design
;alts = alt1, alt2
;rows = 12
;eff = (mnl,d)
;alg = swap
;model:
U(alt1) = 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 - alt1.prEarly - alt1.prOntime)]
                   * Late[25,27,29] /
U(alt2) = b1 * prEarly * Early +
          b2 * prOntime * Ontime +
          b3 * prLate[fcn(1 - alt2.prEarly - alt2.prOntime)] * Late
$
```

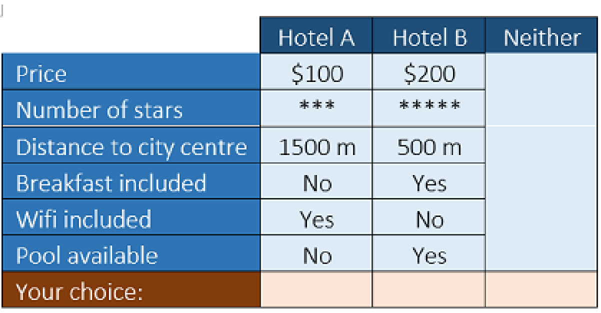
Care must be taken to ensure that no combination of explicitly defined probabilities can exceed one. Note that each attribute in the function is defined by both the alternative and attribute names, with a full stop placed in between. Also, in this example, the probability attributes enter the utility expression only within an interaction (possible since version 1.1), although they could also enter the utility expression as a main effect. At this point in time, only constants, attributes, and plus and minus symbols can enter the expression. When functions are employed, only column based algorithms can be used. This excludes the modified Federov and RSC algorithms, orthogonal designs, and optimal orthogonal in the difference (OOD) designs.

## 8.10 Varying only a subset of attributes in each choice task

Choice tasks can become quite complex for respondents when the number of attributes is large, which puts a significant cognitive burden on the respondent and may increase error variance and attribute non-attendance. Instead of letting respondents trade off on all attributes in each choice task, one can let respondents only trade off on a subset of attributes in each choice task. This can be achieved by creating a *partial profile design*.

In typical experiment designs, each alternative is represented by a full profile of attributes and corresponding attribute levels. For example, consider the experiment in Figure 8.22, and the following syntax that generates an efficient experimental design:

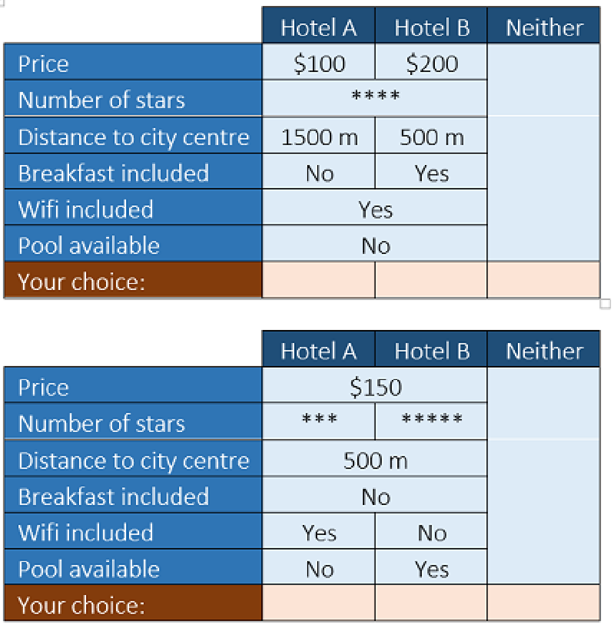
```
Design
;alts = hotelA*, hotelB*, neither
;rows = 12
;eff = (mnl,d)
;model:
U(hotelA)  = b1[-0.01]      * price[100,150,200]      +
             b2[-0.2]       * stars[1,3,5]            +
             b3[-0.001]     * distance[500,1000,1500] +
             b4.dummy[0.3]  * wifi[1,0]               +
             b5.dummy[0.2]  * breakfast[1,0]          +
             b6.dummy[0.2]  * pool[1,0]               +
             b7[0.0002]     * price * stars           /
U(hotelB)  = b1             * price                   +
             b2             * stars                   +
             b3             * distance                +
             b4             * wifi                    +
             b5             * breakfast               +
             b6             * pool                    +
             b7             * price * stars           /
U(neither) = b0[-3.0]
$
```



| | Hotel A | Hotel B | Neither |
|---|---|---|---|
| Price | $100 | $200 | |
| Number of stars | *** | ***** | |
| Distance to city centre | 1500 m | 500 m | |
| Breakfast included | No | Yes | |
| Wifi included | Yes | No | |
| Pool available | No | Yes | |
| Your choice: | | | |

**Figure 8.22: Choice task with full profile with six (non-overlapping) attributes**

One can reduce choice task complexity by forcing some of the attributes to have the same level across multiple alternatives. Such a design is referred to as an *explicit* partial profile design in which overlap between attributes is explicitly shown. Figure 8.23 illustrates such a case in which we have three attributes with overlapping levels in each choice task. It is clear that overlap can only be created if there exist attributes (with the same attribute levels to choose from) that appear in multiple alternatives.



| | Hotel A | Hotel B | Neither |
|---|---|---|---|
| Price | $100 | $200 | |
| Number of stars | **** | | |
| Distance to city centre | 1500 m | 500 m | |
| Breakfast included | No | Yes | |
| Wifi included | Yes | | |
| Pool available | No | | |
| Your choice: | | | |

| | Hotel A | Hotel B | Neither |
|---|---|---|---|
| Price | $150 | | |
| Number of stars | *** | ***** | |
| Distance to city centre | 500 m | | |
| Breakfast included | No | | |
| Wifi included | Yes | No | |
| Pool available | No | Yes | |
| Your choice: | | | |

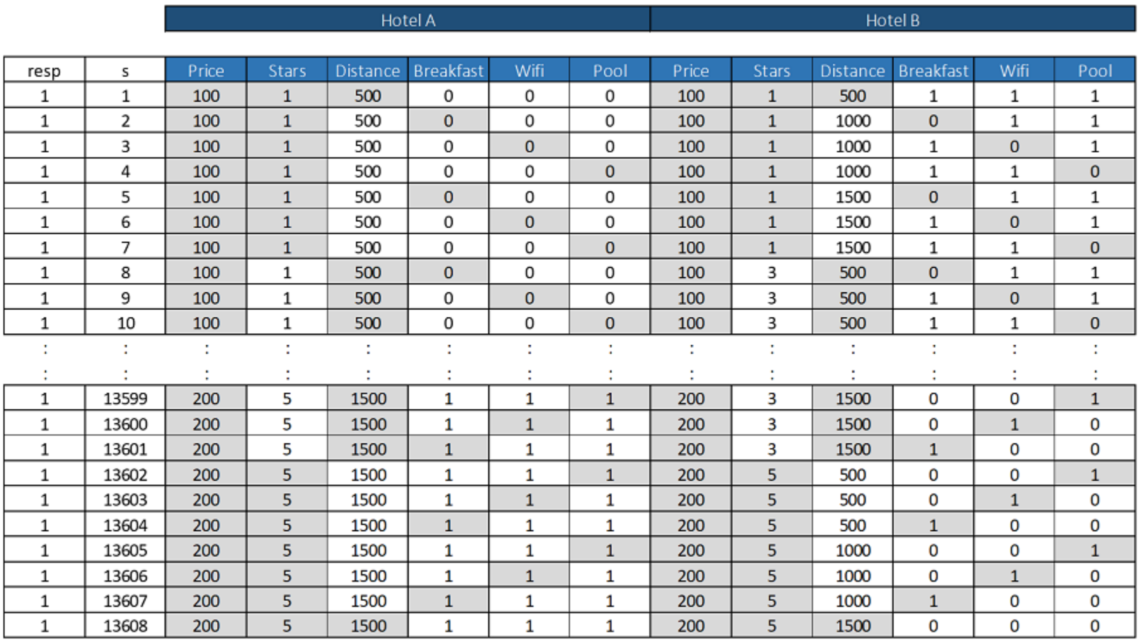
**Figure 8.23: Choice tasks with explicit partial profiles**

In Ngene, the modified Federov algorithm can be used to generate efficient designs with explicit partial profiles. In order to achieve this, a candidate set needs to be created externally, for example in Excel, R, or Matlab. Figure 8.24 illustrates such an external candidate set that contains all 13,608 possible combinations in which three out of six attributes have overlapping levels across the two generic alternatives. Overlapping levels across alternatives are indicated in grey (although such formatting should not exist in the actual file). Using such a large candidate set can make the modified Federov algorithm very slow. It is typically not necessary to include all possible combinations in the candidate set, therefore one could also create an incomplete candidate set that only contains a subset, e.g. 1,000 randomly selected out of these 13,608 choice tasks. The candidate set can be used in the syntax as follows:

```
;alg = mfederov(candidates = explicit_partial_profiles.csv)
```

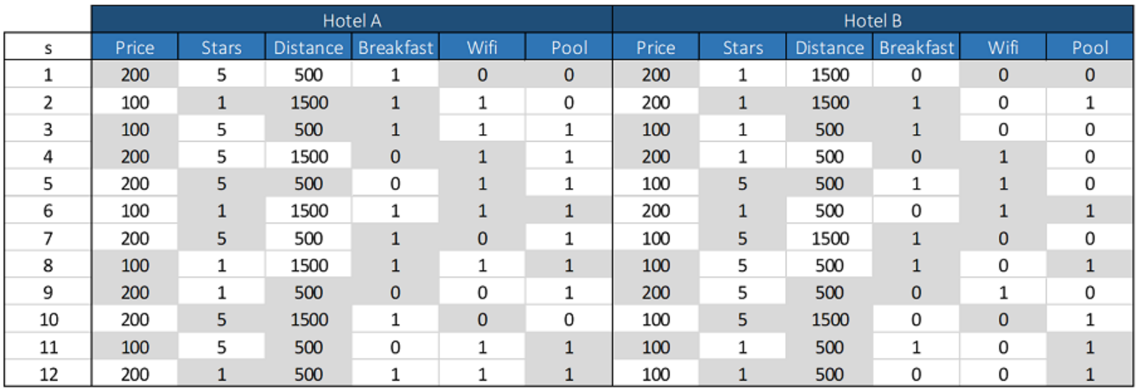
where `explicit_partial_profiles.csv` is the name of the file containing the (complete or incomplete) candidate set. Note that the format of this candidate set is the same as any other design that Ngene can read in for evaluation using `;eval` (see Section 8.7). In other words, one can use the format shown in Figure 8.24 (which has a header row that is ignored by Ngene, a first column indicating that we are generating a homogeneous design for a single respondent, a second column with the choice task number, and all next columns are for attributes in the order in which they appear within the syntax). Running the syntax then produces an experimental design similar to the one shown in Figure 8.25 (again for convenience using grey shading to indicate attribute level overlap).

There does not need to be a lot of intelligence in the candidate set, Ngene automatically selects appropriate choice tasks from the candidate set. In the syntax example above, with the asterisk behind `hotelA` and `hotelB`, Ngene automatically removes choice tasks in which both profiles are identical, automatically avoids choice tasks with dominant alternatives, and automatically avoids repetitions of the same choice task within the design (see Section 8.8 for more information). It is further possible to add constraints using the *;reject* or *;require* command in order to automatically remove other choice tasks from the candidate set as well. The modified Federov algorithm does the rest of the work in selecting the best choice tasks to include in the design. Note that that the modified Federov algorithm cannot guarantee attribute level balance, hence, whilst not strictly necessary, one may wish to add hard constraints on the number of times each attribute level appears within the design, or alternatively impose a soft level balance constraint, see Section 8.1.



| | | Hotel A | | | | | | Hotel B | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| resp | s | Price | Stars | Distance | Breakfast | Wifi | Pool | Price | Stars | Distance | Breakfast | Wifi | Pool |
| 1 | 1 | 100 | 1 | 500 | 0 | 0 | 0 | 100 | 1 | 500 | 1 | 1 | 1 |
| 1 | 2 | 100 | 1 | 500 | 0 | 0 | 0 | 100 | 1 | 1000 | 0 | 1 | 1 |
| 1 | 3 | 100 | 1 | 500 | 0 | 0 | 0 | 100 | 1 | 1000 | 1 | 0 | 1 |
| 1 | 4 | 100 | 1 | 500 | 0 | 0 | 0 | 100 | 1 | 1000 | 1 | 1 | 0 |
| 1 | 5 | 100 | 1 | 500 | 0 | 0 | 0 | 100 | 1 | 1500 | 0 | 1 | 1 |
| 1 | 6 | 100 | 1 | 500 | 0 | 0 | 0 | 100 | 1 | 1500 | 1 | 0 | 1 |
| 1 | 7 | 100 | 1 | 500 | 0 | 0 | 0 | 100 | 1 | 1500 | 1 | 1 | 0 |
| 1 | 8 | 100 | 1 | 500 | 0 | 0 | 0 | 100 | 3 | 500 | 0 | 1 | 1 |
| 1 | 9 | 100 | 1 | 500 | 0 | 0 | 0 | 100 | 3 | 500 | 1 | 0 | 1 |
| 1 | 10 | 100 | 1 | 500 | 0 | 0 | 0 | 100 | 3 | 500 | 1 | 1 | 0 |
| : | : | : | : | : | : | : | : | : | : | : | : | : | : |
| : | : | : | : | : | : | : | : | : | : | : | : | : | : |
| 1 | 13599 | 200 | 5 | 1500 | 1 | 1 | 1 | 200 | 3 | 1500 | 0 | 0 | 1 |
| 1 | 13600 | 200 | 5 | 1500 | 1 | 1 | 1 | 200 | 3 | 1500 | 0 | 1 | 0 |
| 1 | 13601 | 200 | 5 | 1500 | 1 | 1 | 1 | 200 | 3 | 1500 | 1 | 0 | 0 |
| 1 | 13602 | 200 | 5 | 1500 | 1 | 1 | 1 | 200 | 5 | 500 | 0 | 0 | 1 |
| 1 | 13603 | 200 | 5 | 1500 | 1 | 1 | 1 | 200 | 5 | 500 | 0 | 1 | 0 |
| 1 | 13604 | 200 | 5 | 1500 | 1 | 1 | 1 | 200 | 5 | 500 | 1 | 0 | 0 |
| 1 | 13605 | 200 | 5 | 1500 | 1 | 1 | 1 | 200 | 5 | 1000 | 0 | 0 | 1 |
| 1 | 13606 | 200 | 5 | 1500 | 1 | 1 | 1 | 200 | 5 | 1000 | 0 | 1 | 0 |
| 1 | 13607 | 200 | 5 | 1500 | 1 | 1 | 1 | 200 | 5 | 1000 | 1 | 0 | 0 |
| 1 | 13608 | 200 | 5 | 1500 | 1 | 1 | 1 | 200 | 5 | 1500 | 0 | 0 | 0 |

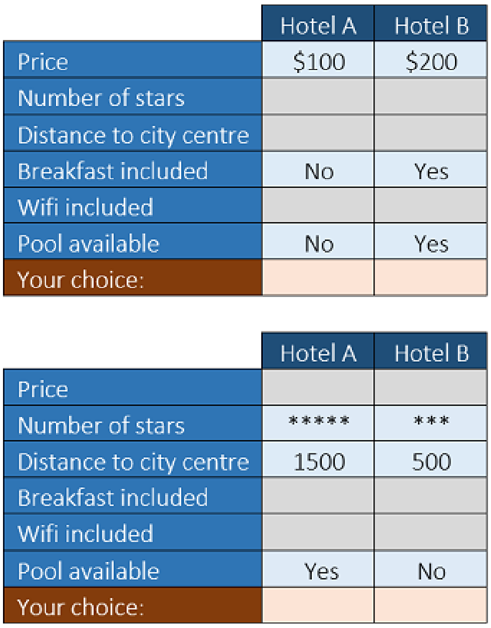
**Figure 8.24: Candidate set with choice tasks consisting of explicit partial profiles**



| | Hotel A | | | | | | Hotel B | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| s | Price | Stars | Distance | Breakfast | Wifi | Pool | Price | Stars | Distance | Breakfast | Wifi | Pool |
| 1 | 200 | 5 | 500 | 1 | 0 | 0 | 200 | 1 | 1500 | 0 | 0 | 0 |
| 2 | 100 | 1 | 1500 | 1 | 1 | 0 | 200 | 1 | 1500 | 1 | 0 | 1 |
| 3 | 100 | 5 | 500 | 1 | 1 | 1 | 100 | 1 | 500 | 1 | 0 | 0 |
| 4 | 200 | 5 | 1500 | 0 | 1 | 1 | 200 | 1 | 500 | 0 | 1 | 0 |
| 5 | 200 | 5 | 500 | 0 | 1 | 1 | 100 | 5 | 500 | 1 | 1 | 0 |
| 6 | 100 | 1 | 1500 | 1 | 1 | 1 | 200 | 1 | 500 | 0 | 1 | 1 |
| 7 | 200 | 5 | 500 | 1 | 0 | 1 | 100 | 5 | 1500 | 1 | 0 | 0 |
| 8 | 100 | 1 | 1500 | 1 | 1 | 1 | 100 | 5 | 500 | 1 | 0 | 1 |
| 9 | 200 | 1 | 500 | 0 | 0 | 1 | 200 | 5 | 500 | 0 | 1 | 0 |
| 10 | 200 | 5 | 1500 | 1 | 0 | 0 | 100 | 5 | 1500 | 0 | 0 | 1 |
| 11 | 100 | 5 | 500 | 0 | 1 | 1 | 100 | 1 | 500 | 1 | 0 | 1 |
| 12 | 200 | 1 | 500 | 1 | 1 | 1 | 100 | 1 | 500 | 0 | 0 | 1 |

**Figure 8.25: Efficient experimental design with explicit partial profiles**

If the experiment is unlabelled (i.e. does not contain labelled alternatives, including ‘no choice’ alternatives like `neither`) and only contains main effects (i.e., no interaction effects such as `price * stars`), then it is possible to further reduce choice task complexity by showing *implicit* partial profiles instead of explicit partial profiles. In an implicit partial profile design only a subset of attributes is shown in each choice task, while varying this subset over choice tasks. Figure 8.26 shows an example in which levels for only three out of the six attributes are shown in each choice task, with the implicit assumption that the levels of the attributes not shown are the same across all alternatives (without actually showing the levels of these attributes).



| | Hotel A | Hotel B |
|---|---|---|
| Price | $100 | $200 |
| Number of stars | | |
| Distance to city centre | | |
| Breakfast included | No | Yes |
| Wifi included | | |
| Pool available | No | Yes |
| Your choice: | | |

| | Hotel A | Hotel B |
|---|---|---|
| Price | | |
| Number of stars | ***** | *** |
| Distance to city centre | 1500 | 500 |
| Breakfast included | | |
| Wifi included | | |
| Pool available | Yes | No |
| Your choice: | | |

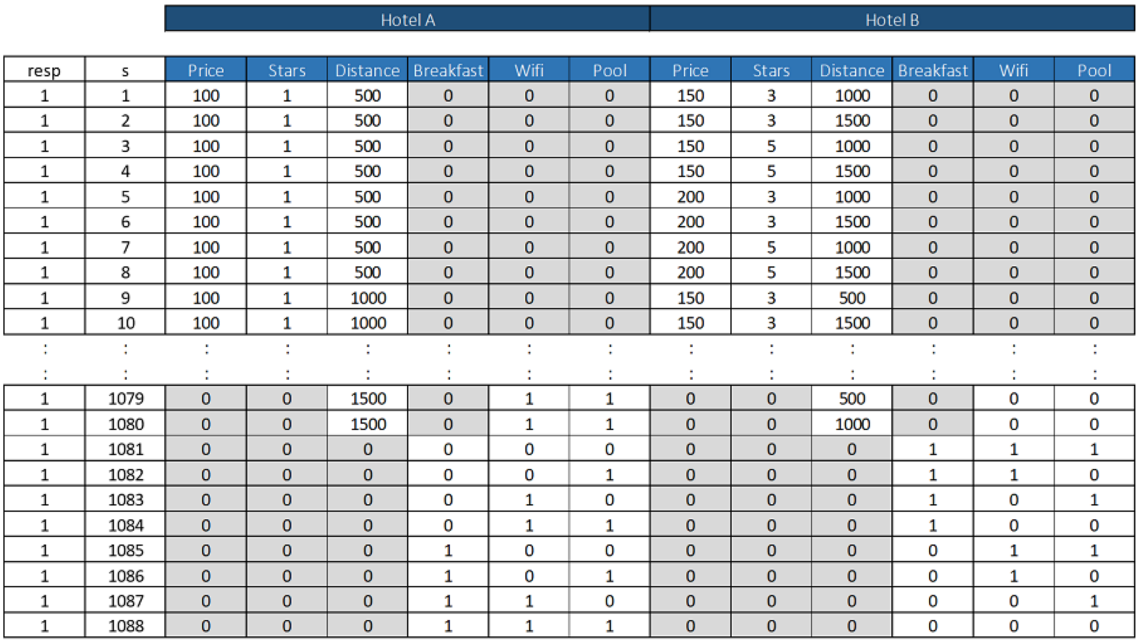
**Figure 8.26: Choice tasks with implicit partial profiles showing only three attributes per choice task**

The following syntax (where we removed the labelled alternative `neither` as well as the interaction effect `price * stars` from the previous syntax) generates an implicit partial profile design:

