

# Discrete Choice Models (DCM): An Object-Oriented Package for Ox

Matias Eklöf  
Department of Economics  
Uppsala University,  
Sweden

Melvyn Weeks  
Faculty of Economics and Politics  
University of Cambridge  
Sidgwick Avenue  
Cambridge, UK

Tuesday, June 8, 2004

## Abstract

DCM (Discrete Choice Models) is a package, written in Ox, for estimating a class of discrete choice models. DCM represents an important development for both the OxMetric and, more generally, microeconomic computing environment in making available a broad range of discrete choice models, including standard binary response models, with notable extensions including conditional mixed logit, mixed probit, multinomial probit, and random coefficient ordered choice models. Developed as a derived class of `ModelBase`, users may access the functions within DCM by either writing Ox programs which create and use an object of the DCM class, or use the program in an interactive fashion. We demonstrate the capabilities of DCM by using a number of applications from both the discrete choice literature. This document will serve as a manual for DCM.

**JEL Classification:** C20; C25; C87; D00.

**Key Words:** Discrete Choice Models, mixed logit, multinomial probit, ordered probit, Ox.

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Disclaimer . . . . .	4
1.2	Availability and citation . . . . .	4
1.3	Installation . . . . .	4
<b>2</b>	<b>Overview</b>	<b>5</b>

<b>3</b>	<b>Data Input and Transformation</b>	<b>6</b>
3.1	Panel data . . . . .	8
3.2	Loading data . . . . .	8
3.3	Transforming between data structures . . . . .	8
3.4	Data manipulation and variable selection . . . . .	9
3.4.1	Creating Variables . . . . .	9
3.4.2	Selecting Variables . . . . .	10
<b>4</b>	<b>Model Specification</b>	<b>12</b>
4.1	Some comments on model specification . . . . .	15
<b>5</b>	<b>Estimation</b>	<b>16</b>
<b>6</b>	<b>Post-estimation</b>	<b>17</b>
6.1	Output . . . . .	17
6.2	Testing . . . . .	17
<b>7</b>	<b>Examples</b>	<b>17</b>
7.1	Conditional, Nested, and Mixed logit models . . . . .	17
7.2	Multinomial probit models . . . . .	22
<b>8</b>	<b>Monte Carlo experiments using Generate()</b>	<b>24</b>
8.1	Example: Ordered probit models . . . . .	27
<b>9</b>	<b>DCM in OxPack for GiveWin</b>	<b>28</b>
9.1	Limitations in OxPack implementation . . . . .	30
9.2	Model Type . . . . .	31
9.3	Variable selection and setting random coefficient distributions . . . . .	31
9.4	Estimation and model options . . . . .	32
<b>10</b>	<b>Notes, remarks, and history</b>	<b>33</b>
10.1	Bug reports and version history . . . . .	33
10.2	Future plans . . . . .	33
<b>A</b>	<b>Technical appendix</b>	<b>33</b>
A.1	Inference on data structure . . . . .	33
A.2	Covariance matrix Specification in Multinomial Probit Models. . . . .	35
A.3	Estimated standard errors . . . . .	37
<b>B</b>	<b>DCM member functions</b>	<b>38</b>
B.1	Exported member functions . . . . .	38
B.2	Non-exported member functions . . . . .	49

## List of Tables

1	First example of DCM code . . . . .	6
2	Example of "multiple rows" structured database . . . . .	7
3	Example of "single row" structured database . . . . .	7
4	<code>Select(<i>iGroup</i>, <i>asVar</i>)</code> options . . . . .	10
5	<code>SetModel(<i>iModel</i>)</code> options . . . . .	12
6	Overview of model specifications . . . . .	13
7	<code>SetCoeffDist(<i>iDist</i>, <i>asVar</i>, <i>iGroup</i>)</code> options . . . . .	14
8	<code>SetErrDist(<i>iErr</i>)</code> options . . . . .	15
9	<code>SetRandom(<i>iRandom</i>, <i>cR</i>)</code> options . . . . .	15
10	<code>SetAlgorithm(<i>iAlgorithm</i>)</code> options . . . . .	16
11	<code>SetStdErr(<i>iStdErr</i>)</code> options . . . . .	16
12	Example A: Conditional logit with post-estimation testing . . . . .	18
13	Example A: Conditional logit, output fragments . . . . .	19
14	Example B: Nested logit with variable transformations . . . . .	20
15	Example B: Nested logit, output fragments . . . . .	21
16	Example C: Mixed logit with "single row" data structure . . . . .	22
17	Example C: Mixed logit, output fragments . . . . .	23
18	Example D: Conditional Logit and Multinomial probit models . . . . .	25
19	Example D: Multinomial models. Automated LaTeX output . . . . .	26
20	Example E: Ordered probit and simulation . . . . .	28
21	Example E: Ordered probit, output fragments . . . . .	29
22	Example E: Ordered mixed probit and Monte Carlo experiment . . . . .	29
23	Variable notation . . . . .	38

## List of Figures

1	Example F: Ordered mixed probit and Monte Carlo generated distributions . . . . .	30
---	---	----

# 1 Introduction

## 1.1 Disclaimer

The DCM package is functional, but we provide no warranty. For general issues relating to Ox and the DCM package we refer users to the ox-users discussion group. Subscription information and archiving is available at

<http://www.mailbase.ac.uk/lists/ox-users>

We are happy to receive suggestions for improvement, although the program for future updates and revisions will be determined by the authors. For Matias Eklof: [matias.eklof@nek.uu.se](mailto:matias.eklof@nek.uu.se); for Melvyn Weeks: [Melvyn.Weeks@econ.cam.ac.uk](mailto:Melvyn.Weeks@econ.cam.ac.uk).

## 1.2 Availability and citation

DCM is available free of charge for academic users from

<http://www.econ.cam.ac.uk/faculty/weeks/DCM/DCMWebPage.htm>.

The only condition of use is that authors cite this document in all reports and publications involving the application of the DCM package.

## 1.3 Installation

- (1) You will need to install Ox version 3.2 or higher. DCM will not work with earlier versions of Ox.
- (2) Create a DCM sub-directory in the `ox\packages` folder.
- (3) Download and unzip `DCM100.zip` to this directory.
- (4) Read the `Readme.txt` file for last minute changes.
- (5) If Ox has been properly installed DCM may be used from any directory. To use the package in your code, include the command  

```
#include "packages\dcm\dcm.ox"
```

at the top of all files that require it.

DCM runs under either the Ox Console version or GiveWin with OxProfessional. These modes of operation requires that the user write short Ox programs. As will become evident in the examples provided below, this may be done with very little experience in using the Ox programming language. OxEdit, an easy to use text editor which supports syntax highlighting and running external tools, may be used to both develop and run Ox programs. This editor can be obtained

from <http://www.oxedit.com/oxedit.html>. DCM v1.0 is distributed with an OxPack graphical user interface, which allows the user to run DCM interactively using OxPack in GiveWine.

We emphasise that the manual is written using a *how-to* style. In this respect we assume that the user is familiar with the econometrics issues relating to the identification, specification, and estimation of discrete choice models. For those users who require background material on the various models referred to in this manual see Eklof and Weeks (2004).

Section 2 provides an overview of DCM. In Sections 3-5 we demonstrate both the functionality and the use of DCM, focussing initially on how to write short Ox programs: section 3 illustrates how to load, create and transform data; section 4 considers the options available for model specification, including how to allow for random coefficients and correlated error terms; and in section 5 we present the different estimation routines available within DCM, including a custom written optimisation routine. Section 6 focusses upon post-estimation, and in particular a test for random preference heterogeneity, and how to translate DCM output into Latex formatted tables.

In section 7 and 8 we demonstrate the use of DCM in estimating a number of discrete choice models. Section 7 presents a number of examples based upon observed data, whereas section 8 demonstrates the use of DCM in both generating and estimating discrete choice models. In section 9 we demonstrate how DCM may be used as a package in conjunction with the GiveWin interface. Using a graphical interface users can specify and estimate models without writing individual programs. Section 10 concludes.

## 2 Overview

Estimating a discrete choice model using DCM can be partitioned into 5 distinct steps: (1) creating a DCM object, (2) loading the database, (3) selecting the variables into the appropriate groups, (4) model settings, and (5) estimating the model. An example of a short Ox program using DCM is given in Table 1. In the example we estimate a mixed logit model with two attributes with joint normally distributed coefficients and alternative specific constants. In the sections which follow we examine the functionality surrounding the structure of this example.

Since DCM is a class written in Ox, the user must first create an object. The DCM object is created using the command

```
decl dcm = new DCM();
```

where the object created is named `dcm`. This command will call the constructor member function of the DCM class and initiate the DCM object called `dcm`. This is the object into which we will load the data, insert the model formulation and

Table 1: First example of DCM code

```

1 #include "packages\\dcm\\dcm.ox"
2 main()
3 {
4     decl dcm = new DCM();           // (1) Creating DCM object
5     dcm.Load("Greene.xls");        // (2) Loading database
6     dcm.Select(Y_VAR,{"Mode",0,0}); // (3) Selecting variables
7     dcm.Select(A_VAR,{"GC",0,0,"Ttme",0,0});
8     dcm.Select(I_VAR,{"Constant",0,0});
9     dcm.SetModel(MXL);             // (4) Model settings
10    dcm.SetCoeffDist(NORMAL,{"GC","Ttme"},1);
11    dcm.Estimate();                // (5) Estimation
12 }

```

specification, and from which we will extract the estimation results. Multiple DCM objects can be handled simultaneously without interference between data sets and model specifications. Note that Ox is case sensitive i.e. `dcm` is not equivalent to DCM.

### 3 Data Input and Transformation

In the following *data format* refers to the format in which the database is initially stored, e.g. GiveWin, Excel, Stata, GAUSS, etc. Given that the DCM class is derived from the `Modelbase` and `Database` classes, DCM can read all formats available to the `Database` class. These data formats include Excel, Lotus, formatted and unformatted ASCII files, Stata, and GiveWin formats. We refer to the Ox manual for a description of available data formats. The *data structure* refers to the internal organization of the data within the database. Most discrete choice software requires data to conform to either one of two specific data structures. A distinguishing characteristic of the DCM package (not seen in other discrete choice packages to our knowledge), is that it places no requirements on how the data is structured, allowing the user considerable flexibility in how they choose to store data.

In what follows a *record* indicates the total information associated with a decision maker's choice. This will possibly include the characteristics of the decision maker, attributes of alternatives, and the chosen alternative. A record will consist of either a single or multiple rows in the database. If a record is contained within a single row, we refer to this as a *single row structure*, otherwise as a *multiple rows structure*. The data structure is often dependent on the context of the data: stated preference data usually conforms with the multiple row structure, whereas revealed preference data comes in the shape of single row structures. DCM handles both types of structures seamlessly. In outlining the various formats we first denote  $N$ ,  $T$ , and  $J$  as, respectively, the number of in-

dividuals, time periods, and alternatives.  $K$  is used to denote the number of characteristics and  $L$  denotes the number of attributes. For a given individual the dependent variable is represented either as the scalar *index* of the preferred alternative (in the single row structure), or as a row vector of *dummy* variables indicating with a single non-zero value the preferred alternative. DCM handles both cases automatically.<sup>1</sup>

The database will include  $NT$  rows in the *single row* structure, and  $NTJ$  rows in the *multiple row* structure. Tables 2 and 3 presents two examples taken from different versions of the Greene transportation mode data set.

Table 2: Example of "multiple rows" structured database

Hinc	PSize	Ttme	Invc	Invt	GC	Mode	
35	1	69	59	100	70	0	<i>individual 1</i>
35	1	34	31	372	71	0	"
35	1	35	25	417	70	0	"
35	1	0	10	180	30	1	"
30	2	64	58	68	68	0	<i>individual 2</i>
30	2	44	31	354	84	0	"
30	2	53	25	399	85	0	"
30	2	0	11	255	50	1	"

Table 3: Example of "single row" structured database

Hinc	PSize	Ttme	Ttme	Ttme	Ttme	...	Mode	
35	1	69	34	35	0	...	4	<i>individual 1</i>
30	2	64	44	53	0	...	4	<i>individual 2</i>

Below follows a formal description of the data structures.

**Multiple row structure:** For each individual and for each time period, a record is comprised of  $J$  rows. Each row spans  $K$  columns holding data on individual characteristics,  $L$  columns on attributes, and a single column indicating with a single non-zero value (usually a "1") the preferred alternative. This structure is common in stated preference analysis.

**Single row structure:** For each individual and for each time period, a record is comprised of a single row. This row contains  $Jk$  columns for individual characteristic  $k$  ( $\sum_{k=1}^K Jk$  in total), and  $JL$  columns for alternative

<sup>1</sup>DCM infers the structure of the data from the selected dependent variable and the number of time periods  $T$ .

See the technical appendix for details on how DCM determines values for  $N$ ,  $T$ , and  $J$  from the database.

attributes. If the chosen alternative is represented as a scalar, it is assumed to indicate the index of the preferred alternative<sup>2</sup>. Otherwise the chosen alternative is represented as a row vector with a non-zero value in the position of the chosen alternative. This structure is common in revealed preference analysis. In this structure, column names must be repeated across columns holding the same variable.

### 3.1 Panel data

The required organisation of panel datasets in DCM follows logically from that of a single cross-section. For both data structures DCM assumes that the data is stacked by individual. In the case of single (multiple) row data structures each individual will contribute  $T$  ( $TJ$ ) rows in the database. For *balanced* panel data, the user must supply the number of time periods per individual using the command

```
dcm.SetPanel(cT);
```

where  $cT$  is an integer equal to the number of time periods. The default value for  $cT$  is 1.

DCM can also handle unbalanced panels, and in this instance the user must supply a variable identifying individuals. This is accomplished with the following command

```
dcm.SetID(sVar);
```

where  $sVar$  is a string holding the name of the ID variable.<sup>3</sup>

### 3.2 Loading data

The database is loaded into the object `dcm` using the command

```
dcm.Load(filename);
```

The extension of *filename* indicates the original data format.

### 3.3 Transforming between data structures

The two data structures described above, single and multiple row, have different merits. Although the single row structure preserves memory, the multiple rows

---

<sup>2</sup>The indexing may start at either 0 or 1.

<sup>3</sup>Strings are enclosed by double or single quotes.

structure is more convenient for variable transformations given that DCM is a class which is derived from the Database class in Ox. Since this is likely to inconvenience users wishing to transform variables prior to estimation, DCM has the ability to transform the loaded database between these two structures. The command

```
TransformData(sY, sNewDbName);
```

toggles between single and multiple row structures, where *sY* is the name of the dependent variable in the database and *sNewDbName* is the name of the transformed database. Generally, the user would first call `dcm.Load(sFilename)` to load the database and then call `dcm.TransformData(sY, sNewDbName)` to transform the data. Note that the command will replace the database in memory by the transformed database. This command will, in general, be used in conjunction with appending and transforming new database variables.

### 3.4 Data manipulation and variable selection

In DCM we separate the process of *creating* and *transforming* variables from the process of selecting variables into the model. This section describes the relevant DCM commands.

#### 3.4.1 Creating Variables

After the data has been loaded the user may create and add new variables to the database using the Database member functions `Append(mX, asX)` and `GetVar(asVar)` described in the Ox manual. Note that these commands may not work as expected in the single row structure where the variables names are repeated across columns, and therefore not unique. This is an artifact of the Database class and cannot be resolved easily within DCM.<sup>4</sup> Given this limitation we recommend that the user transform the data into the multiple row structure using the `TransformData(sY, sNewDbName)` command *before* calls to `Append(...)` or `GetVar(...)` etc.

**Scaling variables** To scale variables by a constant DCM provides a command that works independent of the database structure. The command

```
dcm.ScaleVar(aScale);
```

scales the variables listed in *aScale*, e.g.

---

<sup>4</sup>The user could explore the non-exported DCM member function `GetEveryVar(asVar)` which returns *all* variables in the database labelled as *asVar*.

```
dcm.ScaleVar({'Var1',0.01,'Var2',10});
```

will multiply *Var1* by 0.01 and *Var2* by 10 before estimation. The scaling occurs internally and the variables in the database are unchanged.

### 3.4.2 Selecting Variables

In the canonical form of a discrete choice model (see Eklof and Weeks (2004)), there are three distinct groups of variables: (1) the dependent variable, (2) individual characteristics, and (3) alternative attributes. The command

```
dcm.Select(iGroup, asVar);
```

selects the variables listed in *asVar* into group *iGroup*, where *asVar* follows the syntax described in the Modelbase class<sup>5</sup>. The options for *iGroup* are presented in Table 4.

Table 4: `Select(iGroup, asVar)` options

<i>iGroup</i>	Description
Y_VAR	Dependent variable. Either an index or a dummy variable.
I_VAR	Individual characteristics.
A_VAR	Alternative specific attributes.
S_VAR	Choice Set Indicator.

Consider the case where the loaded database includes two attributes named *Var1* and *Var2*. The command

```
dcm.Select(A_VAR, {"Var1",0,0,"Var2",0,0});
```

selects database variables *Var1* and *Var2* into the group of alternative attributes.<sup>6</sup>

If the choice set varies across individuals, the command

```
dcm.Select(S_VAR, asChoiceSetVar);
```

selects from the database the zero-one indicator variable *asChoiceSetVar* which indicates for each individual whether a specific alternative is in the choice set.

<sup>5</sup>Note that the group X\_VAR is not used in DCM.

<sup>6</sup>The 0's relate to the potential use of lags in the model. However, as DCM v1.0 does not support dynamic models these arguments must be set to 0.

This is now implemented for conditional, nested, and mixed logit models, but not for multinomial, ordered, or ordered mixed probit models.

Some special variable constructions are frequently used in discrete choice modelling, such as alternative specific constants and interactions terms between independent variables. Although these could be generated using e.g. `Append()` commands discussed above, DCM provides the user with convenient shortcuts for creating and including such variables. These are discussed below.

**Deterministic variables** There are two ways of including alternative specific constants into the model. If not already in the database, the user can create a set of alternative specific constants using

```
dcm.Deterministic(TRUE);
```

The alternative specific constants are named `ALT_j`,  $j=0, \dots, J-1$  by default, and are automatically included in the model to be estimated.<sup>7</sup> These constants may be renamed using the command

```
dcm.SetAltNames(asAltNames);
```

where *asAltNames* is an array of strings of alternative names.<sup>8</sup>

A second way to include alternative specific constants is to select the automatically created constant as an individual characteristic.<sup>9</sup> The command

```
dcm.Select(I_VAR, {"Constant", 0, 0});
```

will include alternative specific constants in the estimated model. Since this variable is treated as an individual characteristic, the labels in the output table will not be affected by the `dcm.SetAltNames()` command.

**Interaction terms** The command

```
dcm.Interact(asVar1, asVar2);
```

creates new variables based upon interactions between all combinations in *asVar1* and *asVar2* where *asVar1* and *asVar2* are arrays of strings of variable names. The interaction terms will automatically be selected into the model and will be

---

<sup>7</sup>However, since alternative specific constants are created internally, they may not be referred to within a `Select()` statement.

<sup>8</sup>An array is enclosed by curly brackets `{}`.

<sup>9</sup>The variable `Constant` is automatically appended to the database when loading.

named e.g. "Var1\*Var2". Multiple calls to `Interact(...)` can be used to create "blocks" of interaction terms. Note that `dcm.Interact()` does not check for linear dependence across created terms.

The command

```
dcm.RemoveInteract();
```

removes *all* interaction terms from the model.

## 4 Model Specification

Once the data has been loaded and variables selected, the user must make a number of choices over type of model and specification of the stochastic components. An overview of the available settings for the various models is given below and summarised in Table 6.

The command

```
dcm.SetModel(iModel);
```

sets the estimating model. The options are given in Table 5, together with the number of model parameters.

Table 5: `SetModel(iModel)` options

<i>iModel</i>	Description	No. of model parameters <sup>a</sup>
CL	Conditional/multinomial logit	$(J - 1)K + L = C$
NL	Nested logit	$C + \#nests$
MXL	Mixed logit	$C + C(C + 1)/2$
MXP	Mixed probit	$C + C(C + 1)/2$
MNP	Multinomial Probit	$C + C(C + 1)/2 + J(J - 1)/2 - 1$
OP	Ordered probit	$K + J - 1$
OMP	Ordered mixed probit	$K + K(K + 1)/2 + J - 1$

<sup>a</sup> $K$  potentially includes alternative specific constants.

For a description of the associated likelihood functions, we refer the reader to an accompanying paper Eklof and Weeks (2004).

`SetRefAlt(iRefAlt)` Sets the reference alternative, i.e., the alternative against which the utility of other alternatives in the choice set is compared. This is only relevant for unordered models which contain individual characteristics. The variance of the error term for the *iRefAlt* is set to 1 in the MNP model. Note that the *iRefAlt* cannot be the same as *iScaleAlt* (see below).

Table 6: Overview of model specifications

<i>iModel</i> =	CL	NL	MXL	MXP	MNP	OP	OMP
SetRefAlt( <i>iRefAlt</i> ) (D: <i>iRefAlt</i> =0)	✓	✓	✓	✓	✓	n.a.	n.a.
SetScaleAlt( <i>iScaleAlt</i> ) (D: <i>iScaleAlt</i> =1)	n.a.	n.a.	n.a.	✓	✓	n.a.	n.a.
SetNest( <i>avNest</i> )	n.a.	✓	n.a.	n.a.	n.a.	n.a.	n.a.
SetCoeffDist( <i>iDist</i> , <i>asVar</i> , <i>iGroup</i> )							
<i>iDist</i> = FIXED	(A)	(A)	✓(D)	✓(D)	✓(D)	(A)	(A)
NORMAL	n.a.	n.a.	✓	✓	✓	n.a.	✓
(-)LOGNORMAL	n.a.	n.a.	✓	n.a.	n.a.	n.a.	n.a.
SetErrDist( <i>iErr</i> )							
<i>iErr</i> = IID	n.a.	n.a.	(A)	(A)	✓(D)	(A)	(A)
HETEROSC	n.a.	n.a.	n.a.	n.a.	✓	n.a.	n.a.
UNREST	n.a.	n.a.	n.a.	n.a.	✓	n.a.	n.a.
SetRandom( <i>iRandom</i> , <i>cR</i> )							
<i>iRandom</i> = UNIFORM	n.a.	n.a.	✓	✓	✓	n.a.	✓
HALTON	n.a.	n.a.	✓(D)	✓(D)	✓(D)	n.a.	✓(D)
NONE	n.a.	n.a.	n.a.	✓	✓	n.a.	n.a.
(D: <i>cR</i> =50)							

Note: ✓=Option available, n.a.=Option not available, A=Automatic, D=Default

SetScaleAlt(*iScaleAlt*) Sets the scale alternative of the model i.e. the alternative for which the diagonal element in the cholesky decomposition of error covariance matrix is set to unity. This is only relevant for models with normally distributed error terms. This cannot be the same alternative as *iRefAlt* (see above).

SetNest(*avNest*) Sets the nesting structure of nested discrete choice models. The argument *avNest* is an array of vectors holding the index of the alternatives in each nest. DCM v1.0 only supports one level of nesting, i.e., nests within nests are not supported. Recall that Ox indexing start at 0, hence first alternative is indexed by 0.

*Example:* SetNest( $\{< 0 >, < 1, 2, 3 >\}$ ) will place alternative 0 in one nest and alternatives 1, 2, and 3 in a second nest.

SetCoeffDist(*iDist*, *asVar*, *iGroup*) For random coefficient models, this com-

Table 7: `SetCoeffDist(iDist, asVar, iGroup)` options

<i>iDist</i>	Description
FIXED	Non-random coefficient
NORMAL	Normal distributed random coefficient
(-)LOGNORMAL	Log-normal distributed random coefficient with positive (negative) support

mand sets the distribution of coefficients and the covariance structure. The first argument sets the distribution of the parameters listed in the second argument. The (optional) *iGroup* argument allows for correlation across random coefficients within groups (default *iGroup*=0). Coefficients in group *iGroup*=0 are assumed uncorrelated, whereas as coefficients sharing the same *iGroup*> 0 number are allowed to correlate. This can be used to allow for correlation across different types of distributions. *iGroup* can take any integer value.