```
Design
;alts = hotelA*, hotelB*
;rows = 12
;eff = (mnl,d)
;alg = mfederov(candidates = implicit_partial_profiles.csv)
;model:
U(hotelA)  = b1[-0.01]      * price[100,150,200,0]       +
             b2[-0.2]       * stars[1,3,5,0]             +
             b3[-0.001]     * distance[500,1000,1500,0]  +
             b4.dummy[0.3]  * wifi[1,0]                  +
             b5.dummy[0.2]  * breakfast[1,0]             +
             b6.dummy[0.2]  * pool[1,0]                  /
U(hotelB)  = b1             * price                      +
             b2             * stars                      +
             b3             * distance                   +
             b4             * wifi                       +
             b5             * breakfast                  +
             b6             * pool
$
```

In the syntax above, we read in an external candidate set called `implicit_partial_profiles.csv` that is created by the analyst (e.g. in Excel, R, or Matlab) that consists of implicit partial profiles in which a subset of attributes are omitted in each choice task. Attributes can be omitted by setting their level to zero. Figure 8.27 illustrates what such a candidate set may look like. In this example, we omit three out of the six attributes in each choice task (shaded in grey), and we ensure that the remaining attributes are non-overlapping. This leads to in total 1,088 possible choice tasks. This set is sufficiently small such that we can include this entire candidate set in the modified Federov algorithm, but if desired an incomplete candidate set can be generated with a subset of all possible combinations. In general, the total number of possible implicit partial profiles is much smaller than the total number of possible explicit partial profiles.

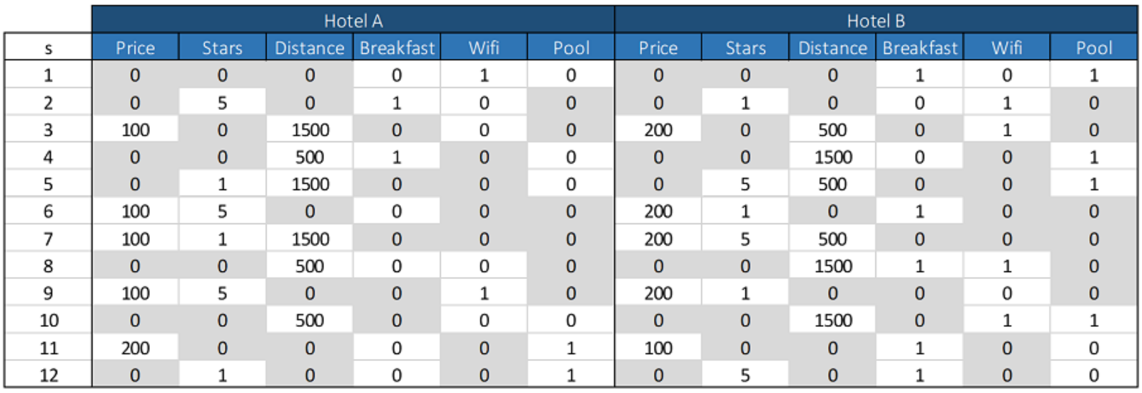
In order to ensure that the attribute levels in the candidate set match the levels in the syntax, it is important to make sure that 0 is one of the feasible levels of each attribute. In the above syntax, a zero is added to the attribute levels of price, stars, and distance, while zeros for the dummy coded variables are already present.



| | | Hotel A | | | | | | Hotel B | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| resp | s | Price | Stars | Distance | Breakfast | Wifi | Pool | Price | Stars | Distance | Breakfast | Wifi | Pool |
| 1 | 1 | 100 | 1 | 500 | 0 | 0 | 0 | 150 | 3 | 1000 | 0 | 0 | 0 |
| 1 | 2 | 100 | 1 | 500 | 0 | 0 | 0 | 150 | 3 | 1500 | 0 | 0 | 0 |
| 1 | 3 | 100 | 1 | 500 | 0 | 0 | 0 | 150 | 5 | 1000 | 0 | 0 | 0 |
| 1 | 4 | 100 | 1 | 500 | 0 | 0 | 0 | 150 | 5 | 1500 | 0 | 0 | 0 |
| 1 | 5 | 100 | 1 | 500 | 0 | 0 | 0 | 200 | 3 | 1000 | 0 | 0 | 0 |
| 1 | 6 | 100 | 1 | 500 | 0 | 0 | 0 | 200 | 3 | 1500 | 0 | 0 | 0 |
| 1 | 7 | 100 | 1 | 500 | 0 | 0 | 0 | 200 | 5 | 1000 | 0 | 0 | 0 |
| 1 | 8 | 100 | 1 | 500 | 0 | 0 | 0 | 200 | 5 | 1500 | 0 | 0 | 0 |
| 1 | 9 | 100 | 1 | 1000 | 0 | 0 | 0 | 150 | 3 | 500 | 0 | 0 | 0 |
| 1 | 10 | 100 | 1 | 1000 | 0 | 0 | 0 | 150 | 3 | 1500 | 0 | 0 | 0 |
| : | : | : | : | : | : | : | : | : | : | : | : | : | : |
| : | : | : | : | : | : | : | : | : | : | : | : | : | : |
| 1 | 1079 | 0 | 0 | 1500 | 0 | 1 | 1 | 0 | 0 | 500 | 0 | 0 | 0 |
| 1 | 1080 | 0 | 0 | 1500 | 0 | 1 | 1 | 0 | 0 | 1000 | 0 | 0 | 0 |
| 1 | 1081 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1082 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1083 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1084 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1085 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1086 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1087 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1088 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 8.27: Complete candidate set with choice tasks consisting of implicit partial profiles**

A resulting efficient experimental design generated by Ngene is shown in Figure 8.28 in which in each choice task three attributes are shown to the respondent, while the other three attributes are omitted (shaded in grey).



| | Hotel A | | | | | | Hotel B | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| s | Price | Stars | Distance | Breakfast | Wifi | Pool | Price | Stars | Distance | Breakfast | Wifi | Pool |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 2 | 0 | 5 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 100 | 0 | 1500 | 0 | 0 | 0 | 200 | 0 | 500 | 0 | 1 | 0 |
| 4 | 0 | 0 | 500 | 1 | 0 | 0 | 0 | 0 | 1500 | 0 | 0 | 1 |
| 5 | 0 | 1 | 1500 | 0 | 0 | 0 | 0 | 5 | 500 | 0 | 0 | 1 |
| 6 | 100 | 5 | 0 | 0 | 0 | 0 | 200 | 1 | 0 | 1 | 0 | 0 |
| 7 | 100 | 1 | 1500 | 0 | 0 | 0 | 200 | 5 | 500 | 0 | 0 | 0 |
| 8 | 0 | 0 | 500 | 0 | 0 | 0 | 0 | 0 | 1500 | 1 | 1 | 0 |
| 9 | 100 | 5 | 0 | 0 | 1 | 0 | 200 | 1 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 500 | 0 | 0 | 0 | 0 | 0 | 1500 | 0 | 1 | 1 |
| 11 | 200 | 0 | 0 | 0 | 0 | 1 | 100 | 0 | 0 | 1 | 0 | 0 |
| 12 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 5 | 0 | 1 | 0 | 0 |

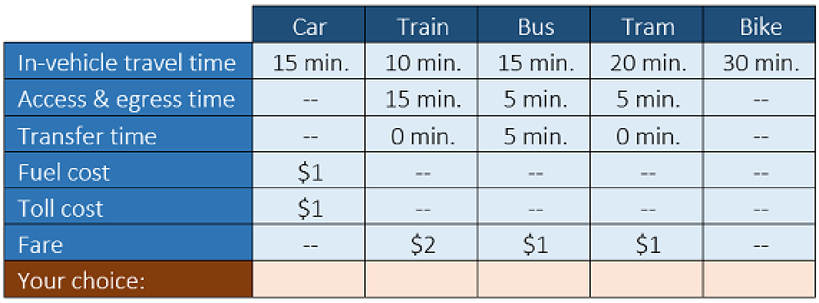
**Figure 8.28: Efficient experimental design with implicit partial profiles**

## 8.11 Showing only a subset of alternatives in each choice task

In labelled experiments the number of alternatives may be large, such that choice tasks can be complex. For example, with five alternatives each with four attributes the respondents are asked to evaluate 20 attribute levels in each choice task. Such complex choice tasks lead to a significant cognitive burden on the respondent, which often means that a respondent adopts heuristic rules in order to simplify the task and may not trade off on all alternatives and attributes. Therefore, the analyst may wish to only show a subset of alternatives in each choice task. We call such a design a *partial choice set design*, and is sometimes also called an *availability design*, referring to the availability of alternatives in choice tasks.

As an example, consider the syntax below that generates a design with five labelled alternatives (car, train, bus, tram, and bike). An example choice task is shown in Figure 8.29.

```
Design
;alts = car, train, bus, tram, bike
;rows = 15
;eff = (mnl,d)
;model:
U(car)   = b1[0.3]                             +
           b2[-0.05]  * ctime[15,20,25]        +
           b3[-0.3]   * fuel[1,2]              +
           b4[-0.4]   * toll[0,1]              /
U(train) = b5[0.2]                             +
           b6[-0.04]  * ttime[10,15,20]        +
           b7[-0.08]  * accegg[5,10,15]        +
           b8[-0.08]  * transfer[0,5,10]       +
           b9[-0.3]   * fare[2,3]              /
U(bus)   = b10[-0.2]                           +
           b11[-0.06] * btime[15,20,25]        +
           b7         * accegg                 +
           b8         * transfer               +
           b9         * fare2[1,2]             /
U(tram)  = b12[0.1]                            +
           b6         * ttime                  +
           b7         * accegg                 +
           b8         * transfer               +
           b9         * fare2                  /
U(bike)  = b13[-0.08] * biketime[20,30,40]
$
```

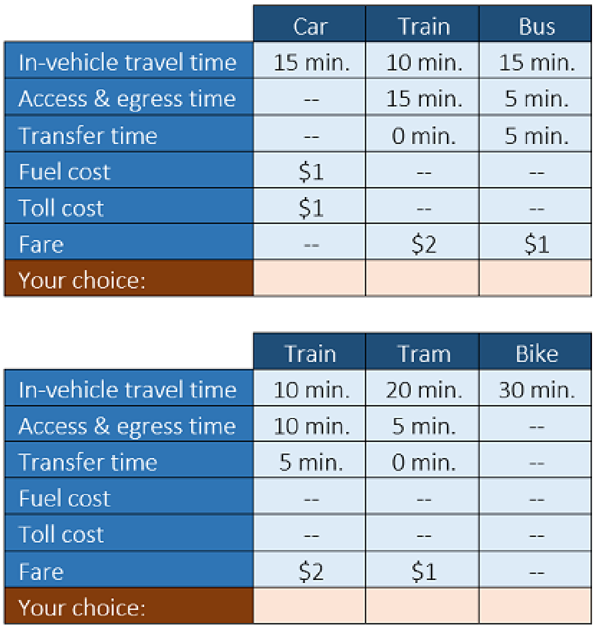


| | Car | Train | Bus | Tram | Bike |
|---|---|---|---|---|---|
| In-vehicle travel time | 15 min. | 10 min. | 15 min. | 20 min. | 30 min. |
| Access & egress time | -- | 15 min. | 5 min. | 5 min. | -- |
| Transfer time | -- | 0 min. | 5 min. | 0 min. | -- |
| Fuel cost | $1 | -- | -- | -- | -- |
| Toll cost | $1 | -- | -- | -- | -- |
| Fare | -- | $2 | $1 | $1 | -- |
| Your choice: | | | | | |

**Figure 8.29: Choice task with a full choice set**

Instead of showing all five alternatives in each choice task, in order to reduce choice task complexity one may wish to show only three alternatives in each choice task, as illustrated in

Figure 8.30.



| | Car | Train | Bus |
|---|---|---|---|
| In-vehicle travel time | 15 min. | 10 min. | 15 min. |
| Access & egress time | -- | 15 min. | 5 min. |
| Transfer time | -- | 0 min. | 5 min. |
| Fuel cost | $1 | -- | -- |
| Toll cost | $1 | -- | -- |
| Fare | -- | $2 | $1 |
| Your choice: | | | |

| | Train | Tram | Bike |
|---|---|---|---|
| In-vehicle travel time | 10 min. | 20 min. | 30 min. |
| Access & egress time | 10 min. | 5 min. | -- |
| Transfer time | 5 min. | 0 min. | -- |
| Fuel cost | -- | -- | -- |
| Toll cost | -- | -- | -- |
| Fare | $2 | $1 | -- |
| Your choice: | | | |

**Figure 8.30: Choice tasks with partial choice sets**

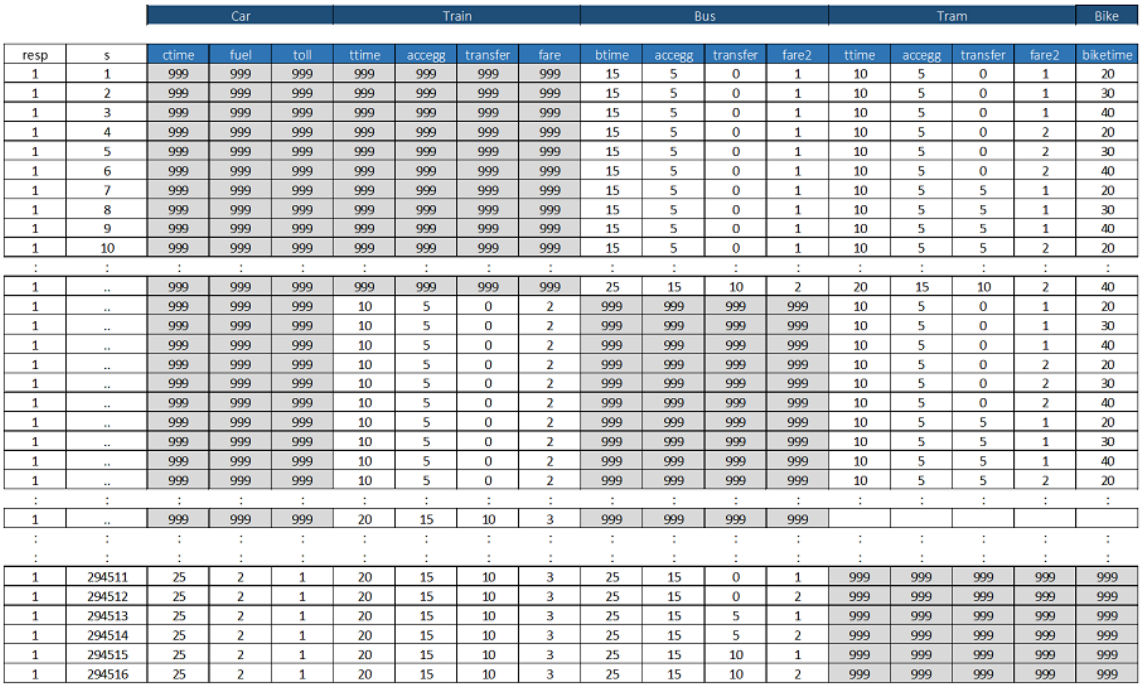
Although it is not possible to let Ngene directly determine which alternatives to show in each choice task, Ngene can indirectly select and optimise the subset of alternatives to show in each choice task by using the modified Federov algorithm together with a specifically constructed candidate set as shown in Figure 8.31. This candidate set (`partial_choice_sets.csv`) is externally created by the analyst (e.g., in Excel, R, or Matlab). In this example we aim to omit two alternatives from each choice task by only showing a subset of three alternatives. Alternatives can be omitted by assigning a large negative utility to it. This can be achieved by setting large positive levels for attributes with a negative prior, or large negative levels for attributes with a positive prior. In our syntax example, all priors are negative and hence we choose a positive large value 999 for all attributes. In the candidate set in Figure 8.31, the first choice task omits alternatives car and train, such that only bus, tram, and bike are shown to the respondent (other alternatives are shaded in grey for illustration purposes, although this formatting should not occur in the actual file). Note that the format of the external candidate set is the same as any other design that Ngene can read in for evaluation using *;eval* (see Section 8.7). The format shown in Figure 8.31 has a header row that is ignored by Ngene, a first column indicating that we are generating a homogeneous design for a single respondent, a second column with the choice task number, and all next columns are for attributes in the order in which they appear within the syntax.

In order to make the syntax consistent with this candidate set, we need to add 999 as a level to each attribute. The following syntax can then be used to create a partial choice set design.

```
Design
;alts = car, train, bus, tram, bike
;rows = 15
;eff = (mnl,d)
;alg = mfederov(candidates = partial_choice_sets.csv)
;model:
U(car)   = b1[0.3]                             +
           b2[-0.05]  * ctime[15,20,25,999]    +
           b3[-0.3]   * fuel[1,2,999]          +
           b4[-0.4]   * toll[0,1,999]          /
U(train) = b5[0.2]                             +
           b6[-0.04]  * ttime[10,15,20,999]    +
           b7[-0.08]  * accegg[5,10,15,999]    +
           b8[-0.08]  * transfer[0,5,10,999]   +
           b9[-0.3]   * fare[2,3,999]          /
U(bus)   = b10[-0.2]                           +
           b11[-0.06] * btime[15,20,25,999]    +
           b7         * accegg                 +
           b8         * transfer               +
           b9         * fare2[1,2,999]         /
U(tram)  = b12[0.1]                            +
           b6         * ttime                  +
           b7         * accegg                 +
           b8         * transfer               +
           b9         * fare2                  /
U(bike)  = b13[-0.08] * biketime[20,30,40,999]
$
```

Instead of adding a large value to each attribute, it is sufficient to add it to only one linear coded attribute in each alternative (e.g. to **ctime**, **ttime**, **btime**, and **biketime**). If at least one attribute of an alternative has a very large negative utility, then the choice probability of this alternative becomes zero and the levels of the other attributes become irrelevant (i.e., can be set to any value). Since adding a large level to a dummy or effects coded variable leads to inconsistencies, the procedure described in this section does not work if the utility functions only consist of dummy and/or effects coded variables. In the case priors are unknown, one can use a small (positive or negative) prior value, and use an appropriate (positive or negative) large attribute levels to indicate an omitted alternative. For example, in case we only know that the coefficient of **ctime** is negative, then we could use **b2[-0.00001]*ctime [15,20,25,9999999]**.

The syntax above can include constraints on attribute levels set by *;reject* or *;require* commands, which automatically removes invalid choice tasks from the candidate set. Further note that the modified Federov relaxes the attribute level balance constraint. If some degree of attribute level balance is required in the design one may wish to set certain upper and lower limits on the number of times each attribute level appears within the design, or alternatively impose a soft level balance constraint, see Section 8.1.

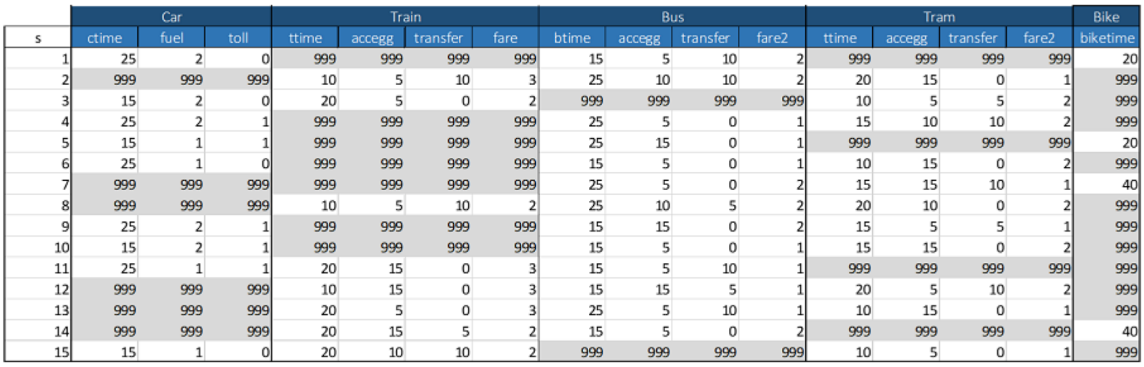


| | | Car | | | Train | | | | Bus | | | | Tram | | | | Bike |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| resp | s | ctime | fuel | toll | ttime | accegg | transfer | fare | btime | accegg | transfer | fare2 | ttime | accegg | transfer | fare2 | biketime |
| 1 | 1 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 15 | 5 | 0 | 1 | 10 | 5 | 0 | 1 | 20 |
| 1 | 2 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 15 | 5 | 0 | 1 | 10 | 5 | 0 | 1 | 30 |
| 1 | 3 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 15 | 5 | 0 | 1 | 10 | 5 | 0 | 1 | 40 |
| 1 | 4 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 15 | 5 | 0 | 1 | 10 | 5 | 0 | 2 | 20 |
| 1 | 5 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 15 | 5 | 0 | 1 | 10 | 5 | 0 | 2 | 30 |
| 1 | 6 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 15 | 5 | 0 | 1 | 10 | 5 | 0 | 2 | 40 |
| 1 | 7 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 15 | 5 | 0 | 1 | 10 | 5 | 5 | 1 | 20 |
| 1 | 8 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 15 | 5 | 0 | 1 | 10 | 5 | 5 | 1 | 30 |
| 1 | 9 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 15 | 5 | 0 | 1 | 10 | 5 | 5 | 1 | 40 |
| 1 | 10 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 15 | 5 | 0 | 1 | 10 | 5 | 5 | 2 | 20 |
| : | : | : | : | : | : | : | : | : | : | : | : | : | : | : | : | : | : |
| 1 | .. | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 25 | 15 | 10 | 2 | 20 | 15 | 10 | 2 | 40 |
| 1 | .. | 999 | 999 | 999 | 10 | 5 | 0 | 2 | 999 | 999 | 999 | 999 | 10 | 5 | 0 | 1 | 20 |
| 1 | .. | 999 | 999 | 999 | 10 | 5 | 0 | 2 | 999 | 999 | 999 | 999 | 10 | 5 | 0 | 1 | 30 |
| 1 | .. | 999 | 999 | 999 | 10 | 5 | 0 | 2 | 999 | 999 | 999 | 999 | 10 | 5 | 0 | 1 | 40 |
| 1 | .. | 999 | 999 | 999 | 10 | 5 | 0 | 2 | 999 | 999 | 999 | 999 | 10 | 5 | 0 | 2 | 20 |
| 1 | .. | 999 | 999 | 999 | 10 | 5 | 0 | 2 | 999 | 999 | 999 | 999 | 10 | 5 | 0 | 2 | 30 |
| 1 | .. | 999 | 999 | 999 | 10 | 5 | 0 | 2 | 999 | 999 | 999 | 999 | 10 | 5 | 0 | 2 | 40 |
| 1 | .. | 999 | 999 | 999 | 10 | 5 | 0 | 2 | 999 | 999 | 999 | 999 | 10 | 5 | 5 | 1 | 20 |
| 1 | .. | 999 | 999 | 999 | 10 | 5 | 0 | 2 | 999 | 999 | 999 | 999 | 10 | 5 | 5 | 1 | 30 |
| 1 | .. | 999 | 999 | 999 | 10 | 5 | 0 | 2 | 999 | 999 | 999 | 999 | 10 | 5 | 5 | 1 | 40 |
| 1 | .. | 999 | 999 | 999 | 10 | 5 | 0 | 2 | 999 | 999 | 999 | 999 | 10 | 5 | 5 | 2 | 20 |
| : | : | : | : | : | : | : | : | : | : | : | : | : | : | : | : | : | : |
| 1 | .. | 999 | 999 | 999 | 20 | 15 | 10 | 3 | 999 | 999 | 999 | 999 | | | | | |
| : | : | : | : | : | : | : | : | : | : | : | : | : | : | : | : | : | : |
| : | : | : | : | : | : | : | : | : | : | : | : | : | : | : | : | : | : |
| 1 | 294511 | 25 | 2 | 1 | 20 | 15 | 10 | 3 | 25 | 15 | 0 | 1 | 999 | 999 | 999 | 999 | 999 |
| 1 | 294512 | 25 | 2 | 1 | 20 | 15 | 10 | 3 | 25 | 15 | 0 | 2 | 999 | 999 | 999 | 999 | 999 |
| 1 | 294513 | 25 | 2 | 1 | 20 | 15 | 10 | 3 | 25 | 15 | 5 | 1 | 999 | 999 | 999 | 999 | 999 |
| 1 | 294514 | 25 | 2 | 1 | 20 | 15 | 10 | 3 | 25 | 15 | 5 | 2 | 999 | 999 | 999 | 999 | 999 |
| 1 | 294515 | 25 | 2 | 1 | 20 | 15 | 10 | 3 | 25 | 15 | 10 | 1 | 999 | 999 | 999 | 999 | 999 |
| 1 | 294516 | 25 | 2 | 1 | 20 | 15 | 10 | 3 | 25 | 15 | 10 | 2 | 999 | 999 | 999 | 999 | 999 |

**Figure 8.31: Candidate set with choice tasks consisting of partial choice sets**

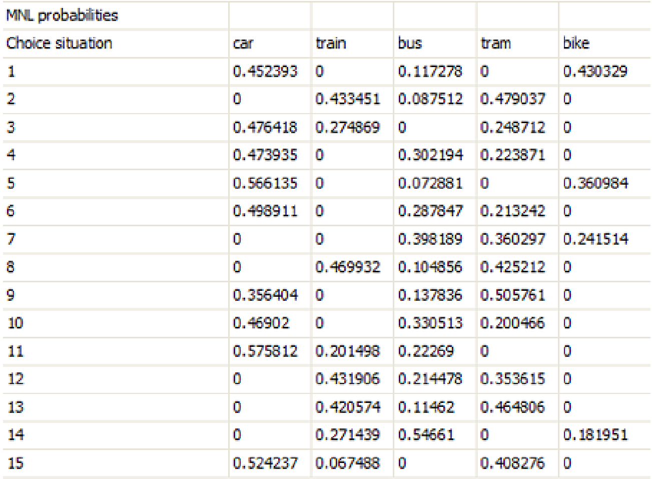
The complete candidate set may be very large, therefore one may wish to select a subset of rows. For example, in our example there exist 294,516 possible choice tasks in which only three out of five alternatives are shown to a respondent. In this case we suggest using an incomplete candidate set with a random selection of, say, 1 per cent of all possible choice tasks.

A resulting efficient experimental design generated by Ngene is shown in Figure 8.32 in which in each choice task three alternatives are shown to the respondent, while the other two alternatives are omitted (shaded in grey for convenience). The corresponding choice probabilities calculated by Ngene as shown in Figure 8.33 confirm that in each choice task two alternatives are omitted since they have a zero probability of being chosen, while the other choice probabilities sum to one. Note that that the modified Federov algorithm cannot guarantee attribute level balance, hence, whilst not strictly necessary, one may wish to add hard constraints on the number of times each attribute level appears within the design, or alternatively impose a soft level balance constraint, see Section 8.1.



| | Car | | | Train | | | | Bus | | | | Tram | | | | Bike |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| s | ctime | fuel | toll | ttime | accegg | transfer | fare | btime | accegg | transfer | fare2 | ttime | accegg | transfer | fare2 | biketime |
| 1 | 25 | 2 | 0 | 999 | 999 | 999 | 999 | 15 | 5 | 10 | 2 | 999 | 999 | 999 | 999 | 20 |
| 2 | 999 | 999 | 999 | 10 | 5 | 10 | 3 | 25 | 10 | 10 | 2 | 20 | 15 | 0 | 1 | 999 |
| 3 | 15 | 2 | 0 | 20 | 5 | 0 | 2 | 999 | 999 | 999 | 999 | 10 | 5 | 5 | 2 | 999 |
| 4 | 25 | 2 | 1 | 999 | 999 | 999 | 999 | 25 | 5 | 0 | 1 | 15 | 10 | 10 | 2 | 999 |
| 5 | 15 | 1 | 1 | 999 | 999 | 999 | 999 | 25 | 15 | 0 | 1 | 999 | 999 | 999 | 999 | 20 |
| 6 | 25 | 1 | 0 | 999 | 999 | 999 | 999 | 15 | 5 | 0 | 1 | 10 | 15 | 0 | 2 | 999 |
| 7 | 999 | 999 | 999 | 999 | 999 | 999 | 999 | 25 | 5 | 0 | 2 | 15 | 15 | 10 | 1 | 40 |
| 8 | 999 | 999 | 999 | 10 | 5 | 10 | 2 | 25 | 10 | 5 | 2 | 20 | 10 | 0 | 2 | 999 |
| 9 | 25 | 2 | 1 | 999 | 999 | 999 | 999 | 15 | 15 | 0 | 2 | 15 | 5 | 5 | 1 | 999 |
| 10 | 15 | 2 | 1 | 999 | 999 | 999 | 999 | 15 | 5 | 0 | 1 | 15 | 15 | 0 | 2 | 999 |
| 11 | 25 | 1 | 1 | 20 | 15 | 0 | 3 | 15 | 5 | 10 | 1 | 999 | 999 | 999 | 999 | 999 |
| 12 | 999 | 999 | 999 | 10 | 15 | 0 | 3 | 15 | 15 | 5 | 1 | 20 | 5 | 10 | 2 | 999 |
| 13 | 999 | 999 | 999 | 20 | 5 | 0 | 3 | 25 | 5 | 10 | 1 | 10 | 15 | 0 | 1 | 999 |
| 14 | 999 | 999 | 999 | 20 | 15 | 5 | 2 | 15 | 5 | 0 | 2 | 999 | 999 | 999 | 999 | 40 |
| 15 | 15 | 1 | 0 | 20 | 10 | 10 | 2 | 999 | 999 | 999 | 999 | 10 | 5 | 0 | 1 | 999 |

**Figure 8.32: Efficient experimental design with partial choice sets**



| MNL probabilities | | | | | |
|---|---|---|---|---|---|
| Choice situation | car | train | bus | tram | bike |
| 1 | 0.452393 | 0 | 0.117278 | 0 | 0.430329 |
| 2 | 0 | 0.433451 | 0.087512 | 0.479037 | 0 |
| 3 | 0.476418 | 0.274869 | 0 | 0.248712 | 0 |
| 4 | 0.473935 | 0 | 0.302194 | 0.223871 | 0 |
| 5 | 0.566135 | 0 | 0.072881 | 0 | 0.360984 |
| 6 | 0.498911 | 0 | 0.287847 | 0.213242 | 0 |
| 7 | 0 | 0 | 0.398189 | 0.360297 | 0.241514 |
| 8 | 0 | 0.469932 | 0.104856 | 0.425212 | 0 |
| 9 | 0.356404 | 0 | 0.137836 | 0.505761 | 0 |
| 10 | 0.46902 | 0 | 0.330513 | 0.200466 | 0 |
| 11 | 0.575812 | 0.201498 | 0.22269 | 0 | 0 |
| 12 | 0 | 0.431906 | 0.214478 | 0.353615 | 0 |
| 13 | 0 | 0.420574 | 0.11462 | 0.464806 | 0 |
| 14 | 0 | 0.271439 | 0.54661 | 0 | 0.181951 |
| 15 | 0.524237 | 0.067488 | 0 | 0.408276 | 0 |

**Figure 8.33: Choice probabilities for partial choice set design**

# Chapter 9

# Designs With Continuous Attribute Levels

# 9 Designs With Continuous Attribute Levels

## 9.1 Theory of designs with continuous levels

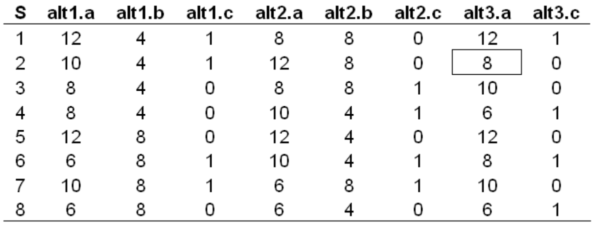
Simply put, any constraint one places on a design will impact upon the overall efficiency of that design (where efficiency is defined in the terms outlined in Chapter 7). Orthogonality, as traditionally viewed within the literature (see Chapter 6), represents one such constraint. A second constraint often imposed on designs is attribute level balance. Attribute level balance occurs when each level of an attribute is forced to occur an equal number of times in the design. This constraint is imposed so that each point in preference space (represented by the attribute levels) is covered an equal number of times. The attribute level balance constraint is often imposed on efficient designs, although this need not be the case. Typically, when this constraint is relaxed, a minimum number of times each level must appear is imposed, otherwise the levels of the design will tend to all go to the extremes of the attribute level range, thus not allowing for tests of non-linearity in preference (e.g., see Section 8.1).

Where such a constraint is maintained, the overall efficiency of a design may be impacted upon as changing one attribute level in one choice situation may result in an overall improvement in the design, but such a change would require that another attribute level be changed somewhere else in the design, possibly resulting in an overall worsening of overall level of efficiency of the design. For example, consider an efficient design constructed using the following syntax.