*Example:*

```
dcm.SetCoeffDist(NORMAL, {"Var1", "Var2"}, 1);
dcm.SetCoeffDist(LOGNORMAL, {"Var3", "Var4"}, 1);
```

sets coefficients associated with the variables *Var1* and *Var2* as distributed joint normal and *Var3* and *Var4* to log-normal. Correlation within and across normal and log-normal coefficients is allowed as they are all in the same *iGroup*=1.

If the variable names listed in *asVar* do not exist among the model parameters DCM terminates with an error.

The log-normal distribution has a *positive support* by default. To allow for negative support, e.g. to model a negative price effect, insert a minus sign before the LOGNORMAL specification.

*Example:* `SetCoeffDist(-LOGNORMAL, {"Var1", "Var2"}, 1);`

For *asVar*=-1 all coefficients in the model are assigned the distribution indicated by *iDist* and *iGroup*.

**SetErrDist(*iErr*)** For the multinomial probit model, this command sets the structure of the error covariance matrix. All available options in DCM will result in an identified covariance matrix. Options are given in Table 8. See Appendix A.2 for a detailed description of the error covariance structure.

**SetRandom(*iRandom*, *cR*)** Sets the type and number of random draws for models that requires Monte Carlo integration techniques (MXL, MXP, MNP, and OMP). For low dimensional multiple index models with normally distributed errors, the user can choose the option NONE and use the Ox numerical integration routines to estimate multivariate normal probabilities.

Table 8: `SetErrDist(iErr)` options

<i>iErr</i>	Description
IID	Independently, identically distributed random errors
HETEROSC	Independent, heteroscedastic distributed random errors
UNREST	Unrestricted joint distributed random errors (identified structure)

Table 9: `SetRandom(iRandom, cR)` options

<i>iRandom</i>	Description
UNIFORM	Uniform pseudo random numbers generated using Ox intrinsic commands
HALTON	Equidistant sequence of numbers
NONE	Use standard numerical integration routines

#### 4.1 Some comments on model specification

One of the principle distinctions between the mixed logit and multinomial probit model can be appreciated from Table 6. For example, consider the case where an analyst is contemplating estimating either a mixed logit or multinomial probit model, and is interested in capturing estimates of random preference heterogeneity. In the case of the MXL model the user will use `SetCoeffDist()` to make choices as to whether individual mean coefficients are to be considered as fixed or random; and, if random, both the form of the mixing distribution and whether random components are correlated. However, the use of the MNP model requires both this setting in conjunction with a specification on the *residual* error component using `SetErrDist()`. This follows from the fact that whereas the MXL model partitions the stochastic component into two additive parts - one heteroscedastic and correlated over the choice set, and another which is i.i.d. type 1 extreme value - the MNP model does not make such a distinction. Two other observations are worth making. First, one disadvantage of the MNP model is that the form of the mixing distribution is fixed and normal. In the current version of DCM, a mixed logit model comes with two mixing distributions: normal and log-normal. Second, if a MNP model is chosen *and* a user allows for free covariance parameters, identification restrictions are required. In contrast the mixed logit model has at its core a kernel logit model, and therefore, identification is automatic.

Table 10: `SetAlgorithm(iAlgorithm)` options

<i>iAlgorithm</i>	Description
BHHH	Uses inverse of outer gradient product as estimate of the Hessian in a Newton type algorithm. Shipped with DCM
BFGS	Intrinsic optimizer in Ox
NEWTON	Intrinsic optimizer in Ox

Table 11: `SetStdErr(iStdErr)` options

<i>iStdErr</i>	Description
ROBUST	Use robust estimate of parameter covariance matrix
HESSIAN	Use negative inverse of final Hessian of log-likelihood function
OGP	Use inverse of final outer gradient product of log-likelihood function

## 5 Estimation

There are number of distinct commands which refer to estimation procedures. These are described below.

`SetStartPar(vPar)` By default DCM uses conditional logit or ordered probit models to create parameter starting values for the more complicated models. This is overridden by this command.

*Example:* Consider the MXL model with 4 alternative specific constants, 1 characteristic, and 1 attributes with a random coefficient. The command

```
SetStartPar(<1;2;3;      // Alt. spec. constants
            1;2;3;      // characteristics
            1;          // attributes
            0.5>);     // std. err. of random coeff
```

would then set the starting values.

`SetAlgorithm(iAlgorithm)` Toggless between available optimization routines. Options are given in Table 10

`SetStdErr(iType)` Sets the type of estimated coefficient covariance matrix. The options are described in Table 11.

`Estimate()` Estimates the specified model. Parameter estimates and covariances can be extracted using e.g. `dcm.GetPar()` and `dcm.GetCovar()` commands.

## 6 Post-estimation

After successful estimation DCM provides a number of commands relating to output and testing. These are discussed below.

### 6.1 Output

`OutputLaTeX(...)`, `OutputLaTeX(aM0,...)` Calling e.g. `dcm.OutputLaTeX(aM0,aM1)` will produce a L<sup>A</sup>T<sub>E</sub>X formatted table with two columns of results for models M0 and M1. Parameters that are set to zero in a given model will appear as blanks. The arguments are optional. If no arguments are passed to `OutputLaTeX()`, DCM will use the most recent set of estimation results.

If arguments are submitted, they should appear as 5 element arrays consisting of: an  $q$  element array of names of estimated coefficients, a  $q \times 1$  vector of estimates, a  $q \times 1$  vector of estimated standard errors, a scalar log-likelihood value, and the number of observations. These arrays can be extracted using the member function `dcm.GetAllResults()`. For example, for model M0

```
decl aM0=dcm.GetAllResults()
```

### 6.2 Testing

`TestRandCoeff(asVar)` Tests for unobserved heterogeneity in coefficients. The test is operationalized by adding a number of artificial variables constructed from a previous CL estimation, and then re-estimating the model. The test is based upon a set of exclusion restrictions, with a chi-square distribution. *asVar* is an array of variable names indicating coefficients will be tested for heterogeneity. Setting *asVar*=-1 test all coefficients.

## 7 Examples

In what follows we demonstrate the use of DCM with reference to a number of examples.

### 7.1 Conditional, Nested, and Mixed logit models

Following earlier work by Daganzo (1979) and McFadden (1977), the transport mode choice problem has continued to figure prominently in both reflecting and promoting developments in discrete choice methodology. In recognition of this fact, we examine a well known modal choice dataset recording the inter-city travel choices between Melbourne, Canberra and Sydney. There are a total of

Table 12: Example A: Conditional logit with post-estimation testing

```

1 #include <oxstd.h>
2 #include "packages\dcm\dcm.ox" // Include DCM package
3
4 main()
5 {
6     decl dcm = new DCM(); // Create a DCM object
7     dcm.Load("Greene.xls"); // Load database
8     dcm.Deterministic(TRUE); // Append alt. spec. constants
9     dcm.SetAltNames({"air","train","bus","car"}); // Set new alt. names
10    dcm.Select(Y_VAR,{"Mode",0,0}); // Select dependent variable
11    dcm.Select(A_VAR,{"GC",0,0,"Ttme",0,0}); // Select independent variables
12    dcm.Interact({"air"},{"Hinc"}); // Append and select interactions
13    dcm.SetRefAlt("car"); // Set reference alternative
14    dcm.SetAlgorithm(BFGS); // Use BFGS optimizer
15    dcm.Estimate(); // Estimate the model
16    dcm.TestRandCoeff(-1); // Test all coefficients
17    dcm.TestRandCoeff({"air","train","bus"}); // Test a subset of coefficients
18 }

```

210 observations of non-business travellers faced with the choice between plane, train, bus, and car.<sup>10</sup> This dataset has been used extensively with examples including Greene (2002), Louviere, Hensher, and Swait (2000), and Ben-Akiva, Bolduc, and Walker (2001). The covariates included are terminal waiting time (*Ttme*), in-vehicle cost (*Inv*), in-vehicle time (*Invt*), generalized costs (*GC*) calculated from *Invt*, *Inv*, and a measure of wage rates, and household income (*Hinc*) interacted with the "air" alternative. The estimated model is

$$U_j = \alpha_j + \beta_1 GC_j + \beta_2 Ttme_j + \beta_3 (Hinc * air_j) + \epsilon_j,$$

where  $\alpha_j$  is an alternative specific constant and  $air_j$  is a alternative specific dummy.

In table 12 we present the code which allows the user to estimate a conditional logit model, followed by a test for random preference heterogeneity.

Fragments from the output produced by the code is presented in Table ?? . An ellipsis (...) indicates that some output has been omitted.

In examining the output we first note that DCM summarises the principal features of the data by indicating the number of individuals  $N$ , time periods  $T$  (if  $T > 1$ ), and alternatives  $J$ . For unbalanced panel DCM will report the minimum and maximum number of time periods and the frequency of the observed periods. DCM also reports the chosen reference and scale alternatives where appropriate. Following this, DCM reports the estimated coefficients and their standard errors,

<sup>10</sup>Note that the dataset is actually choice-based, with undersampling of the more popular mode, car. In order to obtain consistent estimates a weighted exogenous sample maximum likelihood estimator (WESML) should be used. However, we do not do this, since our primary objective is to compare our estimates with those in Greene (2002) and Louviere, Hensher, and Swait (2000).

Table 13: Example A: Conditional logit, output fragments

```

1  ---- DCM: Conditional Logit ----
2  The estimation sample is:  1 (1) - 210 (4)
3  The dependent variable is: Mode (Greene.xls)
4  Data structure : NTJ x K+L+1 with
5      N      = 210
6      J      = 4
7  Reference alternative:      car
8
9      Coefficient Std.Error t-value t-prob
10 -- ASC --
11 air           5.20744   0.9789   5.32  0.000
12 train         3.86904   0.5175   7.48  0.000
13 bus           3.16319   0.5463   5.79  0.000
14 -- Attributes --
15 GC            -0.0155015  0.004945  -3.13  0.002
16 Tme           -0.0961248  0.01506   -6.38  0.000
17 -- Interactions --
18 air*Hinc      0.0130870  0.009264   1.41  0.159
19
20 NOTE: Robust standard errors.
21
22 log-likelihood  -199.128369
23 no. of observations    210  no. of parameters      6
24 AIC.T             410.256737  AIC             1.95360351
25 Time to convergence:  0.04      (hh:mm:ss.hs, excluding time for covariance.)
26 BFGS using analytical derivatives (eps1=0.0001; eps2=0.005):
27 Strong convergence
28 Used starting values:
29      0.00000    0.00000    0.00000    0.00000    0.00000    0.00000
30 ...
31 Test for excluding:
32 ...
33 Subset Chi^2(6) = 74.9196 [0.0000] **
34 ...
35 Test for excluding:
36 ...
37 Subset Chi^2(3) = 55.2356 [0.0000] **

```

Table 14: Example B: Nested logit with variable transformations

```

1 #include <oxstd.h>
2 #include "packages\dcm\dcm.ox"
3
4 main()
5 {
6     decl dcm = new DCM();           // Create DCM object
7     dcm.Load("Greene.xls");        // Load database
8                                     // Append alt. spec. constants
9     dcm.Append(ones(210,1)**unit(4),{"air","train","bus","car"});
10                                    // Append interaction term
11     dcm.Append(dcm.GetVar("air").*dcm.GetVar("Hinc"),"air*Hinc");
12     dcm.Select(Y_VAR,{"Mode",0,0}); // Select dependent variable
13                                     // Select independent variables
14     dcm.Select(A_VAR,{"air",0,0,"train",0,0,"bus",0,0,
15                       "GC",0,0,"Ttme",0,0,"air*Hinc",0,0});
16     dcm.SetRefAlt(3);                // Set reference alternative
17     dcm.SetModel(NL);                // Set model
18     dcm.SetNest({<0>,<1,2,3>});      // Set nesting structure
19     dcm.SetAlgorithm(BFGS);         // Set algorithm
20     dcm.Estimate();                 // Estimate model
21 }

```

subdivided into the variable categories (alternative specific constants, attributes, characteristics, etc.). Below this section standard deviations and correlations of random coefficients and error terms are reported, again where appropriate. Finally, DCM reports the value of the maximised log-likelihood and the starting values.