```
design
;alts = alt1, alt2, alt3
;rows = 8
;eff = (mnl,d,fixed)
;con
;model:
U(alt1) = b1[1.2] + b2[-0.6]*A[6,8,10,12] + b3[-0.4]*B[4,8] + b4[0.3] *C
[0,1] /
U(alt2) = b5[0.6] + b2         *A         + b3     *B     + b6[0.8] *C
      /
U(alt3) =           b2         *A         + b7[-1.0]*C
     $
```

Table 9.1 presents an efficient design generated based on the above syntax. The overall $D_p$-error of the design is 0.799. In Table 9.1, we have highlighted the attribute level for the first attribute for alternative 3 in choice situation 2. Keeping the remainder of the design fixed, if we change this attribute level from a value of 8 to a value of 10, the $D_p$-error of the design will improve to 0.789, however in doing this, this attribute will no longer exhibit the attribute level balance property (10 would now appear three times whilst 8 would now appear only once over the eight choice situations). As such, to maintain attribute level balance, we would be required to change one of the already existing attribute levels of 10 to a value of 8. If we change the level 10 in choice situation three, then the overall $D_p$-error of the design will worsen to a value of 0.820. If we change the level 10 in choice situation seven to 8, then the overall $D_p$-error of the design worsens to 0.829. Thus, whilst changing the original value led to an overall improvement in the efficiency of the design, the attribute level balance property, which requires us to change another level in another choice situation somewhere else in the design, prevents us from maintaining this gain, and in fact, results in a worsening in the designs statistical efficiency. As such, we would prefer the existing design shown in Table 9.1 to one where we swap the attribute levels as discussed above.

**Table 9.1: Attribute level balance and efficient designs**



| S | alt1.a | alt1.b | alt1.c | alt2.a | alt2.b | alt2.c | alt3.a | alt3.c |
|---|---|---|---|---|---|---|---|---|
| 1 | 12 | 4 | 1 | 8 | 8 | 0 | 12 | 1 |
| 2 | 10 | 4 | 1 | 12 | 8 | 0 | 8 | 0 |
| 3 | 8 | 4 | 0 | 8 | 8 | 1 | 10 | 0 |
| 4 | 8 | 4 | 0 | 10 | 4 | 1 | 6 | 1 |
| 5 | 12 | 8 | 0 | 12 | 4 | 0 | 12 | 0 |
| 6 | 6 | 8 | 1 | 10 | 4 | 1 | 8 | 1 |
| 7 | 10 | 8 | 1 | 6 | 8 | 1 | 10 | 0 |
| 8 | 6 | 8 | 0 | 6 | 4 | 0 | 6 | 1 |

Toner *et al*. (1999), Fowkes (2000) and Kanninen (2002) offer a number of different design methods which we collectively call *optimal choice probability* designs that are designed to overcome this problem. Both Toner et al. and Kanninen show analytically that utility or probability balance in choice tasks represent an undesirable property, and in doing so suggest rules that minimize the variance of estimates in an optimal manner, based on desirable or what Toner et al. refer to as magic p's. Although using a different set of arguments, Fowkes (2000) arrived at a similar conclusion deriving a set of designs he termed boundary value designs. In each case, *K*-1 attribute levels are first generated for each *J* alternatives, typically using an orthogonal or optimal orthogonal approach. The last K$^{th}$ attribute for each alternative is then generated as a continuous variable (usually a price attribute). The values of these continuous variables are chosen such that the choice probabilities take certain values that minimize the elements of the AVC matrix under the assumption of non-zero prior parameters. Toner et al. (1999) achieves a similar result to those reported by Kanninen and Fowkes. The boundary value method of Fowkes is somewhat different in derivation although the implications remain the same. Toner et al. (1999), Kanninen (2002) and Johnson et al. (2006) have determined the desirable probabilities for a limited number of designs (i.e., those involving two alternatives), although non-linear programming may be used to determine these for a wider number of designs. The boundary value method of Fowkes is somewhat different in derivation although the implications remain the same. Appendix 9A outlines the steps required for generating this form of design. We now discuss how to generate these designs in Ngene. In all cases however, prior parameters are still required to generate this class of designs.

## 9.2 Designs with continuous levels in Ngene

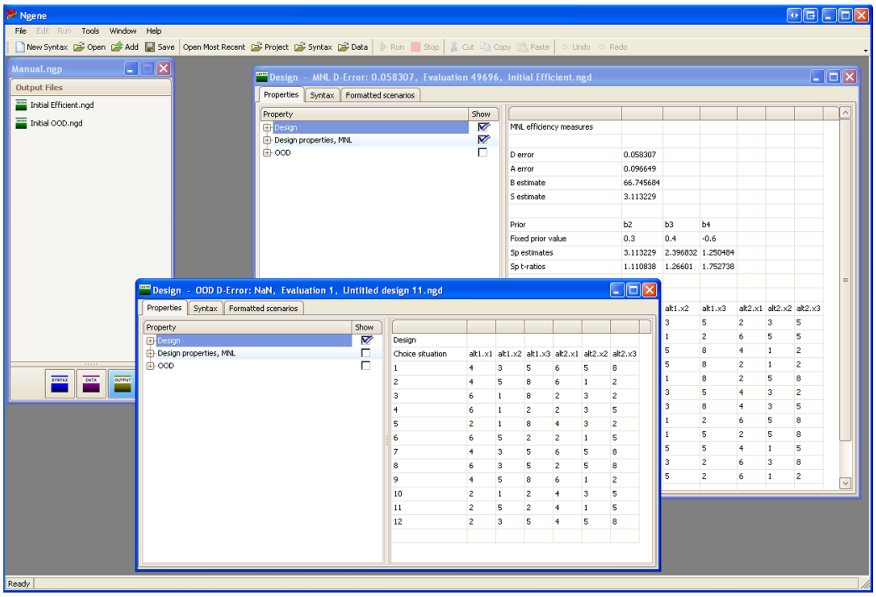
In order to generate an *optimal choice probability* design, the first step is to generate a design with non-continuous attribute levels. This initial design should have the same number of design dimensions (i.e., alternatives, attributes, attribute levels and choice situations) with the exception of the *K*$^{th}$ attribute which is to be treated as continuous. For this attribute, any the levels can be provided as long as they do not violate attribute level balance and hence require a different number of rows be generated. For example, assuming the price attribute as the attribute to be later treated as continuous, assigning it two attribute levels for a design to be generated in nine rows will require a change in the number of rows required. Also, whilst not necessary, the specific levels chosen are best selected if they are within the range that will be allowed when the attribute is later treated as continuous. For example, if in the final design, the analyst will allow the price attribute to take any value between $0 and $20, then the attribute levels for price in the initial design should be within this range also. In generating the initial design, any type of design can be constructed. Note that, in generating the design with continuous attribute levels, only the attributes that are allowed to take on continuous levels will be changed. That is, all other attributes will be fixed based on the initial design. Kanninen (2002) and Johnson *et al*. (2006) suggest using optimal orthogonal designs as the initial start design, however other design types might provide more efficient results, particularly if they are closer to the 'optimal' level of statistical efficiency.

To demonstrate, consider the following two sets of syntax used to generate potential start designs. The first generates an optimal orthogonal design whilst the second creates an efficient design. In specifying the optimal orthogonal design, no priors are required, whilst priors are required for the efficient design. We will use both to construct initial start designs using both sets of syntax.

```
Design
;alts = Alt1, Alt2
;rows = 12
;orth = ood
;model:
U(Alt1) = b1 * X1[2,4,6] + b2 * X2[1,3,5] + b3 * X3[2,5,8] /
U(Alt2) = b1 * X1        + b2 * X2        + b3 * X3
$
```

```
Design
;alts = Alt1, Alt2
;rows = 12
;eff = (mnl, d)
;model:
U(Alt1)  = b1[-0.2]  + b2[0.3]  * X1[2,4,6]  + b3[0.4]  * X2[1,3,5]  + b4[-
0.6] * X3[2,5,8] /
U(Alt2) =            b2       * X1         + b3       * X2          + b4
  * X3
$
```

Figure 9.1 shows the two designs generated using the above syntax. Both designs have been saved as part of a project as can be seen by their appearance in the ‘Output’ tab of the project bar. The designs were saved as ‘Initial OOD.ngs’ and ‘Initial Efficient.ngs’ respectively. Although not shown, the $D_p$-error of the efficient design was 0.058 versus a $D_p$-error of 0.154 for the optimal orthogonal design based on the set or priors assumed in generating the efficient design.


Ngene
File Edit Run Tools Window Help
New Syntax Open Add Save Open Most Recent Project Syntax Data Run Stop Cut Copy Paste Undo Redo
Manual.ngp
Output Files
Initial Efficient.ngd
Initial OOD.ngd
SYNTAX DATA OUTPUT
Ready

Design - MNL D-Error: 0.058307, Evaluation 49696, Initial Efficient.ngd

Properties | Syntax | Formatted scenarios

| Property | Show |
|---|---|
| Design | ☑ |
| Design properties, MNL | ☑ |
| OOD | □ |

| | | | |
|---|---|---|---|
| MNL efficiency measures | | | |
| | | | |
| D error | 0.058307 | | |
| A error | 0.096649 | | |
| B estimate | 66.745684 | | |
| S estimate | 3.113229 | | |
| | | | |
| Prior | b2 | b3 | b4 |
| Fixed prior value | 0.3 | 0.4 | -0.6 |
| Sp estimates | 3.113229 | 2.396832 | 1.250484 |
| Sp t-ratios | 1.110838 | 1.26601 | 1.752738 |

| alt1.x2 | alt1.x3 | alt2.x1 | alt2.x2 | alt2.x3 |
|---|---|---|---|---|
| 3 | 5 | 2 | 3 | 5 |
| 1 | 2 | 6 | 5 | 5 |
| 5 | 8 | 4 | 1 | 2 |
| 5 | 8 | 2 | 1 | 2 |
| 1 | 8 | 2 | 5 | 8 |
| 3 | 5 | 4 | 3 | 2 |
| 3 | 8 | 4 | 3 | 5 |
| 1 | 2 | 6 | 5 | 8 |
| 1 | 5 | 2 | 5 | 8 |
| 5 | 5 | 4 | 1 | 5 |
| 3 | 2 | 6 | 3 | 8 |
| 5 | 2 | 6 | 1 | 2 |

Design - OOD D-Error: NaN, Evaluation 1, Untitled design 11.ngd

Properties | Syntax | Formatted scenarios

| Property | Show |
|---|---|
| Design | ☑ |
| Design properties, MNL | □ |
| OOD | □ |

| Design | | | | | | |
|---|---|---|---|---|---|---|
| Choice situation | alt1.x1 | alt1.x2 | alt1.x3 | alt2.x1 | alt2.x2 | alt2.x3 |
| 1 | 4 | 3 | 5 | 6 | 5 | 8 |
| 2 | 4 | 5 | 8 | 6 | 1 | 2 |
| 3 | 6 | 1 | 8 | 2 | 3 | 2 |
| 4 | 6 | 1 | 2 | 2 | 3 | 5 |
| 5 | 2 | 1 | 8 | 4 | 3 | 2 |
| 6 | 6 | 5 | 2 | 2 | 1 | 5 |
| 7 | 4 | 3 | 5 | 6 | 5 | 8 |
| 8 | 6 | 3 | 5 | 2 | 5 | 8 |
| 9 | 4 | 5 | 8 | 6 | 1 | 2 |
| 10 | 2 | 1 | 2 | 4 | 3 | 5 |
| 11 | 2 | 5 | 2 | 4 | 1 | 5 |
| 12 | 2 | 3 | 5 | 4 | 5 | 8 |

**Figure 9.1: Two different initial designs**

The next step in generating *optimal choice probability* designs is to construct the design, using the initial design as the start point. As discussed in Section 8.6, an already saved design can be used as the initial design when searching for a more efficient design via the start property. Thus for example, to use the already saved 'Initial OOD.ngd' design as the start design, we would specify

```
;start = initial OOD.ngd
```

Kanninen (2002) and Johnson *et al*. (2006) derived analytically, a set of probabilities that will result in an 'optimal' MNL design (i.e., the most efficient design possible, with the smallest standard errors). These derivations however apply only to designs generated for MNL models as well as for only a small subset of possible design dimensions (e.g., these probabilities are known only for designs with two alternatives with between two and eight attributes, and are limited to designs with generic parameters; see Appendix 9A). Rather than limit the type of model and dimensions of the design allowed, Ngene uses a search algorithm known as the Nelder-Mead algorithm to determine the final attribute levels for the attribute that is to be treated as continuous (see Appendix 9B). Whilst this means that the design cannot be guaranteed to be 'optimal', the resulting design should be close to optimal. In any case, the user must specify that they wish to use the Nelder-Mead algorithm when generating designs with continuous variables. This is done via the *alg* property (see Section 8.6). The syntax to do this is

```
;alg = neldermead
```

The Nelder-Mead algorithm has a number of associated parameters that may be useful in limiting the amount of output reported. Unlike efficient designs, when one allows for continuous attribute levels in a design, the number of possible designs effectively becomes infinite (e.g., an attribute might take the value 2.21421452 or 2.21421453 or 2.21421454) with very slight changes producing improvements in statistical efficiency. Specific parameters for the Nelder-Mead algorithm are listed below. Note that these are option parameters, and the syntax above will work without them.

```
neldermead(converge=<float>, runs=<integer>, nointerim, seed=<integer>)
```

where

*converge*: minimum distance required between the best and other all candidate solutions in order to terminate (default=0.001). Distance is relative to attribute level lower and upper bounds.

*runs*: the number of runs to perform (default=1). Each run is an independent trial and begins with an entirely new set of random allocations to the continuous attributes. The sole exception is the first run where one copy of the original design is maintained.

*nointerim*: only report improved designs upon convergence, not as they are found (which is default). Duplicate designs will be reported if there is no improvement between restarts.

*seed*: a number to initialize the pseudo-random number generator. This allows experiments to be repeated if so desired.

Note, that where specified, not all parameters are required. Thus for example, the analyst may specify a convergence criteria in addition to the nointerim criteria but omit the runs and seed parameters.
Note also that in addition to these parameters, stoping criteria, as reported in Section 8.6 may also be applied to the Nelder-Mead algorithm. Thus, for example the following syntax may be used.

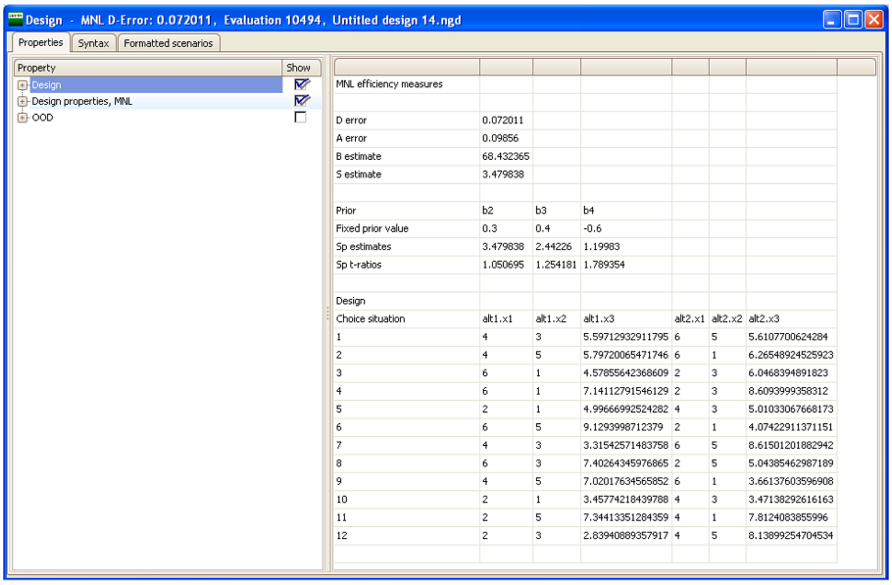
```
;alg = neldermead(nointerim=0, stop=total(5000 iterations))
```

The final syntax used to generate a design with continuous variables is handled within the utility specifications. In generating a design with continuous attribute levels, the analyst must specify which attributes are to be treated as continuous as well as place a range on the levels that these attribute may take. Traditionally, attribute levels in Ngene are specified in square brackets after the attribute name, with different levels separated by commas (e.g., `X3[2,5,8]`). Where an attribute is to be treated as a continuous variable, the analyst must specify the lower and upper values of the range separated by a colon (for example, `X3[2:10]` would allow the attribute levels of X3 to take any value, between 2 and 10). Note that adding a second colon will generate discrete attribute levels, from the lower bound to the upper bound, with a step size specified after the second colon (e.g. `[2:10:0.5]`). Complete syntax for generating a design with continuous levels is given below. In the syntax shown, the start design is given as the 'Initial OOD.ngs' design. This syntax can be easily changed to use the 'Initial Efficient.ngs' as the initial start design.