The `TestRandCoeff(asVar)` commands produces a similar sets of outputs given that this test involves re-estimating the model with artificial variables (which are denoted by the prefix "ART\_"). In Table 13 we present the specific set of outputs from these tests, which are based upon exclusion restrictions for the artificial variables. Test statistics and corresponding p-values are also provided.

In Table 14 the code for the Nested logit model is presented. We assume that the modes can be partitioned into "fly" and "ground" nests, such that "air" (alternative 1 is indexed by 0) is in one nest and "train", "bus", and "car" (alternatives 2, 3, and 4 are indexed as 1,2, and 3, respectively) is in a second. Note that even though the first nest contains a single alternative, the argument must be a vector rather than a scalar. Fragments of the output are presented in Table 15. To illustrate the use of the Database commands `Append()` and `GetVar()`, we construct alternative specific constants and interaction terms manually. See the Ox manual for syntax.

The additional information printed for the Nested logit model is the nesting structure and the set of variables that are constant or varying within nests. This information is given in the first panel. The estimates' panel now includes the estimates of the coefficients of the "inclusive values".

The last model estimated in this subsection is the Mixed logit model. In

Table 15: Example B: Nested logit, output fragments

```

1  ...
2  ---- DCM: Nested Logit ----
3  ...
4  Reference alternative:      ALT_3
5  Alternative(s) in nest 0:  ALT_0
6  Alternative(s) in nest 1:  ALT_1  ALT_2  ALT_3
7  Level 1 var's (constant within nests):  air air*Hinc
8  Level 2 var's (varying within nests) :  train  bus GC  Tme
9
10         Coefficient  Std.Error  t-value  t-prob
11 -- Attributes --
12 air           3.54087    1.791    1.98    0.049
13 train        5.06460    0.7153   7.08    0.000
14 bus          4.09631    0.6910   5.93    0.000
15 GC           -0.0315875   0.01034  -3.05   0.003
16 Tme          -0.112617    0.01961  -5.74   0.000
17 air*Hinc     0.0153313    0.008380  1.83    0.069
18 -- Inclusive values --
19 Incl.Val.[0] 0.586010    0.2045   2.87    0.005
20 Incl.Val.[1] 0.388762    0.1121   3.47    0.001
21 ...

```

order to illustrate the use of the `TransformData()` command, we first create a temporary DCM object into which we load the `Greene.xls` data. This database is originally structured with multiple rows for each record. The `TransformData()` command transforms this structure to "single row" and stores the database in the Excel file "tmp.xls" for later inspection. This database is then used in the estimation that follows.

As indicated in the parameter heterogeneity test above, the alternative specific constants may capture some unobserved heterogeneity across alternatives. We allow the random alternative specific constants to correlate by selecting them into group 1. In addition to illustrate the use of `SetCoeffDist()`, we allow the generalized price coefficient to vary across individuals. As this coefficient is related to a price variable with positive support, it is reasonable to assume that the support over distribution of the coefficient is negative. Recall that the `LOGNORMAL` option in the `SetCoeffDist()` command has a positive support by default. However, inserting a minus sign (`-LOGNORMAL`) shifts the support of the distribution to the negative real line. Further, we assume that this coefficient is uncorrelated with any other coefficients and therefore selected into group 0 (by default).

The output produced by the DCM commands in Table 16 is presented in Table ???. The first few lines relate to the first section of the code where the transformed database is constructed. The remaining rows relates to the estimated model. The structure of the database and the reference alternative are presented in the first panel. The panel of coefficient estimates follows. The estimates of the cholesky decomposition of the coefficient covariance matrix are also presented, with the

Table 16: Example C: Mixed logit with "single row" data structure

```

1 #include <oxstd.h>
2 #include "packages\dcm\dcm.ox"
3
4 main()
5 {
6     // Transform "multiple row structure" to "single row structure"
7     decl dcm = new DCM();           // Create DCM object
8     dcm0.Load("Greene.xls");       // Load original database
9     dcm0.TransformData("Mode","tmp.xls"); // Transform and save database
10    delete dcm0;                   // Delete this object
11
12    decl dcm = new DCM();
13    dcm.Load("tmp.xls");           // Load temporary *transformed* database
14    dcm.Deterministic(TRUE);
15    dcm.SetAltNames({"air","train","bus","car"});
16    dcm.Select(Y_VAR,{"Mode",0,0});
17    dcm.Select(A_VAR,{"GC",0,0,"Ttme",0,0});
18    dcm.Interact({"air"},{"Hinc"});
19    dcm.SetRefAlt("car");
20    dcm.SetModel(MXL);             // Set model to MXL
21    dcm.SetCoeffDist(NORMAL,{"air","train","bus"},1); // Set distr. of coeff's
22    dcm.SetCoeffDist(-LOGNORMAL,{"GC"}); // Use negative support for price coeff's
23    dcm.SetRandom(HALTON,100);    // Set type and no. of random draws
24    dcm.SetAlgorithm(BFGS);
25    dcm.Estimate();
26 }

```

numbers in square brackets indicating the position in the cholesky matrix. Below the estimates, DCM reports the standard errors and the correlations of the random coefficients.

## 7.2 Multinomial probit models

To demonstrate the specification and estimation of multinomial probit models we use the trinomial discrete choice model of the labour force status of married women in the UK as considered by Duncan and Weeks (1998). The model is discrete in that they allow for three states: non-workers supplying zero hours of work; part-time workers whose weekly supply is between 0 and 30 hours; and full-time workers supplying more than 30 hours. The data consist of a random sample of married women drawn from the 1993 Family Expenditure Survey (FES).

Across all specifications we condition our labour supply model on wage rates,<sup>11</sup> and the following socio-demographic *characteristics*; age of the woman, dummies for children in the age groups 0-2, 3-4, 5-10, and above 11, number of children, level of formal education and marital status (whether married or cohabiting).

<sup>11</sup>Since wage rates are not observed in the FES for those not in employment, we base our simulations on wage rate estimates derived from an appropriately corrected reduced form equation. See Duncan and Weeks for further details.

Table 17: Example C: Mixed logit, output fragments

```

1  ...
2  DCM package version 1.00, object created on 19-05-2004
3  ** NOTE: Transforming multiple to single row structure
4  ...
5  ---- DCM: Mixed Logit ----
6  The estimation sample is: 1 (1) - 53 (2)
7  The dependent variable is: Mode (tmp.xls)
8  Data structure : Single row per record with
9      N      = 210
10     J      = 4
11 Pseudo random draws: HALTON (R=100)
12 Reference alternative: car
13
14             Coefficient Std.Error t-value t-prob
15 -- ASC --
16 (1) air      4.33782    2.810    1.54   0.124
17 (2) train    5.49412    3.032    1.81   0.071
18 (3) bus      4.44471    2.555    1.74   0.083
19 -- Attributes --
20 (4) GC      -3.36821    1.029   -3.27   0.001
21 (5) Ttme    -0.115666   0.03849  -3.00   0.003
22 -- Interactions --
23 (6) air*Hinc 0.0466768   0.07750  0.602   0.548
24 -- Cholesky elements of Cov[coeff] --
25 C(Par)[1,1]  4.06479    5.805    0.700   0.485
26 C(Par)[1,2]  0.724140   1.533    0.472   0.637
27 C(Par)[1,3]  0.438704   0.9484   0.463   0.644
28 C(Par)[2,2] -0.532236   4.143   -0.128   0.898
29 C(Par)[2,3] -0.345838   2.548   -0.136   0.892
30 C(Par)[3,3] -0.00261691 0.05693 -0.0460  0.963
31 C(Par)[4,4]  0.00554049 0.1620   0.0342  0.973
32
33 ** NOTE: Robust standard errors.
34 ** NOTE: Mean parameters for log-normal distr. coefficients are
35           calculated as E[ln(beta)] (E[ln(-beta)] for neg. log-norm.).
36
37 Random parameters:
38 Parameter      Std. dev.      Correlation matrix (*=fixed, not estimated)
39 (1) air         4.063         1.000
40 (2) train       0.898         0.807    1.000
41 (3) bus         0.558         0.786    0.999    1.000
42 (4) GC         0.000         0*       0*       0*       1.000
43 ...

```

We also include a single *attribute* variable, net incomes at various hours levels. To generate state-specific net incomes as condition variables for the structural discrete choice models, we simulate tax liabilities and benefit receipts and total net incomes at 0, 20 and 40 hours for each individual in our sample.<sup>12</sup> Finally, to allow for age dependent income effect, we interact net income and age. Our dependent variable is a three-state variable which distinguishes non-participants (category 0), part-time workers between 1 and 30 hours (category 1) and full-timers working in excess of 30 hours (category 2). The reference alternative is the non-participation category. In reading the economic significance of the parameter estimates, a negative coefficient represents a decrease in the likelihood of working either part-time or full-time relative to not working. Which comparison is appropriate is identified for each parameter estimate in the table.

We consider a number of alternative model specifications by focusing upon the stochastic component of choice. These are: the MNL model (Model 1), a MNP with heteroscedastic error terms (Model 2), and a MNP with one random coefficient and IID errors (Model 3).

The DCM commands for estimating these models are presented in Table 18.<sup>13</sup> We also illustrate a number of additional features of DCM and Modelbase, such as extracting estimates and producing LaTeX formatted tables after estimating multiple models. Some of these features require knowledge of the Ox programming. We refer the interested user to the Ox manual.

Noteb that in the example code we call the member function `GetAllResults()`. This command returns an array of results which we will use to tabulate parameter names, estimates, and standard errors, together with the final loglikelihood value and the number of observations. These are stored in an array which will later be used calling the `OutputLaTeX(...)` command. In Table 19 we present the actual table produced by the code. "\*" and "\*\*\*" denote significance at the 10 and 1 per cent level.

## 8 Monte Carlo experiments using `Generate()`

In addition to estimating discrete choice models based upon observed data, DCM allows the user the option to generate discrete choice data. This may be accom-

---

<sup>12</sup>The wage equation is identified from the inclusion of demand-side (quarterly unemployment) and regional characteristics (vacancies and redundancies by region) as well as socio-demographic characteristics (quadratics and interactions between age, partners's age and education; age and number of children). Estimates are available from the authors on request. A problem with this approach is that it becomes difficult to correct the standard errors in the structural model for the inclusion of the generated wage rate term, since the simulated net income terms also depend (non-linearly) on the wage rate used. In the structural models, therefore, the standard errors remain uncorrected.

<sup>13</sup>See Duncan and Weeks (1998) for a discussion of parameter estimates and the application of both nested and non-nested tests across models.