```
Design
;alts = Alt1, Alt2
;rows = 12
;eff = (mnl, d)
;alg = neldermead(nointerim=0, stop=total(5000 iterations))
;start = initial OOD.ngd
;model:
U(Alt1)  = b1[-0.2]  + b2[0.3]  * X1[2,4,6]  + b3[0.4]  * X2[1,3,5]  + b4[-
0.6] * X3[2:10] /
U(Alt2) =               b2      * X1         + b3       * X2          + b4
  * X3[2:10]
$
```

Figure 9.2 shows the resulting design based on the above output, whilst Figure 9.3 shows the resulting design based on using 'Initial Efficient.ngs' as the initial design. A comparison of these two Figures suggests that using the efficient design as the initial design resulted in a lower $D_p$-error (0.058) than the optimal orthogonal design ($D_p$-error = 0.072), hence hinting at the fact that the results may be sensitive to the initial design assumed. The primary reason for this sensitivity lays in the fact that 'optimality' is linked to the choice probabilities and by imposing too narrower a range on the values that a continuous variable might take, the algorithm may not be able to achieve these desirable probabilities. In any case, it need not hold that using an efficient design as the start design will always be a better choice than using a non-efficient design. Nevertheless, this result does highlight that for the specific example chosen, there appears greater room for improvement in terms of statistical efficiency for the initial optimal orthogonal design ($D_p$-error = 0.154 to 0.072) then there was for the initial efficient design ($D_p$-error = 0.058 to 0.057).

Examination of Figures 9.2 and 9.3 reveals that the attribute levels for the attributes that were not allowed to take continuous levels are the same as those assumed in the initial designs (see Figure 9.1). The attribute levels of attribute X3 however are now no longer fixed integers, but rather continuous levels fixed within the range specified in the syntax.

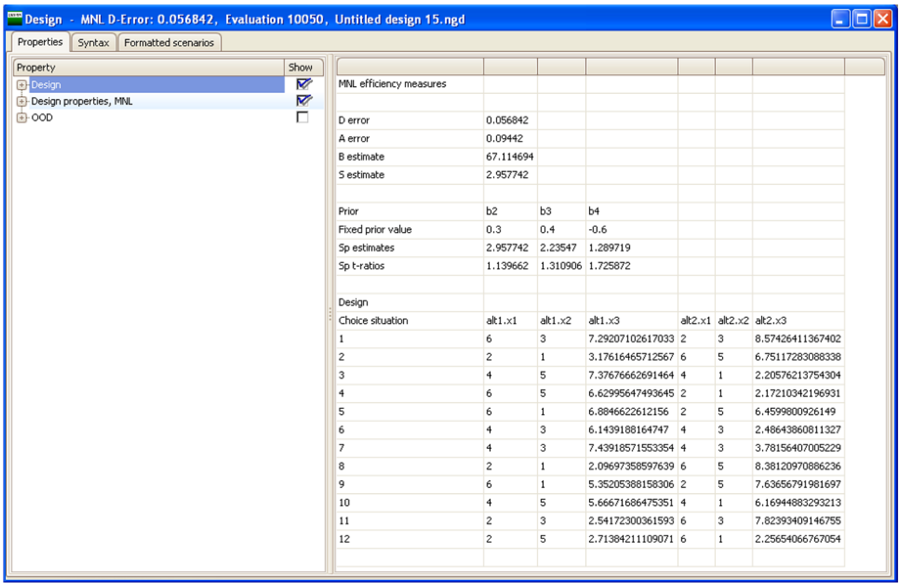


Design - MNL D-Error: 0.072011, Evaluation 10494, Untitled design 14.ngd

| Property | Show |
|---|---|
| Design | ☑ |
| Design properties, MNL | ☑ |
| OOD | ☐ |

| | | | | | | |
|---|---|---|---|---|---|---|
| MNL efficiency measures | | | | | | |
| | | | | | | |
| D error | 0.072011 | | | | | |
| A error | 0.09856 | | | | | |
| B estimate | 68.432365 | | | | | |
| S estimate | 3.479838 | | | | | |
| | | | | | | |
| Prior | b2 | b3 | b4 | | | |
| Fixed prior value | 0.3 | 0.4 | -0.6 | | | |
| Sp estimates | 3.479838 | 2.44226 | 1.19983 | | | |
| Sp t-ratios | 1.050695 | 1.254181 | 1.789354 | | | |
| | | | | | | |
| Design | | | | | | |
| Choice situation | alt1.x1 | alt1.x2 | alt1.x3 | alt2.x1 | alt2.x2 | alt2.x3 |
| 1 | 4 | 3 | 5.59712932911795 | 6 | 5 | 5.6107700624284 |
| 2 | 4 | 5 | 5.79720065471746 | 6 | 1 | 6.26548924525923 |
| 3 | 6 | 1 | 4.57855642368609 | 2 | 3 | 6.0468394891823 |
| 4 | 6 | 1 | 7.14112791546129 | 2 | 3 | 8.6093999358312 |
| 5 | 2 | 1 | 4.99666992524282 | 4 | 3 | 5.01033067668173 |
| 6 | 6 | 5 | 9.1293998712379 | 2 | 1 | 4.07422911371151 |
| 7 | 4 | 3 | 3.31542571483758 | 6 | 5 | 8.61501201882942 |
| 8 | 6 | 3 | 7.40264345976865 | 2 | 5 | 5.04385462987189 |
| 9 | 4 | 5 | 7.02017634565852 | 6 | 1 | 3.66137603596908 |
| 10 | 2 | 1 | 3.45774218439788 | 4 | 3 | 3.47138292616163 |
| 11 | 2 | 5 | 7.34413351284359 | 4 | 1 | 7.8124083855996 |
| 12 | 2 | 3 | 2.83940889357917 | 4 | 5 | 8.13899254704534 |

**Figure 9.2: Continuous level design based using an optimal orthogonal design as the start design**



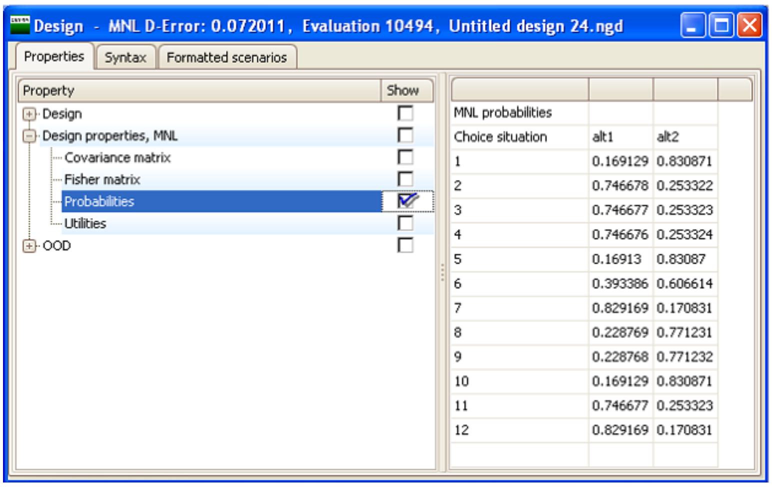
Design - MNL D-Error: 0.056842, Evaluation 10050, Untitled design 15.ngd

| Property | Show |
|---|---|
| Design | ☑ |
| Design properties, MNL | ☑ |
| OOD | ☐ |

| | | | | | | |
|---|---|---|---|---|---|---|
| MNL efficiency measures | | | | | | |
| | | | | | | |
| D error | 0.056842 | | | | | |
| A error | 0.09442 | | | | | |
| B estimate | 67.114694 | | | | | |
| S estimate | 2.957742 | | | | | |
| | | | | | | |
| Prior | b2 | b3 | b4 | | | |
| Fixed prior value | 0.3 | 0.4 | -0.6 | | | |
| Sp estimates | 2.957742 | 2.23547 | 1.289719 | | | |
| Sp t-ratios | 1.139662 | 1.310906 | 1.725872 | | | |
| | | | | | | |
| Design | | | | | | |
| Choice situation | alt1.x1 | alt1.x2 | alt1.x3 | alt2.x1 | alt2.x2 | alt2.x3 |
| 1 | 6 | 3 | 7.29207102617033 | 2 | 3 | 8.57426411367402 |
| 2 | 2 | 1 | 3.17616465712567 | 6 | 5 | 6.75117283088338 |
| 3 | 4 | 5 | 7.37676662691464 | 4 | 1 | 2.20576213754304 |
| 4 | 6 | 5 | 6.62995647493645 | 2 | 1 | 2.17210342196931 |
| 5 | 6 | 1 | 6.8846622612156 | 2 | 5 | 6.4599800926149 |
| 6 | 4 | 3 | 6.1439188164747 | 4 | 3 | 2.48643860811327 |
| 7 | 4 | 3 | 7.43918571553354 | 4 | 3 | 3.78156407005229 |
| 8 | 2 | 1 | 2.09697358597639 | 6 | 5 | 8.38120970886236 |
| 9 | 6 | 1 | 5.35205388158306 | 2 | 5 | 7.63656791981697 |
| 10 | 4 | 5 | 5.66671686475351 | 4 | 1 | 6.16944883293213 |
| 11 | 2 | 3 | 2.54172300361593 | 6 | 3 | 7.82393409146755 |
| 12 | 2 | 5 | 2.71384211109071 | 6 | 1 | 2.25654066767054 |

**Figure 9.3: Continuous level design based using an efficient design as the start design**

Figure 9.4 show the choice probabilities for the continuous attribute level design shown in Figure 9.2. Whilst the choice probabilities are not exactly the same within each choice situation (as should occur if one used the analytical choice probabilities to design the experiment), there does appear to be certain probabilities that re-occur over the design. This once more highlights problems with trying to impose utility or probability balance in a design (see Section 7.1.8), as such probabilities will typically result in a significant loss of statistical efficiency.



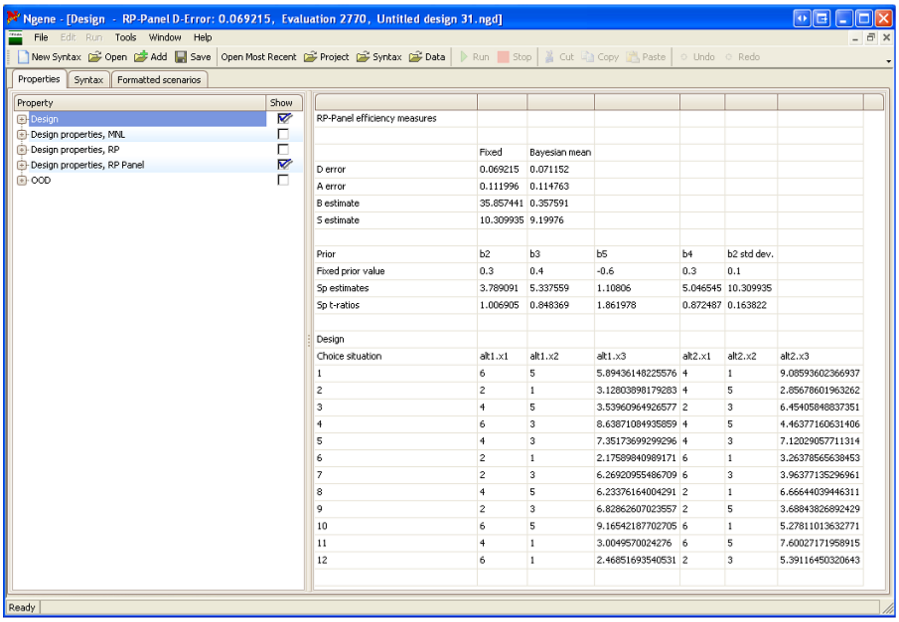
| MNL probabilities | | |
|---|---|---|
| Choice situation | alt1 | alt2 |
| 1 | 0.169129 | 0.830871 |
| 2 | 0.746678 | 0.253322 |
| 3 | 0.746677 | 0.253323 |
| 4 | 0.746676 | 0.253324 |
| 5 | 0.16913 | 0.83087 |
| 6 | 0.393386 | 0.606614 |
| 7 | 0.829169 | 0.170831 |
| 8 | 0.228769 | 0.771231 |
| 9 | 0.228768 | 0.771232 |
| 10 | 0.169129 | 0.830871 |
| 11 | 0.746677 | 0.253323 |
| 12 | 0.829169 | 0.170831 |

**Figure 9.4: Choice probabilities for a continuous level design**

The examples shown above are for MNL designs with only two alternatives and generic parameter estimates. As suggested however, the Nelder-Mead algorithm is not limited to problems dealing with MNL designs or to problems involving only two alternatives. Indeed, the procedures outlined above may be applied to any model type, as well as to experimental design problems with any number of alternatives. Further, the method can also be applied with Bayesian prior parameter distributions. Figure 9.5 shows a design generated with continuous attribute levels for a panel MMNL with generic and alternative specific parameters allowing for Bayesian prior parameters based on the syntax below. We present this Figure to demonstrate the flexibility of the approach.

```
Design
;alts = Alt1, Alt2, Alt3
;rows = 12
;eff = (rppanel, d)
;bdraws = gauss(2)
;rdraws = gauss(2)
;model:
U(Alt1)  = b1[-0.2] + b2[n,0.3,0.1] * X1[2,4,6] + b3[(n,0.4,0.1)] * X2
[1,3,5] + b5[-0.6] * X3[2,5,8] /
U(Alt2) =            b2             * X1         + b4[(n,0.3,0.1)] * X2
     + b5       * X3
$
```