Table 18: Example D: Conditional Logit and Multinomial probit models

```

1 #include <oxstd.h>
2 #include "packages\dcm\dcm.ox" //
3 Include DCM package
4
5 main() {
6     decl dcm = new DCM();
7     dcm.Load("Married4.xls");
8     dcm.Select(Y_VAR,{"REFPRED",0,0}); // Dependent variable
9                                     // Individual characteristics
10    dcm.Select(I_VAR,{"DKID02",0,0,"DKID34",0,0,"DKID510",0,0,
11                    "DKID110",0,0,"TOT_KIDS",0,0,"AGE",0,0,
12                    "EDGT16",0,0,"COHAB",0,0,"LNWFIT",0,0});
13    dcm.Select(A_VAR,{"INC",0,0}); // Alternative attributes
14    dcm.ScaleVar({"INC",0.01}); // Multiply INC by 0.01 before estimation
15    dcm.SetAlgorithm(BFGS);
16
17    // Model 1: Conditional logit
18    dcm.SetModel(CL);
19    dcm.Estimate();
20    decl aM1 = dcm.GetAllResults(); // Store relevant est. results in array
21
22    // Model 2: No random coefficients, heteroscedastic error covariance struct.
23    dcm.SetModel(MNP);
24    dcm.SetRandom(HALTON,50);
25    dcm.SetErrDist(HETEROSC);
26    dcm.Estimate();
27    decl aM2 = dcm.GetAllResults();
28
29    // Model 3: Normal distr. inc. coefficient, IID error covariance structure
30    dcm.SetCoeffDist(NORMAL,{"INC"}); // Normal distr. inc. coefficient
31    dcm.SetErrDist(IID);
32    dcm.SetStartPar(dcm.GetPar()[:18]| // Mean coefficients
33                  1); // standard dev. of random coeff.
34    dcm.SetRandom(HALTON,50);
35    dcm.Estimate();
36    decl aM3 = dcm.GetAllResults();
37
38    // Print all results to one LaTeX table
39    dcm.OutputLaTeX(aM1,aM2,aM3); // Print results from model 1-3 in table
40 }

```

Table 19: Example D: Multinomial models. Automated LaTeX output

	Model 1	Model 2	Model 3
(1) DKID02 1/0	-4.168* (1.865)	-2.753 (1.931)	-2.449 (1.828)
(2) DKID02 2/0	-23.813** (3.716)	-19.122** (5.932)	-21.272** (3.391)
(3) DKID34 1/0	14.807** (2.254)	12.780** (2.150)	13.388** (2.428)
(4) DKID34 2/0	-27.924** (4.306)	-22.594** (7.205)	-24.841** (3.741)
(5) DKID510 1/0	34.780** (3.484)	29.465** (5.680)	30.642** (4.222)
(6) DKID510 2/0	0.317 (3.339)	1.065 (2.799)	-0.518 (3.040)
(7) DKID110 1/0	29.620** (3.230)	24.876** (4.402)	27.244** (4.272)
(8) DKID110 2/0	4.764 (3.036)	4.538* (2.283)	4.687 (3.096)
(9) TOT KIDS 1/0	-7.730** (0.887)	-6.470** (1.556)	-6.533** (0.879)
(10) TOT KIDS 2/0	-7.326** (0.918)	-6.139** (1.509)	-6.333** (0.984)
(11) AGE 1/0	3.088* (1.801)	2.832* (1.323)	3.299* (1.713)
(12) AGE 2/0	-20.998** (3.115)	-17.010** (5.290)	-18.874** (2.982)
(13) EDGT16 1/0	-0.059 (0.636)	0.128 (0.530)	-0.080 (0.527)
(14) EDGT16 2/0	-0.004 (0.639)	0.154 (0.561)	-0.072 (0.539)
(15) COHAB 1/0	-9.873** (1.931)	-8.254** (2.750)	-8.164** (1.787)
(16) COHAB 2/0	4.882** (1.543)	4.138** (1.405)	4.784** (1.593)
(17) LNWFIT 1/0	8.208** (1.757)	6.599** (2.363)	6.421** (1.534)
(18) LNWFIT 2/0	19.701** (2.472)	15.956** (4.402)	17.209** (2.399)
(19) INC	4.281** (0.455)	3.566** (0.798)	3.608** (0.469)
C(Err)[2,2]	–	1.308* (0.523)	–
C(Par)[19,19]	–	–	0.503* (0.242)
Loglik.	-129.2	-130.622	-127.488
Obs.	1520	1520	1520

plished using the following steps:

- (1) Create a DCM object.
- (2) Set the model to be generated using  
`dcm.SetModel(iModel)`
- (3) Set the number of individuals, time periods, and alternatives using  
`dcm.SetDim(cN, cT, cJ)`
- (4) Append independent variables using  
`Append(mX, asX)`  
*mX* is an  $NTJ \times Q$  matrix ( $NT \times Q$  for ordered models) with (potentially) dependent variables. *asX* is an  $Q$  element array with variable names. For OP and OMP models the "single row" structure is required; otherwise the "multiple row" structure.
- (5) Select dependent variables in the data generating model using  
`dcm.Select(iGroup, asVar)`  
`dcm.SetCoeffDist(iDist, asVar, iGroup)`  
and  
`dcm.SetErrDist(iDist)`  
as described above to select variables and set the coefficient distribution and error covariance structure where required.
- (6) Set the true parameter vector using  
`dcm.SetTruePar(vP)`  
The organization of the parameter vector *vP* is described in the technical appendix (ASC, characteristics, attributes, thresholds, inclusive values, vectorized coefficient cholesky elements, vectorized error cholesky elements).
- (7) Generate a dataset using  
`dcm.Generate()`  
DCM will automatically create and `Select()` a dependent variable named "y". This can be renamed using `Rename()` (see Database manual).
- (8) Estimate the model using `dcm.Estimate()`.
- (9) To perform Monte Carlo experiments, store results and repeat (7)-(8).

## 8.1 Example: Ordered probit models

In Table 20 we illustrate the data generation features of DCM in generating data for an ordered probit model. Note that since data generation and estimation are distinct operations, we create two separate DCM objects; the `gen` object is

Table 20: Example E: Ordered probit and simulation

```

1 #include <oxstd.h>
2 #include "packages\\dcm\\dcm.ox"
3
4 main()
5 {
6     decl gen = new DCM(); // (1) Create DCM object
7     gen.SetModel(OP); // (2) Set model to simulate
8     gen.SetDim(1000,1,4); // (3) Set dimension of database
9     gen.Append(rann(1000,2),{"x1","x2"}); // (4) Append variables
10    gen.Select(I_VAR,{"x1",0,0,"x2",0,0}); // (5) Select variables to true model
11    gen.SetTruePar(<1;1;-1;0;1>); // (6) Set true parameter vector
12    gen.Generate(); // (7) Generate dependent variable "y"
13    gen.SaveXls("sim.xls"); // Save database in Excel
14    delete gen; // Delete object
15
16    decl dcm = new DCM(); // Create new DCM object
17    dcm.Load("sim.xls"); // Load database
18    dcm.Select(Y_VAR,{"y",0,0}); // Select dependent variable
19    dcm.Select(I_VAR,{"x1",0,0,"x2",0,0}); // Select independent variables
20    dcm.SetModel(OP); // Set model
21    dcm.Estimate(); // (8) Estimate
22    dcm.OutputLaTeX(); // Print LaTeX table
23 }

```

used for generating a dataset, and the `dcm` object is used as previously described. Output is written in a LaTeX table format.

Table ?? presents fragments of the output. In the estimate section there is a new label indicating estimates of thresholds. We note that the ordered probit model is identified with respect to location by dropping the constant and estimating all  $J - 1$  thresholds.

In the last example we demonstrate how to utilise DCM in a small Monte Carlo experiment, generating and estimating an ordered mixed probit model. In this example, the data generation and estimation is conducted within the same object. Note that we do not need to `Select()` variables in the estimation step as this is automatically done in the generation step. We report the generated distributions of the estimated coefficients and thresholds.<sup>14</sup>

## 9 DCM in OxPack for GiveWin

In this section we describe the OxPack implementation of DCM. In order to use the OxPack implementation the user must have an OxProfessional license properly installed. Before DCM can be used within OxPack the package must be added to the list of available packages. To do this start the OxPack module in GiveWin and use the "Package-Add/Remove Package..." menu to locate and

<sup>14</sup>If using DCM via GiveWin, the figure should be produced by GiveWin, otherwise the user need to locate and open the "OMPMC.eps" file to inspect the graphical results.

Table 21: Example E: Ordered probit, output fragments

```

1 ---- DCM: Ordered Probit ----
2 The estimation sample is: 1 (1) - 250 (4)
3 The dependent variable is: y (sim.xls)
4 Data structure : NT x (J or 1)K+JL+(J or 1) with
5     N       = 1000
6     J       = 4
7
8           Coefficient Std.Error t-value t-prob
9 -- Characteristics --
10 x1           0.950045   0.04647   20.4   0.000
11 x2           0.998723   0.04559   21.9   0.000
12 -- Thresholds --
13 alpha1      -1.03566   0.05612  -18.5   0.000
14 alpha2       0.0483520  0.04828    1.00   0.317
15 alpha3       0.974644   0.05542   17.6   0.000
16 ...

```

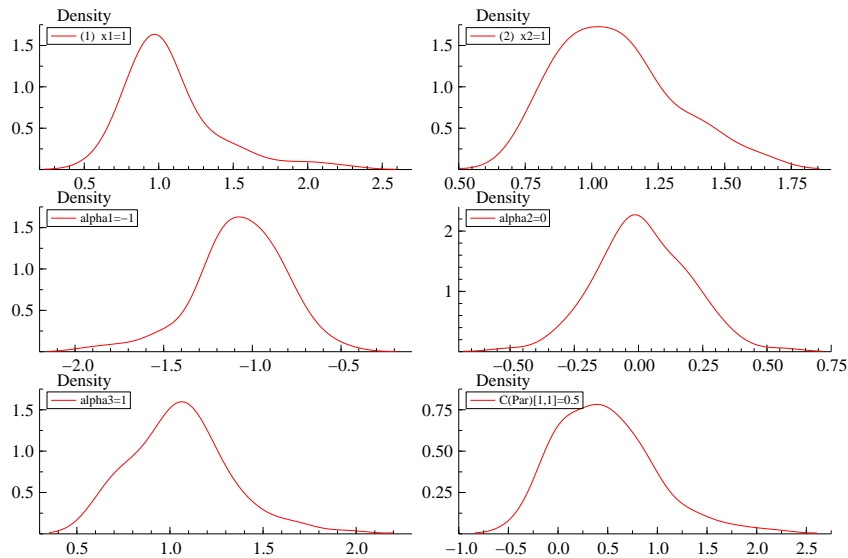
Table 22: Example E: Ordered mixed probit and Monte Carlo experiment

```

1 #include <oxstd.h>
2 #include "packages\dcm\dcm.ox"
3
4 main()
5 {
6     decl mc = new DCM();           // Create DCM object
7     mc.SetModel(OMP);             // Set model to simulate
8     mc.SetDim(1000,1,4);         // Set dimension of database
9     mc.Append(rann(1000,2),{"x1","x2"}); // Append variables
10    mc.Select(I_VAR,{"x1",0,0,"x2",0,0}); // Select variables to true model
11    mc.SetCoeffDist(NORMAL,{"x1"});
12    mc.SetTruePar(<1;1;-1;0;1;0.5>); // Set true parameter vector
13    decl mP = <>;
14    mc.SetPrint(FALSE);           // No print-outs
15    for (decl r=0;r<100;++r){     // Start MC replications
16        println("Replication ",r);
17        mc.Generate();             // Generate dependent variable "y"
18        mc.Estimate();             // Estimate model.
19        if (mc.GetResult()==MAX_CONV) // Store results if convergence
20            mP ~= mc.GetFreePar();
21    }
22    decl asLegend={};
23    for (decl p=0;p<mc.GetFreeParCount();++p)
24        asLegend ~= {sprintf(mc.GetFreeParNames()[p],"=",mc.GetTruePar()[p])};
25    DrawDensity(0,mP,asLegend,1,0,0);
26    ShowDrawWindow();
27    SaveDrawWindow("OMPMPMC.eps");
28    delete mc;
29 }

```

Figure 1: Example F: Ordered mixed probit and Monte Carlo generated distributions



select the `DCM.ox` file. Users may now choose DCM from the "Packages" menu in OxPack. Assuming that a proper database has been loaded, the user may now estimate any of the available models in DCM.

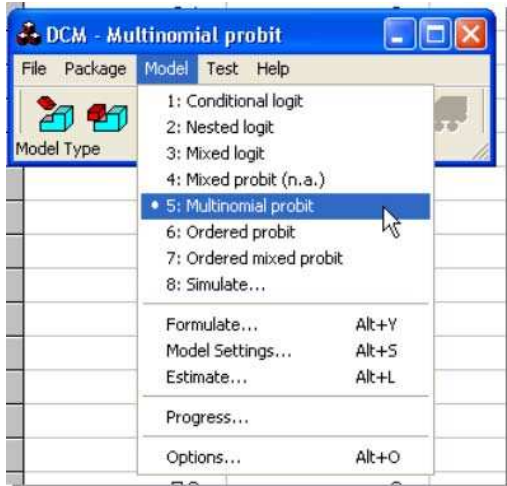
## 9.1 Limitations in OxPack implementation

1. The OxPack implementation of DCM can only be used with "multiple row" data structures
2. Variable constructions, such as `Interact()`, must be performed using the *calculator* or algebra code in GiveWin.
3. Output cannot be translated to LaTeX code.
4. Using the simulation capabilities only standard normal distributed covariates can be considered.
5. Online help is not available.

In the following sub-sections we describe the dialogs which relate to the multinomial probit model. In focussing upon this example, we are able to cover a range of specification issues that apply to other models within DCM.

## 9.2 Model Type

We first select the type of model to be estimated. The available models reside in the "Model" menu in DCM.



The dialog boxes that follow are model specific, in the sense of reflecting the variables and options relevant for the chosen model.

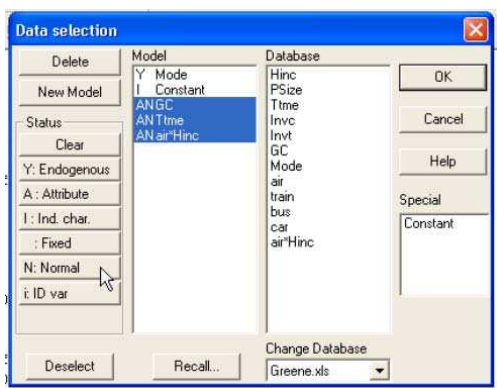
## 9.3 Variable selection and setting random coefficient distributions

The "Data selection" dialog presents the variables contained in the loaded database. Variables are selected into the model by double-clicking the variable name, or selecting the variable and then click the "Add" button. This is done by selecting the variable in the model and clicking the relevant status button on the left. There are three main groups of variables. The first selected variable will automatically be set to the dependent variable and a constant will be added to the model as an individual characteristic<sup>15</sup>. This constant will serve as alternative specific constants. Once a set of variables has been selected into the model, the user needs to specify the group affiliation of each variable. A letter indicating each variable's affiliation is showed directly to the left of the variable name.

In some cases, there are additional buttons indicating the available options for coefficient distribution. Selecting a variable and clicking the relevant button defines the distribution. This setting is indicated in the model variable field.

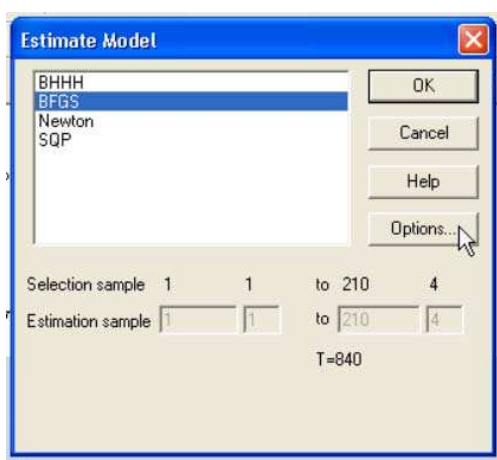
Previously specified models can be retrieved using the "Recall..." button below. Variables can be deleted from the model using the "Delete" button on top. After the model is specified, clicking on "OK" takes the user to the next dialog. In the example below we set the distribution of the coefficients of *Ttme*, *GC*, and *air \* Hinc* to normal.

<sup>15</sup>Note: an exception to this is the ordered probit model.



## 9.4 Estimation and model options

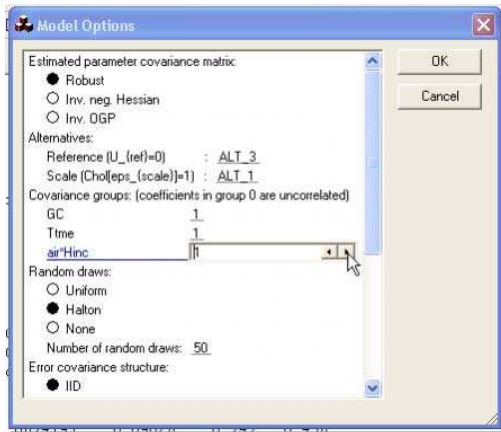
In the "Estimate Model" dialog the user can set the optimization algorithm and other model options. To use the default settings press "OK" and the model will be estimated.



In the "Model Options" dialog, the user can set model specific options. For example, for the conditional/multinomial logit model there are two available options; the type of covariance estimator and the reference alternative. Clicking on the radio buttons sets the covariance estimator. The reference alternative is set by choosing a specific alternative from the list provided. In this dialog the user can specify the correlation structure of the random coefficients. The coefficients which are marked as random in the Model selection dialog are listed in the Options dialog where the user can set the relevant group for each coefficient. Coefficients listed in the same group may correlated with the exception of the

special group 0 which is assumed to include only uncorrelated coefficients.

Related to the optimization procedure, a number of settings control the maximum number of iterations, type of printout during optimization, and convergence criteria. After the options have been set, clicking on "OK" takes the user back to the "Estimate Model" dialog.



After the model has been estimated, the output will appear in the GiveWin output window. The user may now re-estimate the model using a different set of variables and/or options.

## 10 Notes, remarks, and history

### 10.1 Bug reports and version history

**Version 1.0, released June 2004** • First version. Substantial amount of bug fixes and developments since beta version. Not listed.

### 10.2 Future plans

## A Technical appendix

### A.1 Inference on data structure

DCM infers the form of data structure from the selected dependent. The main issue is to determine whether records in the database include single or multiple rows. To do this DCM initially checks for multiple occurrences of the dependent variable in the set of variable names in the database. DCM then checks the values of the dependent variable and infers the data structure, the number of

alternatives, individuals and time periods. In what follows, we present the general outline of this procedure.

- 1. Dependent Variable as a single column:** If there is a single variable in the database named as the dependent variable DCM will continue to check if the maximum value of the dependent variable is 1 or larger.

**Maximum value is 1:** If the maximum value is 1 we may have the case that the data represents a binomial model where the dependent variable is recorded as a 1 or 0. This is a common but special case where each row in the data corresponds to one record, i.e., a "single row" data structure. In order to differentiate between this instance and one where the dependent variable is an indicator but records span multiple rows, DCM analyzes the ratio between the number of rows in the dataset and the number of 1's in the dependent variable. If the number of rows is a multiple of the number of 1's is zero, this indicates that the data is structured with multiple rows per record, otherwise it is structured as "single row".<sup>16</sup>

If the structure corresponds to the "multiple row" case, the number of alternatives  $J$  is derived as the number of rows in the dataset divided by the number of 1's in the dependent variable. The number of individuals  $N$  is then derived as the the number of rows in the data divided by the number of time periods and alternatives (if cross-section or balanced panel); or the number of unique values in the individual ID variable set by `SetId(sVar)` (if unbalanced panel). If the structure is "single row", the number of alternatives  $J$  is 2, the number of individuals  $N$  is the number of rows divided by the number of time periods or the number of unique values in the individual ID variable.

**Maximum value is greater than 1:** If the maximum value of the dependent variable is greater than 1, it should not be a indicator. DCM will then infer that the dependent variable provides the *index* of the preferred choice and that the data structure is of the "single row" type. DCM also checks if the indexing starts at 0 or 1, and adjusts the indexing accordingly. The number of alternatives  $J$  is determined by the difference between the largest and smallest number of the indexes, plus 1.<sup>17</sup> The number of individuals  $N$  is the number of rows divided by the number of time periods or the number of unique values in the individual ID variable.

---

<sup>16</sup>Note that this is not completely true as there might be datasets where the structure is of "single row" type but it happens (by chance) that the number of observations is a multiple of the number of 1's. In that unlikely case, the user need to re-structure the dataset to Type 2 manually by redefining the dependent variable.

<sup>17</sup>Hence, DCM assumes that alternatives are numbered in a sequence of real numbers.

**2. Multiple dependent columns:** If there exist multiple columns with the same name as the selected dependent variable, DCM infers that the dependent variable, for a given individual, is a row vector of zeros and a single non-zero value indicating the preferred alternative; the number of elements in this vector determines the dimension of the choice set,  $J$ . Further, DCM assumes that the data structure is of the "single row" type. If an individual identifier variable is set using `SetID(sVar)` (for unbalanced panels), the number of unique values in this vector defines  $N$ . If no individual ID variable is set, DCM sets the number of individuals to the number of rows in the dataset divided by the number of time periods set by `SetPanel(cT)`.

## A.2 Covariance matrix Specification in Multinomial Probit Models.

In what follows,  $\Sigma_\beta$  refers to the covariance matrix of the random coefficients and  $\Sigma_\varepsilon$  refers to the error covariance matrix. DCM utilises two additional indicator matrices:  $\mathbf{S}_\beta$  and  $\mathbf{S}_\varepsilon$ , which indicate whether the corresponding elements of  $\Sigma_\beta$  and  $\Sigma_{\varepsilon,J}$  are free (estimated:  $> 0$ ) elements or fixed (not estimated:  $= 0$ ).<sup>18</sup> For example, in the case where all mean coefficients are fixed we set  $\mathbf{S}_\beta$  to a null matrix. In specifying a heteroscedastic (uncorrelated) error covariance matrix, then  $\mathbf{S}_\varepsilon$  would be set equal to an identity matrix with zeros in the diagonal positions of the base and scale alternatives. The indicator matrices are set either by default values, or by the user calling the `SetCoeffDist()` and/or `SetErrDist()` procedures.

In order to impose non-negative definiteness and symmetry, we parameterize the lower diagonal of the cholesky decomposition of both  $\Sigma_\beta$  and  $\Sigma_\varepsilon$ , so that e.g.  $\Sigma_\beta = \mathbf{C}_\beta \mathbf{C}'_\beta$ . The vectorization is performed by stacking the columns of the lower diagonal: in the case of  $\Sigma_\beta$   $\vec{\mathbf{C}}_\beta$  denotes a  $Q(Q+1)/2 \times 1$  vector, where  $Q$  indicates the number of mean coefficients. For  $\Sigma_\varepsilon$  the same vectorisation results in the  $J(J+1)/2 \times 1$  vector  $\vec{\mathbf{C}}_\varepsilon$ . Note that some elements in  $\vec{\mathbf{C}}_\beta$  and  $\vec{\mathbf{C}}_\varepsilon$  are fixed (not estimated). We construct the vectorizations of  $\mathbf{S}_\beta$  and  $\mathbf{S}_\varepsilon$  in the same way.

**Initialization of  $\Sigma_\varepsilon$ :** In `InitPar()`, we initially set  $\Sigma_\varepsilon = I_J$ . Depending on the assumed structure of the error covariance matrix we fix the following elements:<sup>19</sup>

**IID:** All elements in  $\Sigma_\varepsilon$  are fixed. For example, in a trinomial choice model the

<sup>18</sup>In the code these indicator matrices are named `m_mFreeParCov` and `m_mFreeErrCov`, respectively. Also note that a value of 1 in the  $\Sigma_\beta$  matrix indicates normal distribution and (-)2 indicates (negative) log-normal distribution and non-zero off-diagonal elements indicates the covariance terms to be estimated. Hence, the term "covariance matrix" for  $\Sigma_\beta$  is somewhat misleading, since it will simultaneously hold the covariances between blocks of normal and log-normal distributed coefficients.

<sup>19</sup>This is actually done in the `SetErrDist()` procedure.

$\Sigma_\varepsilon$  matrix will have the following structure:

$$\Sigma_\varepsilon = \begin{pmatrix} 1^b & \cdot & \cdot \\ 0^b & 1^s & \cdot \\ 0^b & 0^* & 1^* \end{pmatrix}.$$

Where the superscript  $b$  indicates that the entry is fixed at this value because its the *base alternative*,  $s$  indicates that the entry is fixed because it is the *scale alternative*, and  $*$  indicates other fixed entries

### Heteroscedastic

- All elements in the row and column corresponding to the *base alternative* are fixed. (Note: this is the standard normalisation for location)
- All off-diagonal elements of the remaining  $(J - 1 \times J - 1)$ - matrix are fixed at 0.
- The diagonal element corresponding to the *scale alternative* is fixed at 1.
- The start values for the remaining  $J - 2$  diagonal elements are set to 1.