```
Design
;alts = Alt1, Alt2, alt3
;rows = 12
;eff = (rppanel, d)
;bdraws = gauss(2)
;rdraws = gauss(2)
;alg=neldermead
;start= RP panel efficient.ngd
;model:
U(Alt1)  = b1[-0.2] + b2[n,0.3,0.1]  * X1[2,4,6] + b3[(n,0.4,0.1)] * X2
[1,3,5] + b5[-0.6] * X3[2:10] /
U(Alt2) =             b2             * X1        + b4[(n,0.3,0.1)] * X2
     + b5       * X3[2:10]
$
```


Ngene - [Design - RP-Panel D-Error: 0.069215, Evaluation 2770, Untitled design 31.ngd]
File Edit Run Tools Window Help
New Syntax Open Add Save Open Most Recent Project Syntax Data Run Stop Cut Copy Paste Undo Redo
Properties Syntax Formatted scenarios
Property Show
Design ☑
Design properties, MNL □
Design properties, RP □
Design properties, RP Panel ☑
OOD □
Ready

| | | | | | | |
|---|---|---|---|---|---|---|
| RP-Panel efficiency measures | | | | | | |
| | | | | | | |
| | Fixed | Bayesian mean | | | | |
| D error | 0.069215 | 0.071152 | | | | |
| A error | 0.111996 | 0.114763 | | | | |
| B estimate | 35.857441 | 0.357591 | | | | |
| S estimate | 10.309935 | 9.19976 | | | | |
| | | | | | | |
| Prior | b2 | b3 | b5 | b4 | b2 std dev. | |
| Fixed prior value | 0.3 | 0.4 | -0.6 | 0.3 | 0.1 | |
| Sp estimates | 3.789091 | 5.337559 | 1.10806 | 5.046545 | 10.309935 | |
| Sp t-ratios | 1.006905 | 0.848369 | 1.861978 | 0.872487 | 0.163822 | |
| | | | | | | |
| Design | | | | | | |
| Choice situation | alt1.x1 | alt1.x2 | alt1.x3 | alt2.x1 | alt2.x2 | alt2.x3 |
| 1 | 6 | 5 | 5.89436148225576 | 4 | 1 | 9.08593602366937 |
| 2 | 2 | 1 | 3.12803898179283 | 4 | 5 | 2.85678601963262 |
| 3 | 4 | 5 | 3.53960964926577 | 2 | 3 | 6.45405848837351 |
| 4 | 6 | 3 | 8.63871084935859 | 4 | 5 | 4.46377160631406 |
| 5 | 4 | 3 | 7.35173699299296 | 4 | 3 | 7.12029057711314 |
| 6 | 2 | 1 | 2.17589840989171 | 6 | 1 | 3.26378565638453 |
| 7 | 2 | 3 | 6.26920955486709 | 6 | 3 | 3.96377135296961 |
| 8 | 4 | 5 | 6.23376164004291 | 2 | 1 | 6.66644039446311 |
| 9 | 2 | 3 | 6.82862607023557 | 2 | 5 | 3.68843826892429 |
| 10 | 6 | 5 | 9.16542187702705 | 6 | 1 | 5.27811013632771 |
| 11 | 4 | 1 | 3.0049570024276 | 6 | 5 | 7.60027171958915 |
| 12 | 6 | 1 | 2.46851693540531 | 2 | 3 | 5.39116450320643 |

**Figure 9.5: Choice probability design with continuous attribute levels for a panel MMNL model**

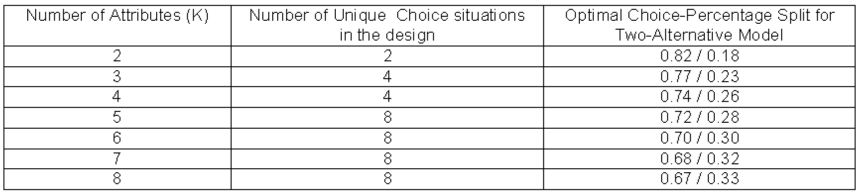
## 9.3 Appendix 9A Steps in generating choice designs with continuous attribute levels

There exist three main steps in generating CP types of designs. We now outline these steps.

*Step 1*: Generate an initial start design. Kanninen (2002, 2005) and Johnson et al. (2006) suggests that this initial design be such that it represents only *k*-1 attributes (i.e., the initial design omits a single (common across alternatives) attribute for each of the alternatives). The k$^{th}$ omitted attribute in CP designs must be continuous in nature, otherwise the method will not work. Given that most choice problems will contain a price or cost attribute, Kanninen suggests that the *k*$^{th}$ omitted attribute be that attribute (in transport problems, time attributes will often also be present, and hence may also be used in generating CP designs). For best results, Johnson et al. (2006) recommends that the initial design be orthogonal and in the case of two alternatives with all attributes taking two levels, that the second alternative be constructed using the foldover of the first alternative.

*Step 2*: Select attribute levels for the $k^{th}$ omitted attribute such that the choice probabilities for each choice situation in the design assume certain values. Note that as with efficient designs, the generation of CP designs requires the use of prior parameter estimates in order to determine the choice probabilities over the design. If zero-valued priors are assumed, as with optimal orthogonal designs, then the choice probabilities will simply be fixed and equal to 1/*J* and hence it will not be possible to generate the design. In allocating the attribute levels, the desirable choice probabilities that the analyst should attempt to aim for are shown in Table 9A.1 for a small number of designs. In generating values for the $k^{th}$ attribute, the analyst may have to let go of the attribute level balance assumption common in generating designs, and further, may have to let go of the assumption that the attribute can only take on integer values.

**Table 9A.1: Optimal Choice probability values for specific designs (adapted Johnson *et al.* 2006)**



| Number of Attributes (K) | Number of Unique Choice situations in the design | Optimal Choice-Percentage Split for Two-Alternative Model |
|---|---|---|
| 2 | 2 | 0.82 / 0.18 |
| 3 | 4 | 0.77 / 0.23 |
| 4 | 4 | 0.74 / 0.26 |
| 5 | 8 | 0.72 / 0.28 |
| 6 | 8 | 0.70 / 0.30 |
| 7 | 8 | 0.68 / 0.32 |
| 8 | 8 | 0.67 / 0.33 |

The probabilities shown in Table 9A.1 were derived analytically. Rather than rely on these probabilities which are known only for designs generated for MNL models, as well as for only a small subset of cases, Ngene uses the Nelder Mead algorithm to search for an optimal design. The Nelder Mead algorithm is discussed in Appendix 9B.

*Step 3*: The final stage, advocated by Kanninen, is to update the prior parameter values and attribute levels so as to optimise the AVC matrix for the data. Seeing that discrete choice modelling is undertaken on choice data and not on choice designs, Johnson *et al*. (2006) advocates using a large pilot or pretest sample, and/or stopping the main sample partway through so as to update the prior parameter values used in generating the original design. With the new updated priors, the levels of the changing attribute can be reworked so as to produce the desired choice probabilities for the *data*. As such, over the course of data collection, different respondents may be given different versions of the design, at least in terms of what they observe for the attribute that is allowed to change.

## 9.4 Appendix 9B The Nelder Mead algorithm

### 9B.1 Introduction

The Nelder-Mead method (Nelder & Mead, 1965) is a computational technique for solving non-linear optimisation problems. The method is what is known as a *local search* technique (also referred to as an *incomplete* method). This means that although the method will locate a solution to a problem, that solution may only be *locally* optimal rather than *globally* optimal (so there may exist a better solution that the method fails to find). The motivation for the use of local methods is that guaranteeing the optimality of a solution is for many problems too computationally intensive to be feasible and is often of little practical benefit. Although in theory there are situations where the Nelder-Mead method will not terminate, in practice the finite precision and bounds of the floating point numbers used in digital computers guarantee that the method will (eventually) converge and terminate.

### 9B.2 Operation

The method maintains a set of tentative solutions. The size of this set is determined by the number of unknowns in the problem. For a problem with *N* unknowns, a set of *N*+1 tentative solutions will be maintained. With respect to optimising SP experimental designs with continuous attributes, the number of unknowns is the number of continuous attributes multiplied by the number of rows in the design. For the initial set of solutions, the levels for the continuous attributes are allocated randomly. Following the initial random allocation of attribute levels, the algorithm iteratively either improves upon the current worst tentative solution or shrinks all tentative solutions towards the best solution. The specific process is as follows.

### 9B.3 Reflection/Extension

The centroid of the set of tentative solutions (excluding the worst) is first calculated. A new tentative solution is obtained by reflecting the worst solution through this centroid, the rationale being that moving away from the worst solution towards the others should result in an improved solution. If the reflected solution does not improve upon the objective value of the worst solution the procedure skips to contraction. Otherwise, if the new solution is an improvement, a further extension away from the worst solution is considered. When the worst solution is a distance *d* from the centroid then the reflected and extended solutions are a distance of 2*d* and 3*d* respectively from the worst solution. The better of these two solutions replaces the worst tentative solution and the iteration is complete.

### 9B.4 Contraction

If reflection does not result in an improved solution, alternate solutions involving smaller changes are considered. Two solutions are considered: one halfway between the current worst and the centroid, the second halfway between the centroid and the reflected solution. Continuing the previous discussion, these solutions will be distances of 0.5*d* and 1.5*d* from the worst solution respectively. If neither of these solutions improves upon the worst the method instead applies shrinking. Otherwise, the contracted solution with the better objective function valuation is adopted in place of the current worst and the iteration is complete.

### 9B.5 Shrinking

If neither of the above steps has produced an improved solution, then all tentative solutions (including the worst) are moved towards the best solution by a factor of 0.5, irrespective of whether this leads to improvements in their respective objective function valuations. This concludes the iteration.

Unless the procedure exhausts its computational resources (i.e. a specified time or number of iterations has elapsed) it will continue to iterate until all tentative solutions are within a distance $\delta$ of the best solution. For a design with *N* rows and *K* continuous attributes, $\delta$ is defined to be:

$$\delta = \alpha N \sum_{k=1}^{K} \left( \max_k - \min_k \right)^2$$

where α is a user-definable value within the range 0 < α < 1. Smaller values of a lead to tighter convergence criteria and hence more iterations prior to convergence.

### 9B.6 Multiple Runs and Tries

As with most local search algorithms, the solution obtained will depend on the starting conditions (the initial set of random tentative solutions). Running the procedure multiple times from different starting locations ensures that a single bad starting location does not unduly prejudice the final outcome. The version of the Nelder-Mead procedure implemented in Ngene allows multiple repeated *runs* and *tries* for this reason.

Each run is entirely independent and may involve multiple tries. Tries are not independent. Within a run, the best solution from each try is conveyed to the subsequent try (as one of the initial tentative solutions) so only the first try of a run uses an entirely random set of initial solutions. This ensures that each try within a run produces an improved (or at least not worsened) solution. A try ends when computational resources are exhausted or the tentative solutions converge. If computational resources remain, a new try begins.

As runs are independent, additional runs do not necessarily lead to improved solutions. It must also be noted that on the first run, a single copy of the design that is initially passed to the procedure is preserved as a tentative solution, meaning that on this run only, the solution set for the first try is not entirely random.

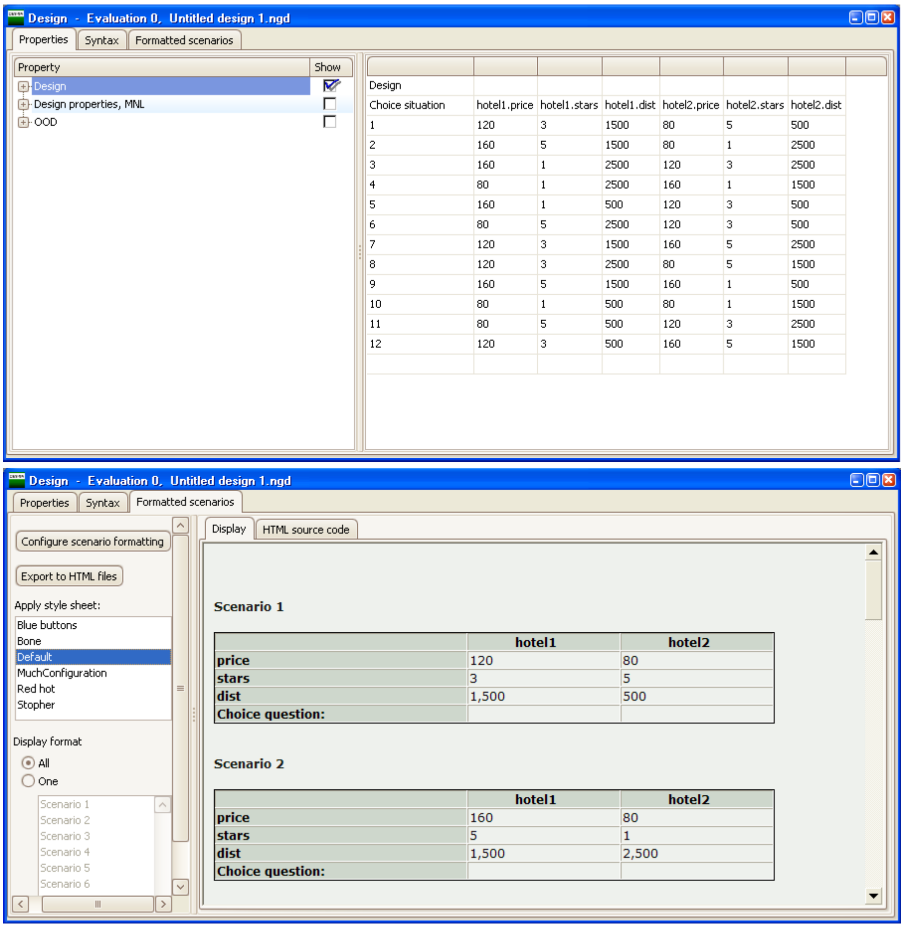
# Chapter 10

# Formatting experiments

# 10 Formatting experiments

The purpose of generating an experimental design is to create attribute levels for choice situations in a survey. The respondent cannot be directly faced with the experimental design, as a matrix of numbers does not have any meaning for the respondent. Instead, the experimental design matrix has to be converted to choice situations that make sense to the respondent.

Ngene has the capability of transforming the design matrix to actual choice situations that can be shown to respondents. In the design window, clicking on the “Formatted scenarios” tab brings up a new screen in which the choice situations in the design are presented in a format that can be understood by respondents (see Figure 10.1). Each row in the design will be put in a separate table that presents the alternatives and attributes in a choice setting.



Design

| Choice situation | hotel1.price | hotel1.stars | hotel1.dist | hotel2.price | hotel2.stars | hotel2.dist |
|---|---|---|---|---|---|---|
| 1 | 120 | 3 | 1500 | 80 | 5 | 500 |
| 2 | 160 | 5 | 1500 | 80 | 1 | 2500 |
| 3 | 160 | 1 | 2500 | 120 | 3 | 2500 |
| 4 | 80 | 1 | 2500 | 160 | 1 | 1500 |
| 5 | 160 | 1 | 500 | 120 | 3 | 500 |
| 6 | 80 | 5 | 2500 | 120 | 3 | 500 |
| 7 | 120 | 3 | 1500 | 160 | 5 | 2500 |
| 8 | 120 | 3 | 2500 | 80 | 5 | 1500 |
| 9 | 160 | 5 | 1500 | 160 | 1 | 500 |
| 10 | 80 | 1 | 500 | 80 | 1 | 1500 |
| 11 | 80 | 5 | 500 | 120 | 3 | 2500 |
| 12 | 120 | 3 | 500 | 160 | 5 | 1500 |

Apply style sheet:
Blue buttons
Bone
Default
MuchConfiguration
Red hot
Stopher

Display format
● All
○ One

Scenario 1
Scenario 2
Scenario 3
Scenario 4
Scenario 5
Scenario 6

**Scenario 1**

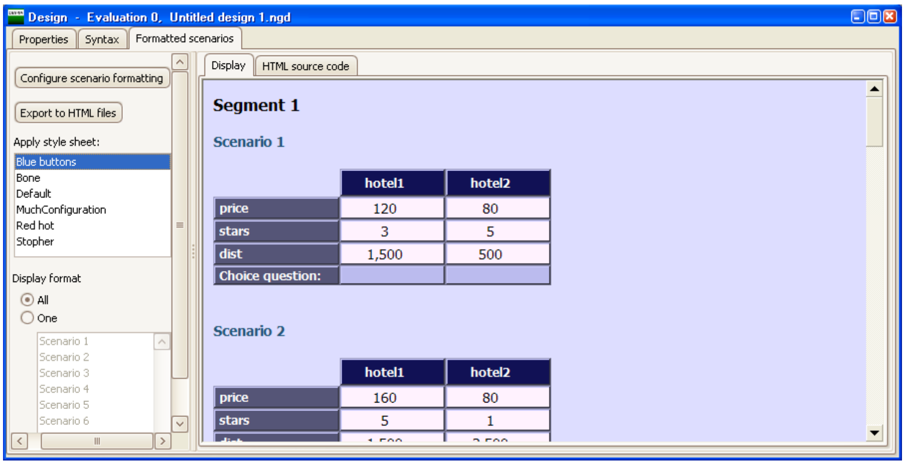
| | hotel1 | hotel2 |
|---|---|---|
| **price** | 120 | 80 |
| **stars** | 3 | 5 |
| **dist** | 1,500 | 500 |
| **Choice question:** | | |

**Scenario 2**

| | hotel1 | hotel2 |
|---|---|---|
| **price** | 160 | 80 |
| **stars** | 5 | 1 |
| **dist** | 1,500 | 2,500 |
| **Choice question:** | | |

**Figure 10.1: Moving from the design matrix to actual choice situations**

Different style sheets can be applied, changing the colors and fonts. Users can create their own stylesheets (*.css files) and put them in the Stylesheets folder found within the Ngene install folder. An example of a different style sheet being applied is shown in Figure 10.2.



Design - Evaluation 0, Untitled design 1.ngd

Apply style sheet:

- Blue buttons
- Bone
- Default
- MuchConfiguration
- Red hot
- Stopher

Display format

● All
○ One

Scenario 1
Scenario 2
Scenario 3
Scenario 4
Scenario 5
Scenario 6

**Segment 1**

Scenario 1

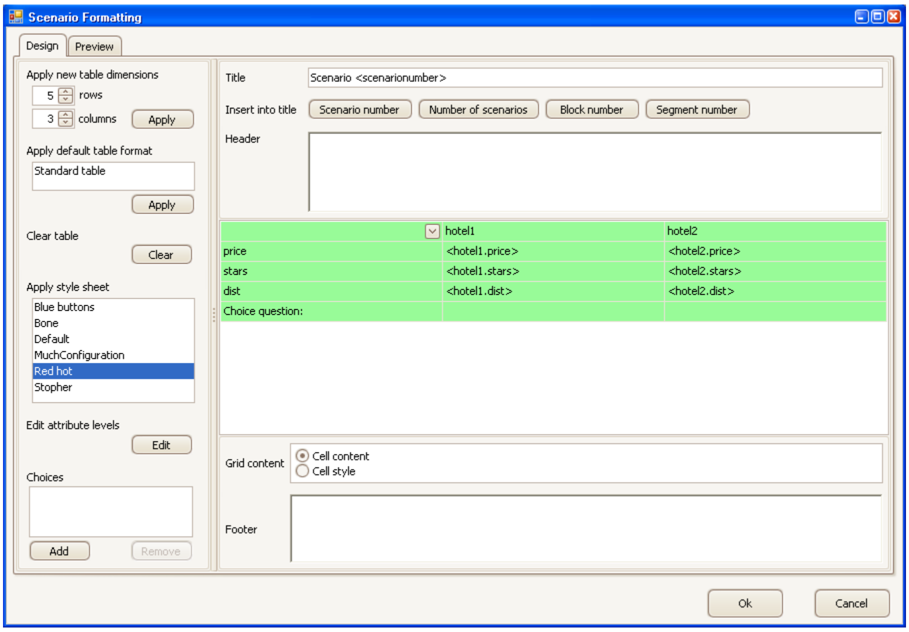
| | hotel1 | hotel2 |
|---|---|---|
| price | 120 | 80 |
| stars | 3 | 5 |
| dist | 1,500 | 500 |
| Choice question: | | |

Scenario 2

| | hotel1 | hotel2 |
|---|---|---|
| price | 160 | 80 |
| stars | 5 | 1 |

**Figure 10.2: Style sheets change the look of the choice situations**

Clicking on the “Configure scenario formatting” button brings up the scenario formatting screen as shown in Figure 10.3.



Scenario Formatting

Apply new table dimensions

5 rows

3 columns

Apply default table format

Standard table

Clear table

Apply style sheet

- Blue buttons
- Bone
- Default
- MuchConfiguration
- Red hot
- Stopher

Edit attribute levels

Choices

Title Scenario <scenarionumber>

Insert into title

Header

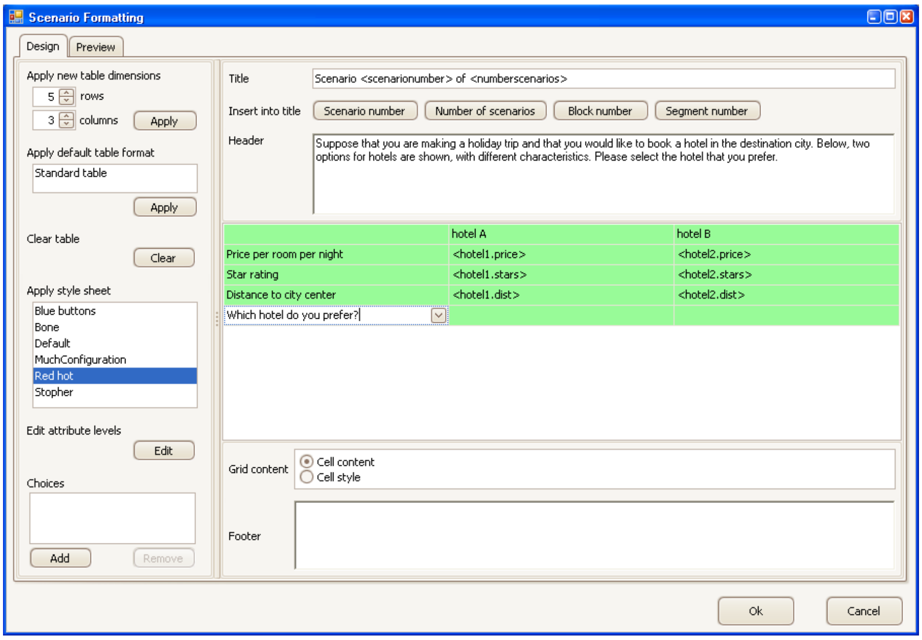
| | hotel1 | hotel2 |
|---|---|---|
| price | <hotel1.price> | <hotel2.price> |
| stars | <hotel1.stars> | <hotel2.stars> |
| dist | <hotel1.dist> | <hotel2.dist> |
| Choice question: | | |

Grid content ● Cell content ○ Cell style

Footer

**Figure 10.3: Scenario formatting screen**

The user can enter titles, headers and footers to the choice screens. Furthermore, the names of the alternatives and attributes presented to the respondent can be entered, changing the names obtained from the syntax file. This is shown in Figure 10.4.



5 rows

3 columns

Apply default table format: Standard table

Apply style sheet: Red hot

Title: Scenario <scenarionumber> of <numberscenarios>

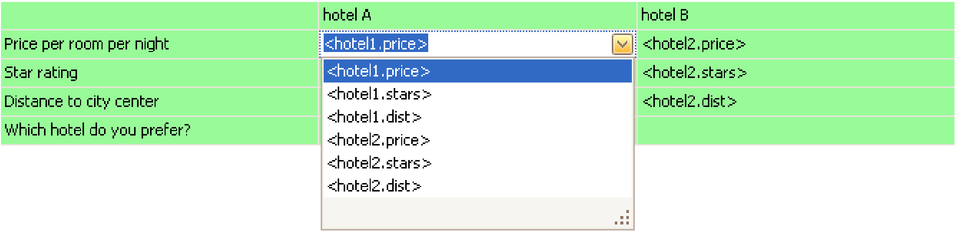
Header: Suppose that you are making a holiday trip and that you would like to book a hotel in the destination city. Below, two options for hotels are shown, with different characteristics. Please select the hotel that you prefer.

| | hotel A | hotel B |
|---|---|---|
| Price per room per night | <hotel1.price> | <hotel2.price> |
| Star rating | <hotel1.stars> | <hotel2.stars> |
| Distance to city center | <hotel1.dist> | <hotel2.dist> |
| Which hotel do you prefer? | | |

Grid content: ● Cell content ○ Cell style

Figure 10.4: Entering title, header and footer text, and change names of alternatives and attributes

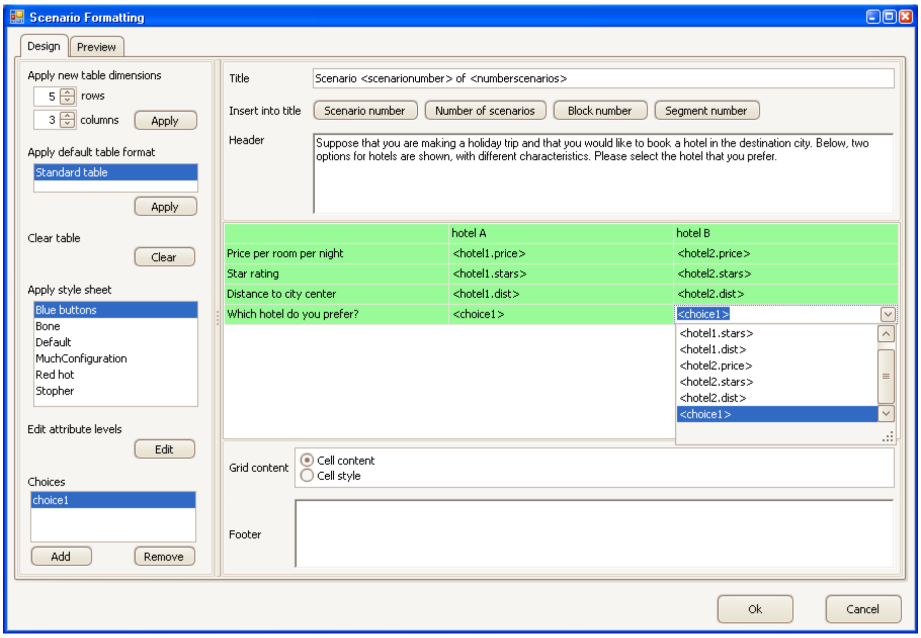
In case the user would like to change the order of the attributes or alternatives, the appropriate cell can be selected, and from the pull-down list the required alternative/attribute combination can be selected, see Figure 10.5.



| | hotel A | hotel B |
|---|---|---|
| Price per room per night | <hotel1.price> | <hotel2.price> |
| Star rating | | <hotel2.stars> |
| Distance to city center | | <hotel2.dist> |
| Which hotel do you prefer? | | |

Figure 10.5: Changing attribute order

To include radio buttons for the respondent to be able to select the preferred alternative, choices have to be added. In the lower left corner, a choice can be added and named. Once at least one choice has been added, it can be selected from the pull-down menu, see Figure 10.6. Choices with the same name are grouped, i.e. only one of the radio buttons in such a group can be selected. In some cases, multiple choices are required (such as a forced and an unforced choice if a no-choice alternative is included).



Title: Scenario <scenarionumber> of <numberscenarios>

Header: Suppose that you are making a holiday trip and that you would like to book a hotel in the destination city. Below, two options for hotels are shown, with different characteristics. Please select the hotel that you prefer.

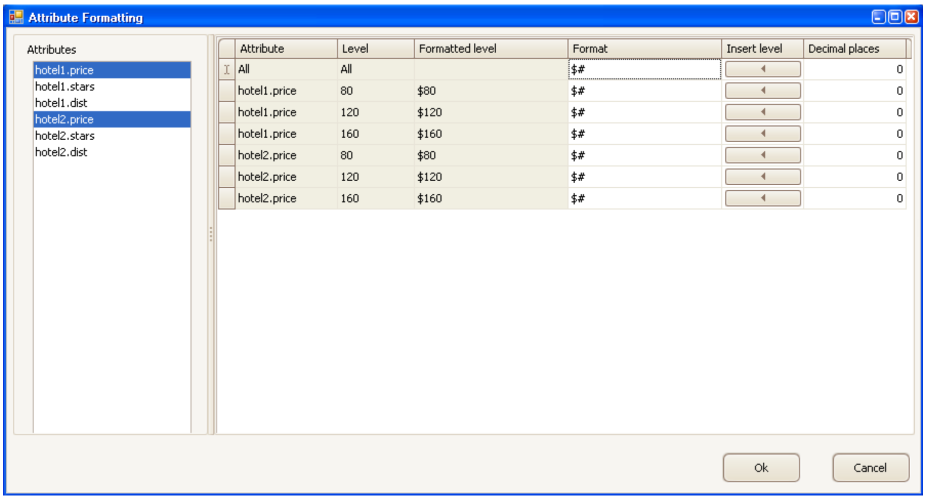
| | hotel A | hotel B |
|---|---|---|
| Price per room per night | <hotel1.price> | <hotel2.price> |
| Star rating | <hotel1.stars> | <hotel2.stars> |
| Distance to city center | <hotel1.dist> | <hotel2.dist> |
| Which hotel do you prefer? | <choice1> | <choice1> |

Grid content: ● Cell content ○ Cell style

Footer:

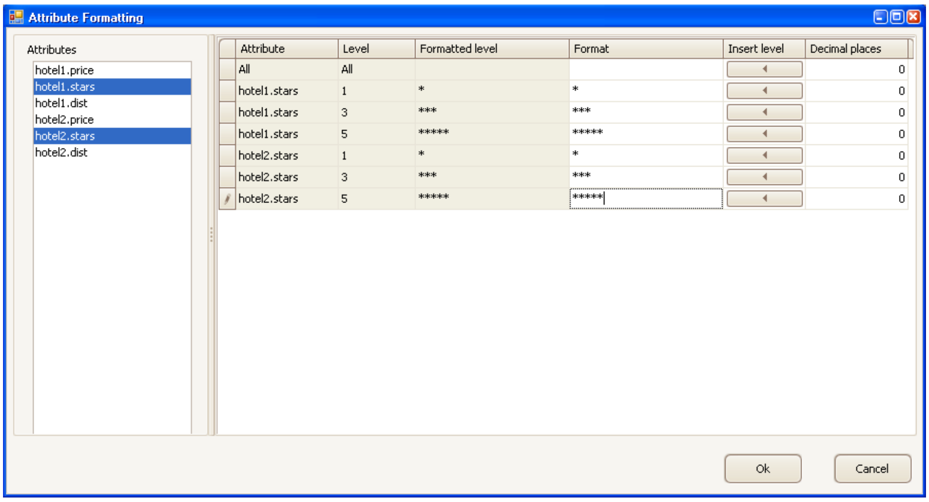
Figure 10.6: Including choices in the choice experiment

Finally, the attribute levels can be formatted by clicking on the “Edit” button on the left, which brings up the attribute formatting window, see Figures 10.7 and 10.8. For each attribute in each alternative, the format of the attribute levels can be altered, such that it does not show just numbers (coding), but shows the true levels to be presented to the respondent. The levels can be formatted for each attribute separately, or for multiple attributes at the same time. For this purpose, select multiple attributes on the left hand side (using <shift>-click to select a whole range, or <ctrl>-click to add extra attributes). The numerical levels from the design and the actual formatted levels are shown. Using the “#” symbol in the format adds the (numerical) level. Instead of numerical levels, words can be used, etc.



| Attribute | Level | Formatted level | Format | Insert level | Decimal places |
|---|---|---|---|---|---|
| All | All | | $# | | 0 |
| hotel1.price | 80 | $80 | $# | | 0 |
| hotel1.price | 120 | $120 | $# | | 0 |
| hotel1.price | 160 | $160 | $# | | 0 |
| hotel2.price | 80 | $80 | $# | | 0 |
| hotel2.price | 120 | $120 | $# | | 0 |
| hotel2.price | 160 | $160 | $# | | 0 |

Figure 10.7: Formatting attribute levels (using numbers)



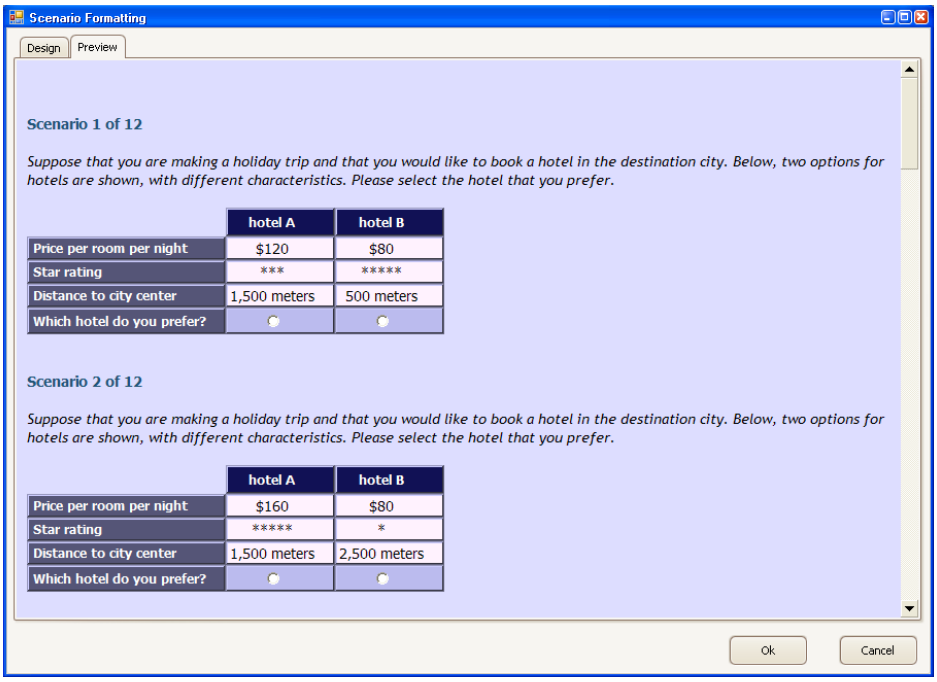
Attributes

- hotel1.price
- hotel1.stars
- hotel1.dist
- hotel2.price
- hotel2.stars
- hotel2.dist

| Attribute | Level | Formatted level | Format | Insert level | Decimal places |
|---|---|---|---|---|---|
| All | All | | | | 0 |
| hotel1.stars | 1 | * | * | | 0 |
| hotel1.stars | 3 | *** | *** | | 0 |
| hotel1.stars | 5 | ***** | ***** | | 0 |
| hotel2.stars | 1 | * | * | | 0 |
| hotel2.stars | 3 | *** | *** | | 0 |
| hotel2.stars | 5 | ***** | ***** | | 0 |

Figure 10.8: Formatting attribute levels (using text or symbols)

While formatting the scenarios, the result can be previewed by clicking the “Preview” tab, see Figure 10.9.



Scenario 1 of 12

*Suppose that you are making a holiday trip and that you would like to book a hotel in the destination city. Below, two options for hotels are shown, with different characteristics. Please select the hotel that you prefer.*

| | hotel A | hotel B |
|---|---|---|
| Price per room per night | $120 | $80 |
| Star rating | *** | ***** |
| Distance to city center | 1,500 meters | 500 meters |
| Which hotel do you prefer? | ○ | ○ |

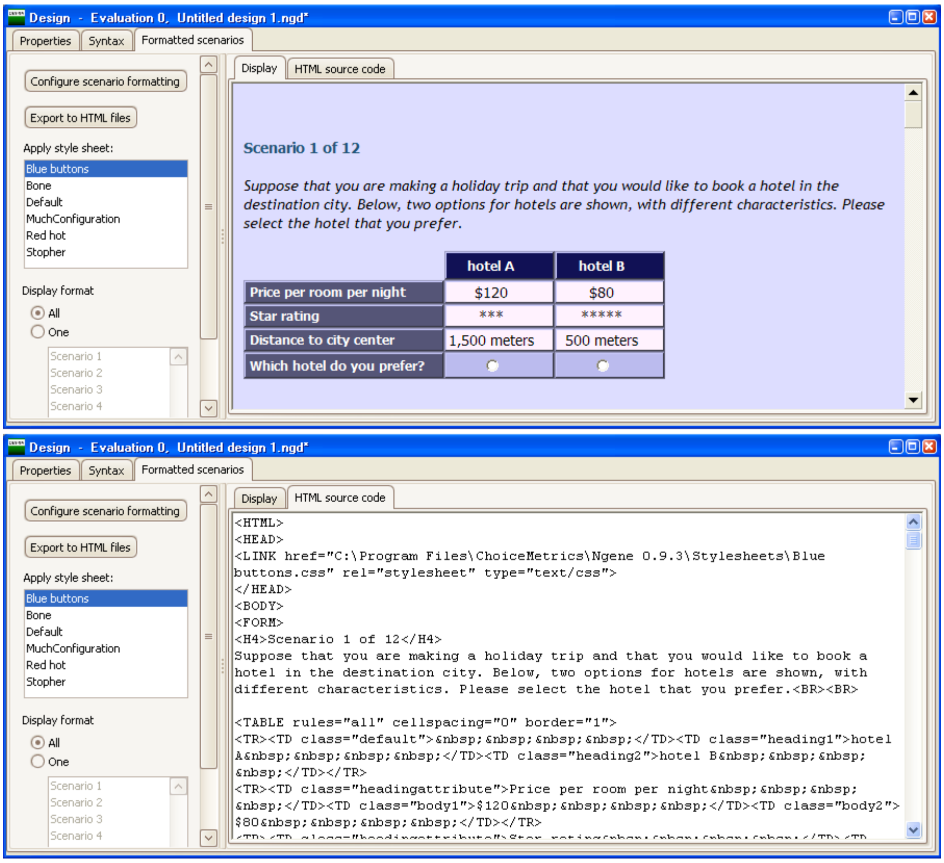
Scenario 2 of 12

*Suppose that you are making a holiday trip and that you would like to book a hotel in the destination city. Below, two options for hotels are shown, with different characteristics. Please select the hotel that you prefer.*

| | hotel A | hotel B |
|---|---|---|
| Price per room per night | $160 | $80 |
| Star rating | ***** | * |
| Distance to city center | 1,500 meters | 2,500 meters |
| Which hotel do you prefer? | ○ | ○ |

Figure 10.9: Previewing the formatted scenarios

After formatting the scenarios, the scenario formatting screen can be closed by pressing “OK”. One can go back to the design and display the final formatted choice screens (with functional radio buttons). The design is formatted using HTML coding, which can be viewed by clicking on the “HTML source code” tab, see Figure 10.10. This way, one can more readily implement the choice experiment as an internet survey, although adding extra questions, managing multiple screens, and storing the results in a database requires extra work.



**Scenario 1 of 12**

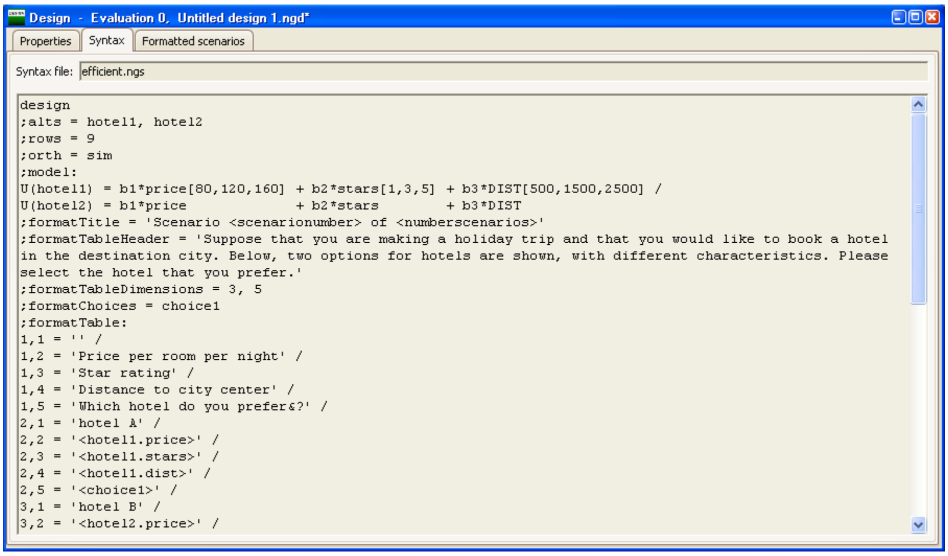
*Suppose that you are making a holiday trip and that you would like to book a hotel in the destination city. Below, two options for hotels are shown, with different characteristics. Please select the hotel that you prefer.*

| | hotel A | hotel B |
|---|---|---|
| Price per room per night | $120 | $80 |
| Star rating | *** | ***** |
| Distance to city center | 1,500 meters | 500 meters |
| Which hotel do you prefer? | ○ | ○ |

```
<HTML>
<HEAD>
<LINK href="C:\Program Files\ChoiceMetrics\Ngene 0.9.3\Stylesheets\Blue
buttons.css" rel="stylesheet" type="text/css">
</HEAD>
<BODY>
<FORM>
<H4>Scenario 1 of 12</H4>
Suppose that you are making a holiday trip and that you would like to book a
hotel in the destination city. Below, two options for hotels are shown, with
different characteristics. Please select the hotel that you prefer.<BR><BR>

<TABLE rules="all" cellspacing="0" border="1">
<TR><TD class="default">&nbsp;&nbsp;&nbsp;&nbsp;</TD><TD class="heading1">hotel
A&nbsp;&nbsp;&nbsp;&nbsp;</TD><TD class="heading2">hotel B&nbsp;&nbsp;&nbsp;
&nbsp;</TD></TR>
<TR><TD class="headingattribute">Price per room per night&nbsp;&nbsp;&nbsp;
&nbsp;</TD><TD class="body1">$120&nbsp;&nbsp;&nbsp;&nbsp;</TD><TD class="body2">
$80&nbsp;&nbsp;&nbsp;&nbsp;</TD></TR>
```

**Figure 10.10: Final formatted choice scenarios and HTML source code**

If the design is saved, the design formatting will be included in the syntax so that the formatting is preserved when the design is reopened. Clicking on the “Syntax” tab shows the syntax in which extra lines have been added to describe the formatting, see Figure 10.11. It is recommended that these properties not be altered directly from the syntax, but through the above tools instead. Directly altering the properties might cause Ngene to crash.



Syntax file: efficient.ngs

```
design
;alts = hotel1, hotel2
;rows = 9
;orth = sim
;model:
U(hotel1) = b1*price[80,120,160] + b2*stars[1,3,5] + b3*DIST[500,1500,2500] /
U(hotel2) = b1*price             + b2*stars        + b3*DIST
;formatTitle = 'Scenario <scenarionumber> of <numberscenarios>'
;formatTableHeader = 'Suppose that you are making a holiday trip and that you would like to book a hotel
in the destination city. Below, two options for hotels are shown, with different characteristics. Please
select the hotel that you prefer.'
;formatTableDimensions = 3, 5
;formatChoices = choice1
;formatTable:
1,1 = '' /
1,2 = 'Price per room per night' /
1,3 = 'Star rating' /
1,4 = 'Distance to city center' /
1,5 = 'Which hotel do you prefer&?' /
2,1 = 'hotel A' /
2,2 = '<hotel1.price>' /
2,3 = '<hotel1.stars>' /
2,4 = '<hotel1.dist>' /
2,5 = '<choice1>' /
3,1 = 'hotel B' /
3,2 = '<hotel2.price>' /
```

**Figure 10.11: Formatting syntax**

Note that currently only the first design will be shown with scenario formatting. So if multiple designs are specified with the *fisher* property, only the first will be shown.

# Chapter 11

# Syntax Reference

# 11 Syntax Reference

The following is an alphabetically ordered list of commands and properties for each command.

## 11.1 Definitions of syntax components

**Routine**
A routine consists of a single command and one or more properties. When a routine is run, a single task is performed. Below is an example routine, which will be referred to in the following definitions.

```
Design
;alts(model1) = car, bus, train
;rows = 12
;eff = model1(d)
;alg = swap(reset=10000, resetinc=5000)
;model(model1):
 U(car)   =   /
 U(bus)   =   /
 U(train) =
$
```

**Command**
**(e.g. `Design`)**
A command instructs Ngene to run a particular type of task. It needs to be configured with one or more properties.

**Property**
**(e.g. `alts`, `rows`)**
A property provides Ngene with information on how to run the task specified in the command. The property word is specified immediately after a semicolon.

**Property value**
**(e.g. `12`, `car`)**
A property value is a piece of information assigned to a property.

**Property qualification**
**(e.g. `(model1)`)**
Some properties can be specified more than once. When this happens, each property that is repeated needs to be qualified by a label that is specified between brackets immediately after the property.

**Label**
**(e.g. `car`, `model1`)**
A label is a user specified word that is used to identify something that has been defined in syntax. In the above example, the label 'car' identifies one of the alternatives, while 'model1' identifies a single model specification. Labels cannot be reserved words.

**Directive**
**(e.g. `;rows = 12`)**
A directive is the combination of a property and a property value. Any configured property will be referred to as a directive in the manual and in error messages to improve clarity.

**Parameter**
**(e.g. `reset=10000`)**
Some properties allow additional information to be specified. These are specified in brackets as a series of comma separated parameter name-parameter value pairs. In the above example, the alg=swap directive has two parameters specified that provide additional information.

## 11.2 How this manual specifies syntax

Some of the Ngene syntax is very complex, and hence we have adopted some conventions in how we prescribe the syntax, so as to avoid ambiguity.

**Syntax in *italics* is optional**

**User specified values**
In many instances, the user will need to enter their own value into the syntax. e.g. 12 in ;rows=12. These user specified values will be treated as follows in our syntax prescription:
`<data type(label)>`
where data type can be:
- integer - a whole number
- decimal - a number with any level of precision
- string - a text value

The label will be a very concise description of what the user specified value is for, and will typically be referred to in the comments section using italics ("label").

**Text colour**
`Blue text` is used to represent syntax that must be specified verbatim. e.g. `;con`
`Red text` is used for other instructions, and should not be entered as is. Examples include user specified values, `[mutually | exclusive | alternatives]` (see below), and repetition ( `...`, see below).

**Mutually exclusive alternatives**
Sometimes several options are available, but only one can be applied. In this case, the mutually exclusive options are surrounded by red square brackets ("[]"), and separated by red pipe symbols ("|"). The orth property is a good example:

```
;orth = [ sim
        | seq
        | seq2
        | ood ]
```

where the four possible cases are:

```
;orth = sim
;orth = seq
;orth = seq2
;orth = ood
```

It is also possible to have multiple levels of square brackets, nested inside each other.

**Repetition**
Often, several items, each with the same syntax rules, can be specified in some sort of list. They may be separated by commas, plus symbols, pipe symbols or forward slashes. Rather than repeatedly list the same syntax prescription, the ... symbol is inserted after the relevant separator symbol. The actual syntax that can be repeated will be highlighted in a colour, as will the associated ... symbol and the separation character that immediately precedes it.

In the first example, the syntax that can be repeated is entirely highlighted:

```
<decimal(weight)> * <string(parameter)> , ...
```

This could be expanded to:

```
0.4*G1, 0.6*B1, 1.2*B2
```

In the second example, all rows of syntax that can be repeated are spanned a vertical line of colour:

```
 [ <string(parameter)>[<PRIOR>]
 | <string(parameter)>.d[<PRIOR> |...]
 | <string(parameter)>.e[<PRIOR> |...]
 ]
 *
 <string(attribute name)><LEVELS>
 * ...
+ ...
```

This could be expanded to:

```
G1[0.4] * att1[2,4,6] * att2[3,5,7] + G2.d[0.6|0.8] * att3 + G3[-1.7] *
att4[1,2,3]
```

**Examples**
Finally, the syntax prescription, while unambiguous, can appear very confusing. Closely examining examples is a useful way to become familiar with how the syntax is applied.

## 11.3 Design

Used to generate designs.

### 11.3.1 alg

**description:** Specifies what algorithm to use when generating efficient designs.

**values:**

```
;alg = [ swap(random=<integer>, swap=<integer>,
swaponimprov=<integer>, reset=<integer>, resetinc=
<integer>, <STOP>)
         | rsc(<STOP>)
         | rs(<STOP>)
         | rc(<STOP>)
         | sc(<STOP>)
         | r(<STOP>)
         | s(<STOP>)
         | c(<STOP>)
         | all
           | mfederov(candidates=<integer>|<string(name or
path)>, <STOP>)
```

```
        | neldermead(converge=<float>, runs=<integer>,
nointerim, seed=<integer>)
        | eval(<string(name or path)>) ]
```

where

`<STOP>` is

```
stop = [ total(<integer> [secs | mins | iterations])
       | noimprov(<integer> [secs | mins | iterations])
]
```

**default:**

If the property is not specified, an efficient design search (;eff) will use ;alg=swap.

;alg = swap(random=500, swap=1, swaponimprov=40, reset=10000, resetinc=5000) (MNL model)
;alg = swap(random=500, swap=1, swaponimprov=40, reset=10000, resetinc=5000) (RP model)
;alg = swap(random=500, swap=1, swaponimprov=40, reset=10000, resetinc=5000) (RP panel model)
;alg = swap(random=500, swap=1, swaponimprov=40, reset=10000, resetinc=5000) (EC model)
;alg = swap(random=500, swap=1, swaponimprov=40, reset=10000, resetinc=5000) (EC panel model)
;alg = swap(random=500, swap=1, swaponimprov=40, reset=10000, resetinc=5000) (RPEC model)
;alg = swap(random=500, swap=1, swaponimprov=40, reset=10000, resetinc=5000) (RPEC panel model)
;alg = swap(random=500, swap=1, swaponimprov=40, reset=10000, resetinc=5000) (model averaging)

;alg = mfederov(candidates=200000)

;alg = neldermead(converge=0.001, runs=1)

If the stop parameter is not specified, the algorithm will run indefinitely, and will only terminate when the user chooses Stop.

**comments:**

**;alg = swap**
Elaborate swapping algorithm.

- random: How many seed iterations to perform during the initial phase of complete design randomization.
- swap: How many swaps to perform for each attribute.
- swaponimprov: How many swaps to perform for each attribute after an improvement has been found by modifying that attribute.
- reset: How many iterations with no improvement must elapse before a new starting point with complete design randomization is generated.
- resetinc: How many iterations to increase 'reset' by after each complete design randomization.

**;alg = rsc (rs, rc, sc, r, s, c)**
Relabelling(r), swapping(s) and cycling(c) algorithm. And combination of the three techniques can be specified.

**;alg = all**
Attempt to evaluate all possible designs. This is only feasible for very small designs. The percentage of all possible designs evaluated so far is shown

below the trace in the output window, in addition to the current evaluation.

**;alg = mfederov**
Modified Federov algorithm.

- candidates=<integer>: The maximum size of the candidate set.
- candidates=<string(name or path)>: Load a design (as a .ngd file or an Excel file) and use all of its choice tasks for the candidate set of the Modified Federov algorithm.

**;alg = neldermead**
Performs a local-search to allocate the continuous attributes of a design. Discrete attributes will not be changed. Note: this algorithm requires that a pre-existing design be loaded using ;eval first.

- converge: minimum distance required between the best and all other candidate solutions in order to terminate. Distance is relative to attribute level lower and upper bounds.
- runs: the number of runs to perform. Each run is an independent trial and begins with an entirely new set of random allocations to the continuous attributes. The sole exception is the first run where one copy of the original design is maintained.
- nointerim: only report improved designs upon convergence, not as they are found (which is default). Duplicate designs will be reported if there is no improvement between restarts.
- seed: a number to initialize the pseudo-random number generator. This allows experiments to be repeated if so desired.

**;alg = eval**
Evaluates an existing design (does not generate or optimize).

- name or path: Either the filename of an open data file, or the complete path of a data file.

**;alg = <any algorithm>(stop = total(200 secs))**
The algorithm <any algorithm> will stop after running for 200 seconds.

**;alg = <any algorithm>(stop = total(3 mins))**
The algorithm <any algorithm> will stop after running for 3 minutes.

**;alg = <any algorithm>(stop = noimprov(10000 iterations))**
The algorithm <any algorithm> will stop if no improvement has been found for 10000 iterations.

**requirements:**

**incompatibilities:**

- Factorial designs (;fact). Factorial designs are generated with a fixed algorithm.
- Orthogonal designs (;orth). Orthogonal and orthogonal efficient designs are generated with a fixed algorithm.
- Design evaluation (;eval).
- Continuous attributes and ;alg=mfederov.
- Attribute level rejection (;reject) and any non row based algorithm (all except ;alg=mfederov).
- Attribute level requirements (;require) and any non row based algorithm (all except ;alg=mfederov).

**example(s):**

```
;alg = rs
;alg = rsc(stop = total(15 mins))
```

```
;alg = (random=500, swap=1, swaponimprov=40, reset=10000, resetinc=5000)
;alg = mfederov(candidates=200000)
```

**relevant manual sections:** Appendix 7B: Steps in generating efficient SC designs
8.6: Algorithms for generating designs in Ngene
Appendix 9B: The Nelder Mead algorithm

## 11.3.2 alts

**description:** Specifies the alternatives in the model.

**values:**

```
;alts(<string(model label)>) = <string(alternative 1 name)>*, <string(alternative 2 name)>* , ...
```

**default:** This property and its property values are mandatory.

**comments:** Names of alternatives may contain numbers, but no spaces. These names need to be used when defining the utility functions in the model property.

All alternative names that are followed by an optional asterix (*) will be treated as unlabeled. A full discussion of the checks performed on unlabeled alternatives is documented in Section 8.8.

**;alts =**
When a single model specification is present in the syntax, the alts property does not need to be qualified.

**;alts(<string(model label)>) =**
When multiple model specifications are present in the syntax, the alts property needs to be qualified with a label. This label will also be used to qualify the utility functions specified in the model property. In this way, different model specifications can have different numbers of alternatives.

**requirements:**

**incompatibilities:**
- The alternative names cannot be reserved words.

**example(s):**

```
;alts = car, train, bus
```

```
;alts(model1) = car, train, bus
;alts(model2) = train, bus
```

**relevant manual sections:** 4.2: An example design syntax: Full factorial designs

## 11.3.3 bdraws

**description:** Specifies the type and number of draws for Bayesian prior parameters.

**values:**

```
;bdraws = [ random(<integer(R)>)
          | halton(<integer(H)>)
          | sobol(<integer(S)>)
          | mlhs(<integer(M)>)
          | gauss(<integer(A)> , ...)
          ]
```

**default:** ;bdraws = halton(200) unless changed in the options dialog box.
If the property is not specified, the presence of Bayesian priors in the utility expressions will determine whether Bayesian draws are drawn.

**comments:** **;bdraws = random**
*R* pseudo-random draws.

**;bdraws = halton**
*H* quasi-random Halton draws.

**;bdraws = sobol**
*S* quasi-random Sobol draws.

**;bdraws = mlhs**
*M* draws using modified latin hypercube sampling.

**;bdraws = gauss**
Gaussian quadrature draws with *A* abscissas. One can specify a single number of abscissas which will be used for all prior parameters, or provide the number of abscissas for each prior parameter. In this case, the number of abscissas per prior parameter are specified in a comma separated list in the same order as the priors are introduced in the models.

The number of Gaussian quadrature draws is equal to the product of each prior parameter's numbers of abscissas. Thus, Gaussian quadrature might need a large number of rows when there are many prior parameters.

**requirements:** • Lack of specification of any Bayesian priors will result in a warning.

**incompatibilities:**

**example(s):**

```
;bdraws = halton(100)
;bdraws = gauss(5)
;bdraws = gauss(1,3,2,3)
```

**relevant manual sections:** 7.3: Bayesian efficient designs

## 11.3.4 block

**description:** Specifies the number of blocks in the design.

**values:**

```
;block = <integer(number of blocks)>,
         [minsum | minmax],
         [ total([<integer> mins | <integer> secs])
         | noimprov([<integer> mins | <integer> secs])
         ],
         newblocking
```

**default:** ;block = minsum, total(3 secs)
The <integer(number of blocks)> property value is mandatory.

If the property is not specified, no blocking column is generated.

**comments:**

**<integer(numBlocks)>** (compulsory)
Simultaneous orthogonal or efficient designs can be blocked, i.e. a design with *S* choice situations is divided into smaller designs with *S/<integer(* number of blocks*)>* choice situations, where <integer(number of blocks)> is the number of blocks.

For a simultaneously efficient design, the blocking column is orthogonal with all other attributes.

For other designs, the correlations between the blocking column and all other attributes will be minimized using a search procedure. The blocking column will only be assigned for these designs when the design window is first open, to prevent this calculation slowing down the efficiency optimization.

**minsum** (optional)
When assigning the blocking column using a search, minimizes the *total* correlation values between the blocking column and all of the attributes.

**minmax** (optional)
When assigning the blocking column using a search, minimizes the *maximum* correlation value between the blocking column and each of the attributes.

**total** (optional)
When assigning the blocking column using a search, spend the specified number of seconds or minutes to find the best blocking column.

**noimprov** (optional)
When assigning the blocking column using a search, accept the current blocking column when no improvement has been found for the specified number of seconds or minutes.

**newblocking** (optional)
By default, if the design contains a blocking column, the levels of that column will be preserved if an existing design is opened or evaluated with the ;eval property. In some instances, the user may wish to generate a new blocking column. For example, the blocking could be performed for a longer time period, to improve the quality of the blocking, or the number of blocks could be altered (in both instances, through changes to the ;block property). To achieve this, add the 'newblocking' property value.

**requirements:**

**incompatibilities:**

**example(s):**

- `;block = 3`
- `;block = 2, minmax, noimprov(10 secs)`
- `;block = 4, minsum, total(1 mins)`
- `;block = 4, minsum, total(10 mins), newblocking`

**relevant manual sections:** 6.2.4: Orthogonal fractional factorial designs

## 11.3.5 bseed

**description:** Specifies the random seed for the 'bdraws=random' and 'bdraws=mlhs' directives.

**values:** `;bseed = <integer>`

**default:** ;bseed = random

**comments:** If bseed is not specified, the bdraws for random and mlhs will be completely random each time the syntax is run. Otherwise, it uses the same seed each time and therefore reproduces the same output each time.

**requirements:**

**incompatibilities:**

**example(s):** `;bseed = 12345`

**relevant manual sections:** 7.3: Bayesian efficient designs

## 11.3.6 con

**description:** Specifies whether constants are to be considered when determining the efficiency of a design.

**values:** `;con`

**default:** If the property is not specified, constants are not considered in determining the design efficiency.

**comments:** Only included whenever constants are to be considered in determining the design efficiency.

**requirements:**

**incompatibilities:**

**example(s):** `;con`

**relevant manual sections:** 7.2.2: Designs for estimating MNL models

## 11.3.7 cond

**description:** Specifies conditional expressions for attribute levels.

**values:**

```
;cond:
 [ if ( <LOGICAL EXPRESSION> , <LOGICAL EXPRESSION> )
 | fractional=<decimal(fractionalSize)%
 ]
```

```
, ...
```

where

```
<LOGICAL EXPRESSION> is
 [ <VALUE>
   [ < | <= | > | >= | = | <> ]
   <VALUE>
 | <VALUE>
   =
   [<decimal(constant)> ,...]
 ]
 [ AND | OR ] ...
```

and

```
<VALUE> is
[ <string(alternative)>.<string(attribute)>
| <decimal(constant)>
| <VALUE> [ + | - ] <VALUE>
]
```

and

```
<VALUE> = [<decimal(constant)> , ...]
```

is equivalent to

```
<VALUE> = <decimal(constant)> OR ...
```

**default:**

**comments:** This property can be used for attribute levels that are conditional on other attribute levels.
If many attributes are related through the conditional expressions, memory problems may result. The solution is to specify a suitably low value of fractionalSize. See 8.2.2: Constrained designs in Ngene for more details.

**requirements:**

- New attribute levels cannot be specified in the cond property. Instead, all possible levels must be declared when the attribute is specified in the model property, and these can then be constrained by the cond property. Be careful, there is currently no check that the attribute levels you specify in the cond property were specified in the model property, and levels that are not in the later will be ignored in the former.

**incompatibilities:**

- Factorial designs (;fact).
- Orthogonal designs (;orth).
- Modified Federov algorithm (;alg=mfederov).
- Multiple designs (i.e. use of the ;fisher property)
- Attributes with a continuous specification

**example(s):**

```
;cond:
if(alt1.A = 10, alt2.B = [15,20,25])
```

**relevant manual sections:** 8.2.2: Constrained designs in Ngene

### 11.3.8 eff

**description:** Generates an efficient design, and specifies the efficiency measure to optimize on.

**values:**