The resulting  $\Sigma_\varepsilon$  matrix is given below:

$$\Sigma_\varepsilon = \begin{pmatrix} 1^b & \cdot & \cdot \\ 0^b & 1^s & \cdot \\ 0^b & 0^* & 1 \end{pmatrix}$$

### Unrestricted

- All elements in the row and column corresponding to the *base alternative* are fixed.
- The diagonal element corresponding to the *scale alternative* is fixed at 1.
- The start values for the remaining  $J(J - 2)/2 - 1$  elements are set to the corresponding values in  $J$ , i.e., diagonal elements at 1 and off-diagonal elements at 0.

The resulting  $\Sigma_\varepsilon$  matrix is given below:

$$\Sigma_\varepsilon = \begin{pmatrix} 1^b & \cdot & \cdot \\ 0^b & 1^s & \cdot \\ 0^b & 0 & 1 \end{pmatrix}$$

**Is this formulation of  $\Sigma_\varepsilon$  identified?** The most general structure of  $\Sigma_\varepsilon$  in the 3 alternative case in DCM is the following:

$$E(\varepsilon\varepsilon') = \Sigma_\varepsilon = \begin{pmatrix} 1^b & \cdot & \cdot \\ 0^b & 1^s & \cdot \\ 0^b & \sigma_{23} & \sigma_{33} \end{pmatrix} \quad (1)$$

**Comments:** The order condition holds since we have  $J(J-1)/2 - 1 = 2$  free covariance parameters in  $\Sigma_\varepsilon$ . However, if we consider the discrete information observed by the analyst as imperfect measures on an underlying latent (utility) model, with the many-to-one observational rule

$$y = \kappa(\mathbf{y}^*), \quad (2)$$

where  $y$  is the index of the maximum element of the vector  $J \times 1$  vector  $\mathbf{y}^*$ , then the realisation that the analyst will only observe the sign of  $y_j^* - y_{j'}^* \forall j \neq j' \in \Omega_J$ , will have implications for the identification of the location and scale of the model. Namely, both location and scale are defined with respect to differences  $y_j^* - y_{j'}^*$ .

To see this consider the following transformation. Begin with a  $J \times J$  identity matrix, and construct a matrix,  $\psi_{j'}$ , deleting row  $j'$ , and replacing column  $j'$  with a vector of  $-1$ 's. Using  $\psi_{j'}$ , we now construct  $\Sigma_{\varepsilon J-1}$  from the original covariance matrix, using the transformation  $\Sigma_{\varepsilon J-1} = \psi_{j'} \Sigma_\varepsilon \psi_{j'}'$ . Taking the difference with respect to alternative 2, we get

$$E(\tilde{\varepsilon}\tilde{\varepsilon}') = \psi_2 \Sigma_\varepsilon \psi_2' = \begin{pmatrix} 1^s + 1^b & \cdot \\ \sigma_{32} + 1^b & \sigma_{33} + 1^b \end{pmatrix} \quad (3)$$

where  $\tilde{\varepsilon} = ((\varepsilon_1 - \varepsilon_2)(\varepsilon_3 - \varepsilon_2))'$ .

By subtracting the fixed value of variance of the base alternative ( $1^b$ ) from each of the entries in (3), we see that we can retrieve the entries in the original levels error covariance matrix in (1). Hence, the covariance specification is identified. The sum of the variances of the base and scale alternative will set the overall scale of the model.

### A.3 Estimated standard errors

- Standard errors can be calculated using the inverse of the negative Hessian,
- inverse of the outer gradient product
- or using robust estimation.

Table 23: Variable notation

	Description
$N$	Number of individuals
$T$	Number of time periods in balanced panels
$O$	Total number of observations in database ( $NT$ in balanced panels)
$J$	Number of alternatives
$K$	Number of individual characteristics
$K^*$	Number of individual characteristics with random coefficients
$L$	Number of attributes
$Q$	Total number of estimated parameters (model dependent)
$C$	Total number of estimated mean coefficients
$C^*$	Total number of estimated random mean co- efficients

## B DCM member functions

Below we list the member functions of the DCM class. We have separated the functions into two sections. The first section lists the functions called by the user, whereas the second section lists the internal functions. The functions are listed in alphabetical order. The DCM code uses *Hungarian notation*, meaning that it uses cases and prefixes which indicate type and scope. We refer the reader to the Ox manual for a description of Hungarian notation. In what follows, we will use notation presented in Table 23 for frequently used variables.

### B.1 Exported member functions

#### DCM::Citation

```
Citation();
```

-

*Return value:*

-

*Description:*

Prints the citation information of the manual and accompanying paper in BibTeX style. Insert the printout in the Bib-file for easy citation in papers which use DCM.

#### DCM::DCM

DCM();

-

*Return value:*

-

*Description:*

Constructor function.

### **DCM::DeSelectByName**

DeSelectByName (const sVar, const iGroup, const iLag);

sVar in: string, name of variable to de-select.

iGroup in: int, group where sVar is included.

iLag in: int, lag-length to de-select.

*Return value:*

-

*Description:*

De-selecting variables from group iGroup. This overrides Modelbase::DeSelectByName() in order to handle the situation where variable names are not unique.

### **DCM::Deterministic**

Deterministic (bDet);

bDet in: boolean, TRUE/FALSE (default).

*Return value:*

-

*Description:*

Creates alternative specific dummies. The default names of the dummies are ALT\_0, ALT\_1, .... See also SetAltNames().

### **DCM::Generate**

Generate();

-

*Return value:*

-

*Description:*

Generates the dependent variable if the simulation model is properly specified. The dependent variable is named "y" and automatically appended to the database. The dependent variable is automatically selected into the model.

## DCM::Get...

Get...-

-

*Return value:*

<i>Function call:</i>	<i>Return value:</i>
GetAlgorithm()	string with optimization routine, e.g. "BHHH".
GetAllResults()	$5 \times 1$ array with parameter names, estimates, standard errors, log likelihood value, and number of observations.
GetAltNames()	$J \times 1$ array of strings (names of alternatives)
GetRefAlt()	index of base alternative ( <i>m_iBaseAlt</i> ).
GetCoeffType()	$C \times 1$ array of strings. The type of coefficient distribution.
GetCorr()	$2 \times 1$ array. The first element returns the correlation matrix of random coefficients, and the second element returns the correlation matrix of errors.
GetCov()	$2 \times 1$ array. The first element returns the covariance matrix of random coefficients, and the second element returns the covariance matrix of errors.
GetcT()	number of observations <i>NT</i> .
GetErrDist()	$J \times J$ matrix with non-zero values in positions of free error covariance elements.
GetNest()	<i>#nests</i> $\times 1$ array of vectors with nesting structure.
GetParDist()	$C \times C$ matrix with 0 in diagonal positions of non-estimated coefficients, 1 in diagonal positions of normal distributed coefficients, 2 in diagonal position of log-normal distributed coefficients. Non-zero off-diagonal elements indicates estimated covariances.
GetRandom()	string, type of random draws.
GetScaleAlt()	returns index of scale alternative ( <i>m_iScaleAlt</i> ).

*Description:*

Some Get... functions can only be called after a successful estimation.

## DCM::GetEveryVar

GetEveryVar(*asName*);

**asName** in:  $k \times 1$  array of strings. Names of variables in data base.

*Return value:*

Returns a  $n \times k^*$  matrix where  $n$  is the number of rows and  $k^*$  is the number of columns in the database representing the variables indicated by *asName*.

*Description:*

`GetEveryVar()` checks the entire database for multiple occurrences of the elements in *asName*. As the `GetVar(asNames)` command only returns the first occurrence of the elements in *asNames*, it cannot be used in the case where several columns in the data base have identical names. `GetEveryVar(asNames)` ensures that all columns are returned.

### **DCM::Interact**

`Interact(const asI, const asA);`

**asI** in:  $k \times 1$  array of strings.

**asA** in:  $l \times 1$  array of strings.

*Return value:*

-

*Description:*

Appends and includes the *kl* interaction terms constructed from the variables listed in *asI* and *asA* in the model. `RemoveInteract()` removes all interaction terms. Re-initialization of data may be required. Multiple calls to `Interact()` accumulates interaction terms.

### **DCM::Load**

`Load(const sDatafile);`

-

*Return value:*

-

*Description:*

Loads the data file *sDatafile* and calls `SetDbName(sDatafile)`.

### **DCM::OutputLaTeX**

`OutputLaTeX(...);`

- aMO* optional in:  $5 \times 1$  array. *aMO[0]*  $q \times 1$  array of parameter names, *aMO[1]*  $q \times 1$  vector of estimates, *aMO[2]*  $q \times 1$  vector of estimated standard errors, *aMO[3]* scalar with final log likelihood, *aMO[4]* scalar with no. of observations.
- aM1* optional, as *aMO*, and refer to another model estimation.

If no input arguments, DCM will use the most recent estimation results to create a table.

*Return value:*

-

*Description:*

Produces a LaTeX table which can be inserted directly into a LaTeX document. Use of this command with multiple arguments creates one multicolumn table combining all results. Asterix indicates significance levels 10 and 1 per cent. Table .

### **DCM::RemoveInteract**

`RemoveInteract()` ;

-

*Return value:*

-

*Description:*

Removes all interaction terms.

### **DCM::ScaleVar**

`ScaleVar (const aScaleVar)` ;

*aScaleVar* in:  $2p \times 1$  array `{sVar,dScale}` where *sVar* is a string and *dScale* is a  $p \times 1$  vector of scaling factors.

*Return value:*

-

*Description:*

Scale the variables listed in *aScaleVar* according to the associated *dScale* factors. To re-set all scalings use `ScaleVar()` ;

*Example:* `ScaleVar("Var1",0.01,"Var2",0.1)` will result in the transformation  $0.01 * Var1$  and  $0.1 * Var2$  (if previously selected or created using `Interact()`).

### DCM::SetAlgorithm

`SetAlgorithm (iAlgorithm);`  
`iAlgorithm` in: int. Specifies the optimization algorithm to be used in estimation. See options below.

*Return value:*

-

*Description:*

`BHHH` BHHH optimizer that comes with DCM (default).  
`BFGS` Ox's intrinsic MaxBFGS optimizer.  
`NEWTON` Ox's intrinsic MaxNewton optimizer.

### DCM::SetAltNames

`SetAltNames (const asAltNames);`  
`asAltNames` in: an array of strings, providing names for alternatives in the choice set.

*Return value:*

-

*Description:*

Sets the names of the alternatives.

### DCM::SetCoeffDist

`SetCoeffDist (const iType, const asPar, const iGroup);`  
`iType` in: int, type of distribution. See options below.  
`asPar` in: an array of strings, providing names of mean coefficients.  
`iGroup` in: int (optional), coefficients included in the same group (except group 0) are allowed to correlate.

*Return value:*

-

*Description:*

Sets the mixing distribution of specific coefficients in the model, together with the correlation structure of the random coefficients. Multiple calls to `SetCoeffDist()` accumulates distributions.

**FIXED**                Fixed (non-random) coefficients (default).  
**NORMAL**                Normal distributed coefficients.  
**(-)LOGNORMAL**        Log-normal distributed coefficients. Note that the log-normal distribution has a positive support. Hence, to model e.g. a negative price-effect, create a price variable with a *negative* support or prefix with a "-" sign.

*Example:* `SetCoeffDist(NORMAL, {"Par1", "Par2"}, 1);`

*Example:* `SetCoeffDist(-LOGNORMAL, {"Par3"}, 1);`

sets the distributions of parameters "Par1" and "Par2" to multivariate normal with non-zero correlation, and the distribution of "Par3" to univariate log-normal with negative support of distribution. All three coefficients are allowed to correlate as they are included in the same group (=1).

### **DCM::SetDim**

`SetDim(cN, cT, cJ);`

`cN` in: scalar, no. of individuals

`cT` in: scalar, no. of time periods

`cJ` in: scalar, no. of alternatives

*Return value:*

-

*Description:*

Sets the dimension of the database when simulating discrete choice data. The command creates the database and appends a temporary variable that indicates the structure of the database.

### **DCM::SetErrDist**

`SetErrDist(const iType);`

`iType` in: int, type of error covariance matrix in MNP models. See options below.