```
;eff =
 <decimal(weight)> * <string(model name)>
 ([mnl | rp | rppanel | ec | ecpanel | rpec | rpecpanel
| ood],
 [d | a | b | s(<decimal(t threshold)>) | wtp(<string
(wtp label)>) | dw | aw | bw | sw(<decimal(t threshold)>
) | imbalance | none],
 [fixed | mean | median | dev | min | max])
+ ...
```

**default:** ;eff = 1 * (mnl, d, fixed)

**weight:** 1
**t threshold:** 1.96
**model name:** blank (corresponding to a single ;model property that has not been qualified with a label)
**wtp label:** blank (corresponding to a single ;wtp property that has not been qualified with a label)

**comments:** This property directs Ngene to search for the most efficient design, where the desired efficiency measure is specified through the property values.

The following efficiency measures are available:
**d:** d-error (based on the determinant of the AVC), minimized.
**a:** a-error (based on the trace of the AVC), minimized.
**b:** utility balance, maximized. Reported as a percentage, with higher percentages representing greater utility balance.
**s:** s-efficiency measure (sample size based), minimized. Optionally calculated with the user specified t threshold.
**wtp:** willingness to pay efficiency, minimised. Optimizes on a willingness to pay measure specified with the ;wtp property. If "**wtp label**" is not specified, the single unqualified ;wtp property is used to define the willingness to pay measure. If "**wtp label**" is specified, the ;wtp property qualified with "**wtp label**" is used to define the willingness to pay measure.
**dw:** d-error, maximized.
**aw:** a-error, maximized.
**bw:** attribute level balance, minimized.
**sw:** willingness to pay efficiency, maximised.
**imbalance:** a measure of attribute level imbalance. 0 is full balance (all levels appear an equal number of times), and 1 is full imbalance.
**none:** no comparison is made between two designs, and so all designs considered during the search are reported. It is strongly recommended that stopping criteria are specified for the search algorithm through the ;alg property, as the sheer number of designs that are found will quickly overwhelm Ngene.

For Bayesian designs, one efficiency measure is calculated per Bayesian draw, resulting in a set of efficiency measures *E*. These measures can be aggregated in a variety of ways:
**mean:** the mean *E*.
**median:** the median of *E*.
**dev:** the standard deviation of *E*.

**min:** the minimum value in *E*.
**max:** the maximum value in *E*.
**fixed:** the efficiency measure using a fixed prior. The mean prior is used for normally distributed Bayesian priors, and the midpoint of the upper and lower bounds is used for uniformly distributed Bayesian priors.

The efficiency measures will vary according to the type of model assumed. The following model types are available:
**mnl:** multinomial logit model.
**rp:** MMNL model. Random parameters need to be specified in the utility expressions. Any error components specified will be ignored.
**rppanel:** MMNL model accounting for panel nature. Random parameters need to be specified in the utility expressions. Any error components specified will be ignored.
**ec:** EC model. Error components need to be specified in the utility expressions. Any random parameters specified will be treated as non-random.
**ecpanel:** EC model accounting for panel nature. Error components need to be specified in the utility expressions. Any random parameters specified will be treated as non-random.
**rpec:** MMNL model with error components. Random parameters and/or error components can be specified in the utility expressions, and all will be considered.
**rpecpanel:** MMNL model with error components, accounting for panel nature. Random parameters and/or error components can be specified in the utility expressions, and all will be considered.
**ood:** optimal orthogonal designs. Only the d error can be optimised on.

Ngene allows multiple sets of utility expressions to be specified via the model property, with each set being labeled. For example:
;model(short): ...
;model(medium): ...
If this is done, the eff property must reference the correct set of utility expressions, with "model name" matching the label in the desired model property. For example:
;eff = short(d)

If only a single set of utility expressions is specified, the model property does not need to be qualified. For example:
;model: ...
If this is done, the eff property should not contain any value for "**model name** ". For example:
;eff = (d)

The specification of an efficiency measure, a Bayesian moment, a model type, and a reference to a specific set of utility expressions will result in a single efficiency value for any given design. However, multiple efficiency values can be additively combined in the ;eff property using the + operator. Prior to summation, each efficiency value added can be multiplied by " **weight**", to place greater or lesser importance on each efficiency value. Any number of individual efficiency measures can be added, although be warned that this may slow down the search considerably, especially if panel model types are specified.

**requirements:**

**incompatibilities:**
- Factorial designs (;fact).
- Orthogonal designs (;orth).
- Design evaluation (;eval).

**example(s):**

```
;eff = (mnl, s(3), fixed)
;eff = 1.5 * short(rp, d, mean) + 2 * medium(rp, d, mean) + long(rp, d, mean)
;eff = (ec, wtp(wtp_all), median)
;eff = (mnl, d) + 2 * (imbalance)
```

**relevant manual sections:** 7.2.1 Efficiency measures
8.1 Attribute level balance and fractional factorial designs

### 11.3.9 eval

**description:** Evaluates the specified data file.

**values:** `;eval = <string(name or path)>`

**default:** The property value is mandatory.

**comments:** "name or path" can be the full path of an Excel or .CSV data file. Alternatively, if the workspace is managed, "name or path" can refer to the name of a data file in the current project.

A design will be created using the information specified in the entire syntax (utility expressions, priors etc), with the levels as specified in the data file.

The data file should not contain a header row. Each row represents a single choice situation. The first column must contain a number representing the design number. If there is only one design, this must be a column of 1's. The second column must contain increasing choice situation numbers (1, 2, ...). All subsequent columns must contain the design levels, with a single column representing an attribute within an alternative. Columns in the data file will be assigned to attributes in the order that the attributes are specified in the syntax.

**requirements:**

**incompatibilities:** When ';eval' is specified, the design is generated by reading in the design levels from the specified data file. The following properties are an alternative way for instructing Ngene how to generate a design, and hence are incompatible with ';eval'.
- Factorial designs (;fact).
- Orthogonal designs (;orth).
- User specified algorithms (;alg).

**example(s):**

```
;eval = RawDesign.xls
;eval = C:\Store\RawDesign.xls
```

**relevant manual sections:** 8.7: Evaluating existing designs in Ngene

## 11.3.10 fact

**description:** Generates a full or fractional factorial design.

**values:**

```
;fact
```

**default:** N/A

**comments:**

**Full factorial designs**
To generate a full factorial design, specify ';rows=all'. Care must be taken, as large design dimensions will lead to a design with a huge number of rows in the full factorial, and Ngene will crash when it runs out of memory.

**Fractional factorial designs**
To generate a fractional factorial design, specify the number of desired rows in the fractional factorial with the ';rows' property. The design will be populated with a random subset of the full factorial design.

**Constraints**
The factorial design can be constrained with ';reject' and ';require', but not 'cond'.

**requirements:**

**incompatibilities:**

- Orthogonal designs (;orth). Correlation values can still be interrogated in the design window. The full factorial design will be orthogonal (using Pearson Product Moment, CP Coefficient, Point Biserial and J Index correlation measures). Use ';orth' in place of 'fact' to achieve orthogonality for fractional factorial designs, as they are unlikely to be orthogonal using '; fact'.
- Efficient designs (;eff). Efficient designs are merely an optimised fractional factorial design, so the ';fact' property is superfluous when an efficient design is desired. Nonetheless, all available efficiency results can be interrogated in the design window.
- User specified algorithms (;alg).
- Design evaluation (;eval).
- Blocking (;block).

**example(s):**

```
;fact
```

**relevant manual sections:** 6.2.1 Full factorial designs
6.2.2 Fractional factorial designs

## 11.3.11 fisher

**description:** Specifies the design names, model names and weights that are used to construct a Fisher matrix. Used with pivot designs and designs with covariates.

**values:**

```
;fisher(<string(fisher label)>) =
 <string(design  label)>(<string(model  label)>[<decimal
(exact weight)> | <decimal(lower weight)>:<decimal(upper
weight)>] , ...)
+ ...
```

**default:**

**comments:** Specifying multiple model labels within a single design (as in example one below) will cause a homogeneous design to be constructed. Specifying one model label per design will generate heterogeneous designs.

Currently only one fisher property can be specified. This constraint is likely to be relaxed in the future. Also, the formatted scenarios will only show the first design. Again, this constraint will be relaxed in the future.

**requirements:**

**incompatibilities:**

**example(s):**

```
;fisher(fish) = des1(small[0.33], medium[0.33], large
[0.34])               ? homogeneous design
;fisher(fish) = des1(small[0.33]) + des2(medium[0.33]) +
des3(large[0.34]) ? heterogeneous designs
```

**relevant manual sections:** 8.3.2 Pivot designs in Ngene
8.4.1 Designs with covariates

## 11.3.12 foldover

**description:** Specifies whether a fold-over design will be generated.

**values:** `;foldover`

**default:** N/A (by default no fold-over design is generated)

**comments:** A fold-over design doubles the number of choice situations, but removes all correlation between two-way interactions.

**requirements:**

- Orthogonal designs (;orth).

**incompatibilities:**

- Factorial designs (;fact).
- User specified algorithms (;alg).
- Design evaluation (;eval).
- Attribute level rejection (;reject).
- Attribute level requirements (;require).

**example(s):** `;foldover`

**relevant manual sections:** 6.2.5 Orthogonal fractional factorial designs with two-way interactions

## 11.3.13 Formatting properties

A range of properties are appended to the syntax when design matrices are formatted using the Scenario formatting system. It is strongly recommended that these properties are only modified through this system, and not directly from the syntax. Directly editing these properties may cause Ngene to crash when opening the design. Consequently, the syntax structure is not documented, and the properties are only listed for your reference.

#### 11.3.13.1 formatattributes

#### 11.3.13.2 formatchoices

#### 11.3.13.3 formatstylesheet

#### 11.3.13.4 formattable

#### 11.3.13.5 formattabledimensions

#### 11.3.13.6 formattablefooter

#### 11.3.13.7 formattableheader

#### 11.3.13.8 formattablestyle

#### 11.3.13.9 formattitle

### 11.3.14 model

**description:** Specifies the model structure.

**values:**

```
;model(<string(label)>):
 U(<string(alternative)>) =
  [ <string(parameter)>[<PRIOR>]
  | <string(parameter)>.d[<PRIOR> |...]
  | <string(parameter)>.e[<PRIOR> |...]
  ]
  *
  <string(attribute name)>[.ref | .piv | .covar]<LEVELS>
  * ...
 + ...
/ ...
```

where
<PRIOR> is

```
[ <decimal(fixed prior)>
|  (n, <decimal(Bayesian mean)>,           <decimal(Bayesian
std dev.)>   )
|  (u,  <decimal(Bayesian  lower  bound)>,  <decimal(Bayesian
upper bound)>)
| n,   [ <decimal(rp mean)>         | __ | __ ] , [ <decimal
(rp std dev.)>    | __ | __ ]
| u,   [ <decimal(rp lower bound)> | __ | __ ] , [ <decimal
(rp upper bound) | __ | __ ]
| ec, [ <decimal(ec std dev.)>    | __ | __ ]
]
```

and
<LEVELS> is

```
[  [<decimal(level)>  , ...]  ([<integer(exact  frequency)> |
<integer(low  frequency)>  -  [  <integer(high  frequency)> |
inf ] ]  , ...)
| [<string(attribute name to mimic)>]
|    [fcn([<decimal(constant)>|<string(alternative   name)>.
<string(attribute name)>]  [+|-] ...)]
|   [<decimal(lower   continuous   limit)>   :   <decimal(upper
continuous limit)>]
|   [<decimal(lower   limit)>   :   <decimal(upper   limit)>   :
<decimal(step size)>]
| [<decimal(pivot percentage)>%  , ...]
]
```

**default:**

**comments:** This is the most elaborate property to be specified in the Design command. It expresses the utility functions of each alternative (if one utility function is left out, then this alternative is considered to be a no-choice option).

Each utility function is a linear combination of parameters and attributes. Parameter names and attribute names are user-defined and may not include spaces. If for different alternatives the same parameter name is used, then this parameter is considered generic over these alternatives. Using the same attribute name in different alternatives does not have an impact (although this can be used as a shortcut: attributes in subsequent alternatives with the same name and no levels specified will assume the levels of the first attribute instance).

**Parameters and their priors**
parameter[*x*] specifies that *x* is the prior value of parameter, where *x* can be a single value for a fixed parameter, or can denote a random parameter distribution, or may denote a Bayesian prior distribution (or a combination). For some designs (such as orthogonal designs) *x* need not be specified and can therefore be omitted. For generic parameters, the prior value *x* can only be specified the first time and should be omitted in all other utility functions.

**Fixed parameters with fixed priors:**
parameter[*x*]

**Random parameters with fixed priors:**
parameter[n,*x*,*y*] for normal distribution with mean *x* and standard deviation *y* (requirements: y>=0),
parameter[u,*x*,*y*] for uniform distribution with lower bound *x* and upper bound *y*

(requirements: y>x).

**Error component with fixed prior:**
parameter[ec,y] for normal distribution with mean 0 and standard deviation *y* (requirements: y>=0).

**Fixed parameters with Bayesian priors:**
parameter [(n,*x*,*y*)] for Bayesian normal distribution with mean *x* and standard deviation *y* (requirements: y>=0),
parameter [(u,*x*,*y*)] for Bayesian uniform distribution with lower bound *x* and upper bound *y* (requirements: y>x).

**Random parameters with Bayesian priors:**
parameter [n,(u,*x1*,*y1*),(n,*x2*,*y2*)] for normal distribution with Bayesian mean distributed with a uniform distribution (with parameters *x1* and *y1*) and a Bayesian standard deviation distribution with a normal distribution (with parameters *x2* and *y2*). Other combinations of distributions can be made.

**Fixed parameters with fixed priors for a dummy or effects coded attribute:**
parameter.d[x|y|z] for specification of priors for the *first* three dummy coded levels of the associated four level attribute.
parameter.e[x|y|z] for specification of priors for the *first* three effects coded levels of the associated four level attribute.
The associated attribute does not need to have levels specified. If levels are specified, there must be one more than the number of priors, and the levels will internally be dummy or effects coded for calculations in the current model specification. Note however that in other model specifications that use the same attribute, the specified levels may be used if the parameter associated with the attribute is not dummy or effects coded. In this way, the dummy or effects coding is associated more closely with the parameter specification than the attribute specification.

**Attributes and their levels**
attribute [*x*] specifies that *x* is the range of attribute levels, where *x* denotes a discrete or continuous range of attribute levels, or may be relative attribute levels pivoted around a reference level.

**Discrete attribute levels:**
attribute[*x1*,*x2*,…]. Each level can only be specified once.

**Non-balanced discrete attribute levels:**
attribute [*x1*,*x2*,…](*y1*–*z1*,*y2*,…) is to be used in case attribute level balance is not required; then attribute level *x1* is required to appear between *y1* and *z1* times, x2 is required to appear exactly y2 times etc. To specify a maximum that equals the number of rows, specify "inf" for the maximum.

**Continuous attribute levels:**
attribute [*x1:x2*]. Specific algorithms, such as Nelder-Mead, are required to take advantage of continuous attribute levels. For other algorithms, the number of levels generated will be equal to the number of rows in the design, and the levels will be equally spaced.

**Discrete attribute levels with bounds and a step size:**
attribute[*xLower*:*xUpper*:*stepSize*] allows a large number of attribute levels to be quickly specified. The levels *xLower*, *xLower*+*stepSize*, *xLower*+2**stepSize*, ...

are utilised, until the upper bound *xUpper* is exceeded. If too few rows are specified, not all levels may be used in the design.

**Attributes that maintain the same levels as another attribute:**
attribute[other attribute] not only specifies that the levels of the other attribute be used for attribute, but also that the levels of the two attributes will be the same for any given row of any given design. This is useful for including scenarios in the experimental design.

**Attribute level functions:**
attribute[fcn(...)] will generate the attribute level by evaluating the function, which can include constants, attributes, and plus and minus operators. This is useful for specifying probabilities. Refer to Section 8.9 for more details.

**Reference and pivot attributes**
attribute.ref[<*one level*>] indicates that the attribute is a reference attribute with a single level. attribute.piv[*x1,x2,...*] is a pivot attribute, where the levels can be specified as positive or negative absolute values, or positive or negative percentages.

**Covariate attributes**
attrib.covar[<*one level*>] indicates that the attribute is a covariate.

**Interactions**
The above specification does not adequately cover interactions, especially when the interactions involve dummy coded attributes. For precise information on interactions, refer to Section 7.2.9, and Section 6.2.4.

**requirements:**

**incompatibilities:**

**example(s):**

```
;model:
U(alt1) = b1 + b2 * A[0,1,2] + b3 * B[0,1]     /
U(alt2) =      b2 * A        + b4 * C[0,1,2,3]
? b2 is generic
? b3 and b4 are alternative-specific
```

```
;model:
U(alt1) = b1[-1] + b2[2] * A[0:2]                 + b3[0.5]
* B[0,1]     /
U(alt2) =          b2    * C[1,2,3](1-3,1-3,2) + b4[0.3] *
D[0,1,2,3]
? all parameters have fixed priors,
? levels of attribute A are continuous,
? levels of C need not be balanced
? (level 1 should appear 1 to 3 times, level 3 should
appear exactly twice)
```

```
;model:
U(alt1) = b1[-1] + b2[n,1,(u,0,0.2)] * A[0,1,2] + b3
[(n,0.5,0.1)] * B[0,1]    /
U(alt2) =           b2                    * A[1,2,3] + b4
[n,0.3,0.1]   * C[0,1,2,3]
? b1 is a fixed parameter with a fixed prior,
? b2 is a random parameter with a Bayesian standard
deviation,
? b3 is fixed parameter with a Bayesian prior,
? b4 is a random parameter with fixed priors.
```

**relevant manual sections:** 4.2: An example design syntax: Full factorial designs

### 11.3.15 orth

**description:** Generates an orthogonal design.

**values:**

```
;orth = [ sim
        | seq
        | seq2
        | ood ]
```

**default:** ;orth = sim

**comments:** Generates a fully orthogonal design, with no Pearson Product Moment correlations between the levels of the attributes. The actual attribute pairs for which no correlations will exist depends on the type of orthogonality specified. There may not exist an orthogonal design with the number of rows specified with ';rows'. Hence, the user-defined number of rows will be used as a lower bound.

Orthogonal designs may be blocked using ';block', and a foldover column may be added using ';foldover'.

**;orth = sim**
Maintains orthogonality within and across all alternatives. This may require many rows to be generated in the design.

**;orth = seq**
Maintains orthogonality only within each alternative. Each alternative must have the same attributes with the same levels.

**;orth = seq2**
Maintains orthogonality only within each alternative. Each alternative may have different attributes with different numbers of levels.

**;orth = ood**
Generates a design using the OOD efficiency measure and the OOD algorithm.

For greater detail on the various types of orthogonal design, including their generation within Ngene, refer to Chapter 6.

**Efficient orthogonal designs**
Specify ';eff' in addition to ';orth' to generate efficient orthogonal designs. A custom algorithm will be implemented that spans all possible orthogonal designs.

**requirements:**

**incompatibilities:**

- Factorial designs (;fact). Orthogonal designs are fractional factorial designs, and full factorial designs are orthogonal, so ';fact' is superfluous when ';orth' is specified.
- User specified algorithms (;alg).
- Design evaluation (;eval).
- Attribute level rejection (;reject).
- Attribute level requirements (;require).

**example(s):**

```
;orth = seq
```

**relevant manual sections:** 6.2.4 Orthogonal fractional factorial designs

## 11.3.16 prec

**description:** Specifies the precision of all numbers reported in Ngene.

**values:**

```
;prec = <integer>
```

**default:** ;prec = 6 unless changed in the options dialog box.

**comments:** Naturally, all calculations are made with maximum precision, and outputs are only rounded immediately prior to being reported.

**requirements:**

**incompatibilities:**

**example(s):**

```
;prec = 8
```

**relevant manual sections:**

## 11.3.17 rdraws

**description:** Specifies the type and number of draws for random prior parameters.

**values:**

```
;rdraws = [ random(<integer(R)>)
          | halton(<integer(H)>)
          | sobol(<integer(S)>)
          | mlhs(<integer(M)>)
          | gauss(<integer(A)> , ...)
          ]
```

**default:** ;rdraws = halton(200) unless changed in the options dialog box.
If the property is not specified, the presence of random priors in the utility expressions will determine whether random draws are drawn.

**comments:** **;rdraws = random**
*R* pseudo-random draws.

**;rdraws = halton**
*H* quasi-random Halton draws.

**;rdraws = sobol**
*S* quasi-random Sobol draws.

**;rdraws = mlhs**
*M* draws using modified latin hypercube sampling.

**;rdraws = gauss**
Gaussian quadrature draws with *A* abscissas. One can specify a single number of abscissas which will be used for all prior parameters, or provide the number of abscissas for each prior parameter. In this case, the number of abscissas per prior parameter are specified in a comma separated list in the same order as the priors are introduced in the models.

The number of Gaussian quadrature draws is equal to the product of each prior parameter's numbers of abscissas. Thus, Gaussian quadrature might need a large number of rows when there are many prior parameters.

**requirements:**

- Lack of specification of any random priors will result in a warning.

**incompatibilities:**

**example(s):**

```
;rdraws = halton(100)
;rdraws = gauss(5)
;rdraws = gauss(1,3,2,3)
```

**relevant manual sections:** 7.2.3: Designs for estimating random parameters models

## 11.3.18 reject

**description:** Specifies which combinations of attribute levels in choice situations should be rejected.

**values:**

```
;reject:
  <VALUE>
  [ < | <= | > | >= | = | <> ]
  <VALUE>
  [ AND | OR ] ...
, ...
```

`<VALUE>` is

```
[ <string(alternative)>.<string(attribute)>
| <decimal(constant)>
| <VALUE> [ + | - ] <VALUE>
]
```

**default:** When the property is specified, a value is mandatory.

**comments:** This property will prevent certain combinations of attribute levels from appearing in the same row of the design. Logical expressions are specified in this property, and if they evaluate to true for any potential row, the row is rejected and cannot be placed in the design. In this way, it is possible to find a constrained design. (An alternative approach is to use the ;require property, where all logical expressions must evaluate to true when applied to all rows of the design.) The ;reject property will only work on design search strategies that modify designs by changing an entire row: factorial designs and the modified Federov algorithm. To specify constraints with the swapping algorithm, use the ;cond property instead.

Any number of independent logical expressions can be specified, although care must be taken. A large number of constraints will reduce the number of rows available to populate the design with (i.e. the candidate set size). The number of rows that do not violate the constraints should be at least equal to the number of rows for a factorial design, and greater than the number of rows for the modified Federov algorithm (to allow some row exchange to take place).

**requirements:**

- Factorial designs (;fact) or a row based search algorithm (;alg=mfederov).

**incompatibilities:**

- Orthogonal designs (;orth).
- Non row based search algorithms (all except ;alg=mfederov).

**example(s):**

```
;reject:
alt1.A + 1 > alt2.B ,
alt1.B = alt1.C and alt1.D <> 0
```

**relevant manual sections:** 8.2.2: Constrained designs in Ngene

## 11.3.19 rep

**description:** The number of draws to use in the sample of a panel based model.

**values:**

```
;rep = <integer>
```

**default:** When the property is specified, a value is mandatory.
When the property is not specified, the default is 200.

**comments:**

**requirements:**

**incompatibilities:**

**example(s):**

```
;rep = 500
```

**relevant manual sections:** 7.2.3: Designs for estimating random parameters models

## 11.3.20 require

**description:** Specifies attribute level conditions that must be met for a choice situation to be acceptable in the design.

**values:**

```
;require:
  <VALUE>
  [ < | <= | > | >= | = | <> ]
  <VALUE>
  [ AND | OR ] ...
```