*Return value:*

-

*Description:*

Sets the structure of the error covariance matrix. See Appendix A.2 for a comprehensive discussion on the error covariance structure in MNP models.

IID iid errors. No free error variance-covariance elements (default).

HETEROSC Heteroscedastic errors.  $J - 2$  free error variance; no free covariance elements.

UNREST Unrestricted errors.  $J(J - 1)/2 - 1$  free error variance-covariance elements.

### DCM::SetID

SetDim(*sVar*);

*sVar* in: string, variable name of individual identifier

*Return value:*

-

*Description:*

For unbalanced panels (and balanced panels in OxPack) DCM needs one variable that identifies the individual. This variable is set by `SetID()`. DCM will use this variable to make inference on database structure.

### DCM::SetModel

SetModel(const iModel);

iModel in: int, type of model. See options below.

*Return value:*

-

*Description:*

Sets the type of model. The options are:

CL	Conditional/multinomial logit (default)	$(J - 1)K + L = C$
NL	Nested logit	$C + \#nests$
MXL	Mixed logit	$C + C(C + 1)/2$
MXP	Mixed probit	$C + C(C + 1)/2$
MNP	Multinomial Probit	$C + C(C + 1)/2 + J(J - 1)/2 - 1$
OP	Ordered probit	$C + J - 1$
OMP	Ordered mixed probit	$C + C(C + 1)/2 + J - 1$

### DCM::SetNest

SetNest(const avNest);

avNest in: array of vectors, nesting structure.

*Return value:*

-

*Description:*

Sets the nesting structure.

*Example:* `SetNest(<0>, <1,2,3>)` puts alternative 0 in one nest and alternatives 1, 2, and 3 in a second nest. A large number of nests can be created, but the beta version only supports a single level of nesting, i.e., the beta-version does not support nests within nests.

### **DCM::SetPanel**

`SetPanel (const cT);`

`cT` in: int, number of time periods in balanced panel data. Default value=1.

*Return value:*

-

*Description:*

Note that the panel features in DCM 1.0 are quite primitive. The basic effect of setting the panel is that random draws are repeated across time periods for each individual.

### **DCM::SetRandom**

`SetRandom(const iType, const cR);`

`iType` in: int, type of random numbers.

`cR` in: int, number of random draws used in simulations. Default value=50.

*Return value:*

-

*Description:*

Sets the types and number of random draws.

**UNIFORM** Standard uniform pseudo-random numbers.

**HALTON** Halton sequence based on primes (default).

**NONE** Use Ox intrinsic integration routines based upon multivariate normal probabilities.

### **DCM::SetRefAlt**

`SetRefAlt (const iRefAlt);`

`iRefAlt` in: int, index of base alternative  $0, \dots, J - 1$ .  
Default value=0.

or: string, name of reference alternative.

*Return value:*

-  
*Description:*

The `iRefAlt` is used as the reference alternative. The alternative specific coefficient (associated with an individual characteristic) for the `iRefAlt` alternative is set to zero. The off-diagonal elements in the `iRefAlt`'th row and column in the error covariance matrix will be set to zero, and the diagonal element will be set to unity. See Appendix A.2 for details concerning the error covariance matrix in MNP models.

### **DCM::SetScaleAlt**

`SetScaleAlt(const iScaleAlt);`

`iScaleAlt` in: int, index of scaling alternative. Default:  
second alternative (`iScaleAlt=1`).  
or: string, name of scale alternative

*Return value:*

-  
*Description:*

Sets the alternative used for setting the scale of MNP models. This cannot be the same as `iRefAlt` (see `SetRefAlt()`). The diagonal element in the cholesky decomposition of the error covariance matrix corresponding to alternative `iScaleAlt` is set to 1.

### **DCM::SetStartPar**

`SetStartPar(const vP);`

`vP` in:  $Q \times 1$  vector, starting values for iterative optimization algorithms.

*Return value:*

-  
*Description:*

Let there be  $J$  alternatives,  $K$  characteristics,  $L$  attributes, and  $M$  nests. The vector of start values is then organized as coefficients associated with  $J-1$  alternative specific constants,  $K(J-1)$  characteristics,  $L$  attributes,  $J-1$  thresholds,  $M$  inclusive values,  $K(K+1)/2$  elements in stacked columns of the lower cholesky decomposition of the coefficient covariance matrix, and  $J(J+1)/2$  stacked columns of the lower cholesky decomposition of the error covariance matrix. The vector of parameters can either conform with the number of free (estimated) parameters or the total number of model parameters.

### **DCM::SetStdErr**

**SetStdErr(const iType);**

**iType** in: integer, type of estimated coefficient standard errors.

*Return value:*

-

*Description:*

Options for **iType** are  
**ROBUST** Robust standard errors  
**HESSIAN** Standard errors based on final inverse of negative Hessian.  
**OGP** Standard errors based on final inverse of outer gradient product.

### **DCM::SetTruePar**

**SetTruePar(const vPar);**

**vPar** in:  $Q \times 1$  vector of parameters.

*Return value:*

-

*Description:*

Sets the vector of true parameter used by **Generate()** when simulating discrete choice data. See **SetStartPar(vPar)** for organization.

### **DCM::TransformData**

**TransformData(const sVar,const sOutfile);**

**sVar** in: string, name of dependent variable in database

**sOutfile** in: string, name of output file. Extension indicates file type.

*Return value:*

-

*Description:*

Creates and saves the database in a different structure. The command toggles between "single row" and "multiple row" data structures. The "multiple row" structure is more convenient for variable transformation, whereas "single row" structure takes up less memory. The new transformed database automatically replaces the old database in the object.

### DCM::TestRandCoeff

TestRandCoeff(const *asPar*);

*asPar* in: array of strings or int, names of coefficients.

*Return value:*

-

*Description:*

Tests the coefficients listed in *asPar* for unobserved heterogeneity. If *asPar*=-1 all coefficients are tested.

## B.2 Non-exported member functions

### DCM::Covar

Covar ();

-

*Return value:*

Returns TRUE if successful, FALSE otherwise

*Description:*

Estimates the covariance matrix of parameter estimates and sets the value of the  $Q \times Q$  covariance matrix *m\_mCovar*. The covariance matrix is calculated from the inverse of the negative Hessian at the point of convergence. If this matrix does not converge, *m\_mCovar* is constructed from the inverse of the outer gradient product.

### DCM::fCLogit

fCLogit (const *vP*, const *adFunc*, const *avScore*, const *amHess*);

*vP* in:  $Q \times 1$  vector of parameter values.

*adFunc* in: address of variable.

out:  $NT \times 1$  vector of individual contributions to sample log-likelihood evaluated at *vP*.

*avScore* in: address of variable, or 0.

out:  $NT \times Q$  matrix of individual contributions to sample score evaluated at *vP* if *avScore*≠0, 0 otherwise.

*amHess* in: address of variable, or 0.

out: Sample Hessian evaluated at *vP* if *amHessian*≠ 0, 0 otherwise.

*Return value:*

Returns **TRUE** if successful, **FALSE** otherwise.

*Description:*

Calculates individual contributions to sample log-likelihood values and sample scores. The same structure of arguments applies to `fNestedLogit()`, `fMixedLogit()`, `fProbit()`, `fOProbit()`, `fOMProbit()`. The member functions with the postfix "SQP" are wrapper functions adapted to Ox maximization functions where the sample log-likelihood (double valued) and score (vector valued) is required.

### **DCM::fGHK**

`fGHK(const mD, const mW, const mU, const avV);`

`mD` in:  $J - 1 \times 2$  matrix with integration limits.

`mW` in:  $J - 1 \times J - 1$  covariance matrix.

`mU` in:  $J - 1 \times R$  matrix with pseudo random draws.

`avV` in: address of variable.

out:  $J - 1 \times R$  matrix of simulated truncated multivariate normal draws.

*Return value:*

Returns a  $1 \times R$  vector with simulated normal truncation probabilities.

*Description:*

Implements the Geweke-Hajivassiliou-Keane (GHK) simulator.

### **DCM::fHalton**

`fHalton (const r, const c);`

`r` in: int, number of rows.

`c` in: int, number of columns

*Return value:*

Returns the  $r \times c$  matrix of Halton draws based on the first  $r$  primes and dropping the first 10 elements in the sequence.

*Description:*

Generates a Halton sequence based on the first 13 primes.

### **DCM::fMapFreetoPar**

`fMapFreetoPar (const vFreePar);`

`vFreePar` in:  $Q \times 1$  vector of free (estimated) parameters.

*Return value:*

If `m_iModel=`

CL	$1 \times 1$ array, {vBeta}
OP	$2 \times 1$ array, {vBeta,vAlpha}
OMP	$3 \times 1$ array, {vBeta,vAlpha,mCbeta}
NL	$2 \times 1$ array, {vBeta,vLambda}
MXL	$2 \times 1$ array, {vBeta,mCbeta}
MNP	$3 \times 1$ array, {vBeta,mCbeta,mCeps}

where

vBeta	out: $C \times 1$ vector of mean coefficients.
vAlpha	out: $J - 1 \times 1$ vector of threshold estimates.
vLambda	out: $\#nests \times 1$ vector of inclusive values.
mCbeta	out: $C \times C$ matrix. Lower triangular cholesky decomposition of random coefficient covariance matrix.
mCeps	out: $J \times J$ matrix. Lower triangular cholesky decomposition of error covariance matrix.

*Description:*

Maps free (estimated) parameters to the full set of model parameters.

### DCM::GetStartPar

GetStartPar()

-

*Return value:*

Returns TRUE if successful, FALSE otherwise.

*Description:*

This function creates starting values for MXL, MXP, MNP, and OMP models using CL for unordered and OP for ordered models. The starting values of standard deviations of random coefficients are set to 0.1 and 0 for covariance terms. The starting values for error standard deviations is set to 1 and 0 for covariances. DCM creates a temporary object to perform the estimations.

### DCM::InitData

InitData();

-

*Return value:*

Returns TRUE if successful, FALSE otherwise.

*Description:*

Initializes the matrices of dependent and independent variables using the information submitted in the `Determinstic()`, `Select()`, and `Interact()` com-

mands. The internal organization of the data matrices are  $NTJ \times K + L + 1$  for unordered models and  $NT \times K + L + 1$  for ordered models.

### **DCM::InitPar**

InitPar ();

-

*Return value:*

Return TRUE if successful, FALSE otherwise.

*Description:*

Initializes the relevant parameters for the specified model.

### **DCM::IsUnivariate**

IsUnivariate();

-

*Return value:*

Returns FALSE

*Description:*

Some data organizations in discrete choice models require multiple columns for the dependent variable. If DCM encounters this form of data this function returns FALSE.

### **DCM::SetParDist**

SetParDist (const mFreeParCov, ...);

mFreeParCov in: int, or  $C \times 1$  vector, or  $C \times C$  matrix, with integers defining the coefficient distributions.

*Return value:*

-

*Description:*

Internal function handling the bookkeeping of random coefficients. Sets the member variable m\_mFreeParCov.

## References

- BEN-AKIVA, M., D. BOLDUC, AND J. WALKER (2001): "Specification, Identification and Estimation of the Logit Kernel (or Continuous Mixed Logit) Model," Working Paper, MIT. <http://web.mit.edu/jwalker/www/home.htm>. Accessed April 1, 2002.
- DAGANZO, C. (1979): *Multinomial Probit: The Theory and Its Application to Demand Forecasting*. Academic Press.
- DUNCAN, A., AND M. WEEKS (1998): "Non-Nested Models of Labour Supply with Discrete Choices," Working Paper, Department of Economics, University of York .
- EKLOF, M., AND M. WEEKS (2004): "Estimation of Discrete Choice Models Using DCM for Ox," CWPE 0427, Department of Applied Economics, University of Cambridge.
- GREENE, W. H. (2002): *Econometric Analysis Fifth Edition*. Prentice Hall, Upper Saddle River, New Jersey.
- LOUVIERE, J. J., D. A. HENSHER, AND J. D. SWAIT (2000): *Stated Choice Methods: Analysis and Application*. Cambridge University Press.