```
, ...
```

`<VALUE>` is

```
[ <string(alternative)>.<string(attribute)>
| <decimal(constant)>
| <VALUE> [ + | - ] <VALUE>
]
```

**default:** When the property is specified, a value is mandatory.

**comments:** This property will require that certain attribute level conditions be met for a choice situation to be acceptable in the design. Logical expressions are specified in this property, and all expressions must evaluate to true for a row to be placed in the design. (An alternative approach is to use the ;reject property, where if any logical expression evaluates to true for any potential row, the row is rejected and cannot be placed in the design.) The ;require property will only work on design search strategies that modify designs by changing an entire row: factorial designs and the modified Federov algorithm. To specify constraints with the swapping algorithm, use the ;cond property instead.

Any number of independent logical expressions can be specified, although care must be taken. A large number of constraints will reduce the number of rows available to populate the design with (i.e. the candidate set size). The number of rows that meet the conditions specified in the ;require property should be at least equal to the number of rows for a factorial design, and greater than the number of rows for the modified Federov algorithm (to allow some row exchange to take place).

**requirements:**

- Factorial designs (;fact) or a row based search algorithm (;alg=mfederov).

**incompatibilities:**

- Orthogonal designs (;orth).
- Non row based search algorithms (all except ;alg=mfederov).

**example(s):**

```
;require:
alt1.A + 1 > alt2.B ,
alt1.B < alt1.C
```

**relevant manual sections:** 8.2.2: Constrained designs in Ngene

## 11.3.21 rows

**description:** Specifies the number of choice situations.

**values:**

```
;rows = [ <integer>
        | all
        ]
```

**default:** This property and its property value are mandatory.

**comments:** **;rows = <integer>**
Specifies the exact number of choice situations to be generated. There may not exist an orthogonal design with the specified number of rows. Hence, if orthogonal designs are specified with ';orth', the user-defined number of rows will be used as a lower bound.

**;rows = all**
This option is only available when a factorial design is specified with ';fact'. The maximum number of choice situations will be generated (i.e., the full factorial). Care must be taken, as large design dimensions will lead to a design with a huge number of rows in the full factorial, and Ngene will crash when it runs out of memory.

Specification of the ';foldover' property will double the number of choice situations specified with ';rows'.

**requirements:**

- ';rows = all' requires that ';fact' be specified.
- <integer> cannot exceed the size of the full factorial if the ';fact' property is specified.

**incompatibilities:**

**example(s):** `;rows = 12`

**relevant manual sections:** 4.2: An example design syntax: Full factorial designs

### 11.3.22 rseed

**description:** Specifies the random seed for the 'rdraws=random' and 'rdraws=mlhs' directives.

**values:** `;rseed = <integer>`

**default:** random

**comments:** If rseed is not specified, the rdraws for random and mlhs will be completely random each time the syntax is run. Otherwise, it uses the same seed each time and therefore reproduces the same output each time.

**requirements:**

**incompatibilities:**

**example(s):** `;rseed = 12345`

**relevant manual sections:** 7.2.3: Designs for estimating random parameters models

### 11.3.23 start

**description:** Uses the specified design as the starting design for an efficient design search.

**values:** `;start = <string(name or path)>`

**default:** When the property is specified, a value is mandatory.

**comments:** When performing an efficient design search, the seed design will be loaded from the data file specified by the ;eval property.

"name or path" can be the full path of an Excel or .CSV data file. Alternatively, if the workspace is managed, "name or path" can refer to the name of a data file in the current project.

The starting design will be created using the information specified in the entire syntax (utility expressions, priors etc), with the levels as specified in the data file.

The data file should not contain a header row. Each row represents a single choice situation. The first column must contain a number representing the design number. If there is only one design, this must be a column of 1's. The second column must contain increasing choice situation numbers (1, 2, ...). All subsequent columns must contain the design levels, with a single column representing an attribute within an alternative. Columns in the data file will be assigned to attributes in the order that the attributes are specified in the syntax.

**requirements:**
- Efficient design search must be specified (;eff).

**incompatibilities:**

**example(s):**
- `;start = RawDesign.xls`
- `;start = C:\Store\RawDesign.xls`

**relevant manual sections:** 8.6: Algorithms for generating designs in Ngene

## 11.3.24 store

**description:** Specifies how many designs to store in memory during a search.

**values:**

```
;store = [ <integer>
         | all
         ]
```

**default:** 10

**comments:**

**;store = <integer>**
Retains the most recent <integer> designs in memory during a search, plus the first design.

**;store = all**
Retains all designs in memory during a search, but the user must accept the risk of memory issues.

Refer to the Options dialog box for more information.

**requirements:**

**incompatibilities:**

**example(s):** `;store = 15`

**relevant manual sections:** 3.4: Output window

### 11.3.25 trimdist

**description:** Additionally reports Bayesian efficiency measures using a subset of all Bayesian draws.

**values:** `;trimdist = <decimal number(low)>, <decimal number(high)>`

**default:** The two property values are mandatory.
If the property is not specified, the additional Bayesian efficiency measure outputs are not reported.

**comments:** This allows a second set of Bayesian efficiency measures to be reported using a subset of all available Bayesian draws.

For each efficiency measure, L draws with the lowest efficiency measures are removed, as are H draws with the highest efficiency measures,
where:
L = Round(<decimal number(low)> / <total number of draws>)
H = Round(<decimal number(high)> / <total number of draws>).
Only the remaining draws are used to calculate the various Bayesian efficiency moments (mean, minimum, maximum, standard deviation, median). The draws discarded may vary from one efficiency measure (e.g. d, a) to another.

**requirements:**
- Bayesian priors must be specified in the utility expressions for this to be useful.

**incompatibilities:**

**example(s):** `;trimdist = 10, 10`

**relevant manual sections:**

### 11.3.26 wtp

**description:** Specifies a willingness to pay expression that is used to generate a willingness to pay efficiency measure.

**values:** `;wtp(<string(model label)>) = <string(wtp label)>( [ * | <decimal(weight)> * <string(parameter)> , ... ] / <string(cost parameter)> )`

**default:** When the property is specified, a value is mandatory.

**comments:** Multiple wtp efficiency measures can be generated, so long as each is labelled with a unique "wtp label". All specified parameters must exist in the associated model (either the model specification labeled with "model label", or otherwise the default unlabeled model if the wtp property is unqualified).

If a star is specified in the numerator, all non-cost parameters will be included in the wtp efficiency calculation. Alternatively, individual parameters can be specified and weighted with "weight".

**requirements:**

**incompatibilities:**

**example(s):**

```
;wtp(m1) = wtp1(0.4*G1, 0.6*B1 / G2),  ? weighted
           wtp2(* / G2)                 ? all non-cost
parameters in the numerator
```

**relevant manual sections:** 7.2.1 Efficiency measures

## 11.4 Reserved words

There are a number of words that are reserved by Ngene, and may not be used for user defined variables such as the names of alternatives, attributes, priors. These words are listed below.

- Any of the property names listed in this chapter (alg, rows, model, etc)
- Design
- Any word that contains the following symbols: ? ; $ : = , . | ( ) [ ] * + -
- Any word that contains only numbers

It is recommended that user defined variable names consist only of alpha-numeric characters, and that all other symbols be left out of the names. Failure to observe this might lead to unexpected error messages or software crashes.

# Chapter 12

# Endnotes

# 12 Endnotes

1: Labeled choice experiments involve studies where the names of the alternatives on offer convey meaning to the respondents beyond the order in which they are shown to respondents (e.g., the alternatives may be labeled as car, bus and train). In unlabeled choice experiments, the names of the alternatives are only meaningful in so far as they relate the order of the alternative as shown to the respondent (e.g., Option A, Option B, etc.). In the later case, each alternative may actually represent a car or a bus or a train in terms of the attribute levels shown to the respondent, but the fact that the alternative resembles one of these modes is not explicitly stated to the respondent. An exception to this rule exists where the different alternatives are treated as an attribute in the experiment. Also, in many SC experiments, a type or brand of alternative is often mentioned in the scenario descriptor of the task. In such cases, all the alternatives represent different versions of the same type or brand (e.g., Option A, Option B, etc., represent different alternative buses).

2: A degree of freedom is defined here as the total number of parameters (excluding the constants), plus 1. All constants are accounted for in the "plus 1".

3: For example, the authors once constructed a survey where the two alternatives represented different potential dates. One attribute in the experiment was that the potential date either had children or did not. Because the design required that one potential date always had children whilst the other did not, problems arose, particularly with younger respondents, who always selected the date without children. This occurred to the point where no information could be gained on the other attributes of the design.

4: The term asymptotic refers to the fact that it is consistent in large samples, or it is representative as an average for small samples when the survey would be repeated many times.

5: The assumption of single respondent is just for convenience and comparison reasons and does not have any further implications. Any other sample size could have been used, but it is common in the literature to base it on a single respondent.

6: The theoretical lowest rate of convergence for quasi-random MC simulation is $O((\ln^K R) / R)$, which depends on the number of dimensions, $K$, such that in theory quasi-random MC simulation can become quite slow for higher dimensions. The fastest theoretical rate of convergence is $O(1/R)$. In practice, the rate of convergence seems to be much closer to this faster rate, even for higher dimensions.

7: As an example, consider the 5th draw using 2 (the first prime number) as base. Then $r = 5$ can be expressed using three digits as 101 in base 2, because $5 = 1.2^0 + 0.2^1 + 1.2^2$. The 5th draw is then given by $1.2^{-0-0} + 0.2^{-1-1} + 1.2^{-2-1} = 0.5 + 0 + 0.125 = 0.625$.

8: For example, suppose that the first parameter has two abscissas and the second parameter has three. Let $\tilde{\beta}_1^{(1)}$ and $\tilde{\beta}_1^{(2)}$ denote the abscissas for the first parameter and $\tilde{\beta}_2^{(1)}$, $\tilde{\beta}_2^{(2)}$ and $\tilde{\beta}_2^{(3)}$ the abscissas of the second parameter. Then the draws for $\tilde{\beta}$ will be $(\tilde{\beta}_1^{(1)},\tilde{\beta}_2^{(1)})$, $(\tilde{\beta}_1^{(1)},\tilde{\beta}_2^{(2)})$, $(\tilde{\beta}_1^{(1)},\tilde{\beta}_2^{(3)})$, $(\tilde{\beta}_1^{(2)},\tilde{\beta}_2^{(1)})$, $(\tilde{\beta}_1^{(2)},\tilde{\beta}_2^{(2)})$ and $(\tilde{\beta}_1^{(2)},\tilde{\beta}_2^{(3)})$, hence 6 draws in total.

9: The minimum number of abscissas is typically two, such that with 10 random parameters, the minimum number of draws possible using Gaussian quadrature is $2^{10}$ = 1,024. Using three abscissas per random parameter increases this number to $3^{10}$ = 59,049.

10: The assumption of single respondent is just for convenience and comparison reasons and does not have any further implications. Any other sample size could have been used, but it is common in the literature to normalize it to a single respondent.

11: If Monte Carlo simulations are used rather than the true analytical second derivatives to calculate the AVC matrix for each design matrix, the amount of computing time required may be such that at most only a few hundred or so possible designs may be explored, particularly for more advanced models such as the MMNL model using Bayesian prior parameter distributions. For this reason, using the true analytical second derivatives for the specified model is preferred, yet even so, it is still unlikely that for designs of even a moderate size, all possible designs can be evaluated.

# Chapter 13

# References

# 13 References

Adamowicz, W. and P. Boxall (2001) Future Directions of Stated Choice Methods for Environment Valuation, *paper prepared for: Choice Experiments: A New Approach to Environmental Valuation*, April, London, England.

Anderson, D.A. and J.B. Wiley (1192) Efficient Choice Set Designs for Estimating Available Cross-Effects Models, *Marketing Letters*, 3(4), 357-370.

Ben-Akiva, M. and S.R. Lerman (1985) Discrete Choice Analysis: Theory and Application to Travel Demand, MIT Press, USA.

Bateman, I., Carson, R.T., Day, B., Hanemann, M., Hanley, N., Hett, T., Jones-Lee, M., Loomes, G., Mourato, S., Ozdemiroglu, E., Pearce, D.W.,
Sugden, R., and Swanson, J. (2003) Economic Valuation with Stated Preference Techniques: A Manual (in association with the DTLR and DEFRA), Edward Elgar.

Batsell, R. and Louviere, J.J. (1991) Experimental Analysis of Choice, *Marketing Letters*, 2(3), 199-214.

Bennett, J. and Blamey, R. (2001) The Choice Modelling Approach to Environmental Valuation, Edward Elgar.

Bhat, C.R. (2001), “Quasi-random maximum simulated likelihood estimation of the mixed multinomial logit model,” *Transportation Research B*, 35(7), 677-693.

Bhat, C.R. (2003), “Simulation estimation of mixed discrete choice models using randomized and scrambled Halton sequences,” *Transportation Research B*, 37(9), 837-855.

Bliemer, M.C. and Rose, J.M. (2009) Efficiency And Sample Size Requirements For Stated Choice Experiments, *Transportation Research Board Annual Meeting*, Washington DC January.

Bliemer, M.C.J, Rose, J.M. and Hensher, D.A. (2009) Constructing efficient stated choice experiments allowing for differences in error variances across subsets of alternatives, Transportation Research Part B, 43(1), 19-35.

Bliemer, M.C.J., Rose, J.M. & Hess, S. (2008) Approximation of Bayesian Efficiency in Experimental Choice Designs, *Journal of Choice Modelling*, 1(1), 98-127.

Bliemer, M.C.J. and Rose, J.M. (2006) Designing Stated Choice Experiments: State-of-the-art, paper presented at the 11th International Conference on Travel Behaviour Research, Kyoto, Japan, August.

Bliemer, M.C.J. and J.M. Rose (2005a) Efficiency and Sample Size Requirements for Stated Choice Studies. Report ITLS-WP-05-08, Institute of Transport and Logistics Studies, University of Sydney.

Bliemer, M.C.J., and J.M. Rose (2005b) Efficient Designs for Alternative Specific Choice Experiments. Report ITLS-WP-05-05, Institute of Transport and Logistics Studies, University of Sydney.

Burgess, L. and Street, D.J. (2005) Optimal designs for choice experiments with asymmetric attributes, *Journal of Statistical Planning and Inference*, 134, 288-301.

Carlsson, F. and P. Martinsson (2002) Design Techniques for Stated Preference Methods in Health Economics. *Health Economics* **12**, 281-294.

Carson, R., Louviere, J.J., Anderson, D., Arabie, P., Bunch, D., Hensher, D.A, Johnson, R., Kuhfeld, W., Steinberg, D., Swait, J., Timmermans, H., and Wiley, J. (1994) Experimental Analysis of Choice. *Marketing Letters*, 5 (October), pp351-367.

Chaloner, K. and Verdinelli, I. (1995) Bayesian Experimental Design: A Review, *Statistical Science*, 10(3), 273-304.

Collins, A.T., Bliemer, M.C.J. and Rose, J.M. (2014) Constrained stated choice experimental designs. Report ITLS-WP-14-22, Institute of Transport and Logistics Studies, University of Sydney.

Cook, R.D., and Nachtsheim, C.J. (1980) A comparison of algorithms for constructing exact *D*-optimal designs. *Techometrics* **22**, 315-324.

Dhar, R. (1997) Consumer Preference for a No-Choice Option, *Journal of Consumer Research*, 24, 215-231.

Dhar, R. and Simonson, I. (2003) The effect of forced choice on choice, *Journal of Marketing Research*, May, 146-160.

El Helbawy, A.T. and Bradley, R.A. (1978) Treatment Contrasts in Paired Comparisons: Large-Sample Results, Applications and Some Optimal Designs, *Journal of the American Statistical Association* **73**, 831-839.

Fang, K.-T.,, Wang, Y. (1994) Number-Theoretic Methods in Statistics, Chapman and Hall, London.

Ferrini, S. and Scarpa, R. (2007) Designs with a-priori information for nonmarket valuation with choice-experiments: a Monte Carlo study, *Journal of Environmental Economics and Management*, **53**, 342-363.

Fowkes, A.S. (2000) Recent developments in state preference techniques in transport research, in Ortuzar, J de D. (ed.), Stated Preference Modelling Techniques, PTRC Education and Research Services Ltd, 37-52.

Garrido, R.A. (2003) Estimation performance of low discrepancy sequences in stated preferences, paper presented at the 10th International Conference on Travel Behaviour Research, Lucerne, Switzerland.

Grasshoff, U. and Schwabe, R. (2007) Optimal design for the Bradley–Terry paired comparison model, Statistical Methods and Applications, 17(3) 275-289.

Johnson, F.R., Kanninen, B.J. and Bingham, M. (2006) Experimental Design For Stated Choice Studies, in Kanninen, B.J. (Ed.) *Valuing Environmental Amenities Using Stated Choice Studies: A Common Sense Approach to Theory and Practice*, Springer, the Netherlands, p159-202.

Kontoleon, A. and 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.

Halton, J. (1960) On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals, *Numerische Mathematik*, 2, 84-90.

Hahn, G.J., and S.S. Shapiro (1966) A catalog and computer program for the design and analysis of orthogonal symmetric and asymmetric fractional factorial experiments. General Electric

Research and Development Center, Schenectady, NY, USA.

Hensher, D.A. and Barnard, P.O. (1990) The Orthogonality Issue in Stated Choice Designs, in Fischer, M., Nijkamp, P. and Papageorgiou, Y. (Eds.), Spatial Choices and Processes, North-Holland, Amsterdam; 265-278.

Hensher, D.A., and Smith, N.A. (1984) Automobile classification and demand modelling, Transport Reviews, 4(3), 245-271.

Hensher, D.A., Rose, J.M. and Greene, W.H. (2005) *Applied Choice Analysis: A Primer*, Cambridge University Press, UK.

Hess, S., Train, K.E. and Polak, J.W. (2005), “On the use of a Modified Latin Hypercube Sampling (MLHS) approach in the estimation of a Mixed Logit model for vehicle choice,” *Transportation Research B*, 40(2), 147-163.

Huber, J. and K. Zwerina (1996) The Importance of Utility Balance in Efficient Choice Designs. *Journal of Marketing Research* **33**, 307-317.

Kanninen, B.J. (1993a). Optimal experimental design for double bounded dichotomous choice contingent valuation, *Land Economics*, 69, 138–146.

Kanninen, B.J. (1993b). Design of sequential experiments for CV studies, *Journal of Environmental Economics and Management*, 25, 1–11

Kanninen, B.J. (2002) Optimal Design for Multinomial Choice Experiments. *Journal of Marketing Research* **39**, 214-217.

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

Khuel, R.O., (1994), Statistical Principles of Research Design and Analysis 2nd Ed., Duxbury Press.

Lanscar, E. and Louviere J.J. (2006) Deleting ‘irrational’ responses from discrete choice experiments: a case of investigating or imposing preferences?, *Health Economics*, **15**, 797–811.

Lanscar, E., Louviere, J.J. and Flynn, T. (2006) Several methods to investigate relative attribute impact in stated preference experiments, *Social Science and Medicine*, **64**(8), 1738-1753.

Louviere, J.J. and D.A. Hensher (1983) Using Discrete Choice Models with Experimental Design Data to Forecast Consumer Demand for a Unique Cultural Event. *Journal of Consumer Research* **10**, 348-361.

Louviere, J.J., D.A. Hensher, and J.D. Swait (2000) *Stated Choice Methods—Analysis and Application*. Cambridge University Press, UK.

Louviere, J.J., and G. Woodworth (1983) Design and analysis of simulated consumer choice or allocation experiments: an approach based on aggregated data. *Journal of Marketing Research* **20**, 350-367.

McFadden, D. (1974) Conditional logit analysis of qualitative choice behaviour. In: P. Zarembka (ed.) *Frontiers of Econometrics*. Academic Press, New York, 105-142.

Niederreiter, H. (1992) Random number generation and quasi-Monte Carlo methods, CBMS-NFS Regional Conference Series in Applied Mathematics, 63, SIAM, Philadelphia, PA.

Ortúzar, J. de D. and Willumsen, L.G. (2001), Modelling Transport, 3rd ed., John Wiley and Sons, Chichester.

Revelt, D. and Train, K. (1998) Mixed logit with repeated choices: households' choices of appliance efficiency level, *The Review of Economics and Statistics*, 647-657.

Rose, J.M. and Bleimer, M.C.J. (2008) Stated Preference Experimental Design Strategies, in Hensher, D.A. and Button, K.J. (eds) *Handbook of Transport Modelling*, Elsevier, Oxford, Ch 8, 151-180.

Rose, J.M. and Bliemer, M.C. (2006) Designing Efficient Data for Stated Choice Experiments, accepted for presentation at 11th International Conference on Travel Behaviour Research - Kyoto, August 16-20, 2006, Japan.

Rose, J.M., and M.C.J. Bliemer (2005a) Constructing efficient choice experiments. Report ITLS-WP-05-07, Institute of Transport and Logistics Studies, University of Sydney.

Rose, J.M. and M.C.J. Bliemer (2005b) Sample optimality in the design of stated choice experiments. Report ITLS-WP-05-09, Institute of Transport and Logistics Studies, University of Sydney.

Rose, J.M., Bliemer, M.C.J, Hensher, D.A., and Collins, A.C. (2008) Designing Efficient Stated Choice Experiments Involving Respondent Based Reference Alternatives, *Transportation Research Part B*, 42(4), 395-406.

Rose, J.M., Scarpa, R. and Bliemer, M.C.J (2009) Incorporating model uncertainty into the generation of efficient stated choice experiments: A model averaging approach, *International Choice Modelling Conference*, March 30-April 1, Yorkshire U.K.

Sándor, Z. and Train, K. (2004), "Quasi-random simulation of discrete choice models," *Transportation Research B*, 38 (November), 313-327.

Sándor, Z., and M. Wedel (2001) Designing Conjoint Choice Experiments Using Managers' Prior Beliefs. *Journal of Marketing Research* **38**, 430-444.

Sándor, Z., and M. Wedel (2002) Profile Construction in Experimental Choice Designs for Mixed Logit Models, *Marketing Science* **21**(4), 455-475.

Sándor, Z., and M. Wedel (2005) Heterogeneous conjoint choice designs. *Journal of Marketing Research* **42**, 210-218.

Scarpa, R., S. Ferrini, and K.G. Willis (2005) Performance of error component models for status-quo effects in choice experiments. In: *Applications of simulation methods in environmental and resource economics*, Springer, 247–274.

Scarpa, R. and Rose, J.M. (2008) Designs efficiency for non-market valuation with choice modelling: how to measure it, what to report and why, submitted to *Australian Journal of Agricultural and Resource Economics*, 52(3), 253-282.

Starmer, C. (2000) Development in non-expected utility theory: the hunt for a descriptive theory of choice under risk. *Journal of Economic Literature* **18**, 332-382.

Stoer, J. and Bulirsch, R. (2002) Introduction to Numerical Analysis. Springer-Verlag, New York, 3rd edition

Street, D.J., D.S. Bunch, and B.J. Moore (2001) Optimal designs for 2k paired comparison experiments. *Communications in Statistics, Theory, and Methods* **30**(10), 2149-2171.

Street, D.J., and L. Burgess (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.

Street, D.J., L. Burgess, and J.J. Louviere (2005) Quick and easy choice sets: Contructing optimal and nearly optimal stated choice experiments. *International Journal of Research in Marketing* **22**, 459-470.

Toner, J.P., Clark, S.D., Grant-Muller, S.M. and Fowkes, A.S. (1999) Anything you can do, we can do better: a provocative introduction to a new approach to stated preference design, *WCTR Proceedings*, **3**, Antwerp, 107-120.

Train, K. (2003) *Discrete Choice Methods with Simulation*. Cambridge University Press, UK.

Tuffin, B. (1996) On the use of low-discrepancy sequences in Monte Carlo methods, *Monte Carlo Methods and Applications*, 2(4), 295-320.

Tversky, A. and E. Shafir (1992) Choice Under Conflict: The Dynamics of Deferred Decision, *Psychological Science*, 6, 358-361.

Yu, J., Goos, P. and Vandebroek, M. (2008). Comparing different approaches for approximating the integrals involved in the Bayesian optimal design of choice experiments. Technical Report.

Yu, J., Goos, P. and Vandebroek, M. (2009) Efficient conjoint choice designs in the presence of respondent heterogeneity, *Marketing Science*, 28(1), 122-135.

Van der Corput, J.G. (1935) Verteilungsfunktionen. Nederl. Akad. Wetensch., Proc. Ser. B (Amsterdam), 38, 813-821.

Winiarski, M., (2003) Quasi-Monte Carlo Derivative Valuation & Reduction of Simulation Bias, M. Sc. Thesis, Royal Institute of Technology (KTH), Sweden.