

**ALOGIT Software & Analysis Ltd.**

## **ALOGIT 4.5 User Documentation**

Andrew Daly, 9 September 2022

# Table of Contents

<b>Table of Contents .....</b>	<b>1</b>
<b>1. Introduction .....</b>	<b>5</b>
<b>1.1 Function and operation of ALOGIT .....</b>	<b>5</b>
<b>1.2 The logit model and its generalisations in ALOGIT .....</b>	<b>6</b>
1.2.1 Size variables .....	7
1.2.2 Tree structure (nesting) .....	8
1.2.3 Error components (mixed logit) .....	10
1.2.4 Limitations of ALOGIT specifications .....	12
1.2.5 Estimation procedure .....	12
<b>1.3 Development, ownership and origin of ALOGIT .....</b>	<b>13</b>
<b>1.4 Purpose and organisation of the User Documentation .....</b>	<b>14</b>
<b>2. The ALO File .....</b>	<b>15</b>
<b>2.1 Components of the ALO file .....</b>	<b>15</b>
2.1.1 INCL.FILE Control .....	15
2.1.2 Comment and blank lines .....	16
2.1.3 \$ Commands .....	16
2.1.4 File specifiers .....	16
2.1.5 Data transformations .....	17
2.1.6 Utility and Size Functions .....	17
<b>2.2 Characters and Lines .....</b>	<b>18</b>
2.2.1 Characters .....	18
2.2.2 Lines .....	18
<b>3. File Commands .....</b>	<b>20</b>
<b>3.1 The main data file .....</b>	<b>20</b>
<b>3.2 File command syntax .....</b>	<b>21</b>
File specification .....	21
Variable list .....	21
<b>3.3 File subcommands .....</b>	<b>21</b>
3.3.1 BINARY subcommand .....	22
3.3.2 FOLLOWS subcommand .....	22
3.3.3 HANDLE subcommand .....	22
3.3.4 INTEGER subcommand .....	22
3.3.5 KEY subcommand .....	23
3.3.6 MATRIX subcommand .....	23
3.3.7 NAME subcommand .....	24
3.3.8 NOBS subcommand .....	24
3.3.9 OUTPUT subcommand .....	24
3.3.10 SCENARIO subcommand .....	24
3.3.11 SKIP1 subcommand .....	25
3.3.12 TRANSPOSE subcommand .....	25
3.3.13 Interaction of FILE subcommands .....	25
<b>3.4 File format specification .....</b>	<b>25</b>

3.4.1	Free format.....	26
3.4.2	Explicit fixed format.....	26
<b>3.5</b>	<b>Variable list syntax .....</b>	<b>28</b>
<b>4.</b>	<b>\$ Commands .....</b>	<b>29</b>
<b>4.1</b>	<b>\$ALGOR command.....</b>	<b>29</b>
<b>4.2</b>	<b>\$ARRAY command.....</b>	<b>30</b>
<b>4.3</b>	<b>\$COEFF command.....</b>	<b>31</b>
<b>4.4</b>	<b>\$DIMENSION command.....</b>	<b>32</b>
<b>4.5</b>	<b>\$ESTIMATE command .....</b>	<b>32</b>
<b>4.6</b>	<b>\$GEN.STATS command .....</b>	<b>33</b>
<b>4.7</b>	<b>\$KEEP command.....</b>	<b>33</b>
<b>4.8</b>	<b>\$L_S_M command.....</b>	<b>34</b>
<b>4.9</b>	<b>\$NEST command.....</b>	<b>35</b>
<b>4.10</b>	<b>\$PRINT command.....</b>	<b>35</b>
<b>4.11</b>	<b>\$ROW command .....</b>	<b>37</b>
<b>4.12</b>	<b>\$TAB.VARS command .....</b>	<b>38</b>
<b>4.13</b>	<b>\$TITLE and \$SUBTITLE commands.....</b>	<b>39</b>
<b>4.14</b>	<b>\$VAL.LABEL .....</b>	<b>40</b>
<b>5.</b>	<b>Data Transformations.....</b>	<b>41</b>
<b>5.1</b>	<b>Basics of the ALOGIT language.....</b>	<b>41</b>
<b>5.2</b>	<b>Data items .....</b>	<b>42</b>
5.2.1	User-defined data items .....	42
5.2.2	System-defined data items .....	42
5.2.3	User-defined constants.....	43
5.2.4	System-defined constants.....	43
<b>5.3</b>	<b>Arrays .....</b>	<b>44</b>
5.3.1	User-defined arrays.....	44
5.3.2	System-defined arrays.....	44
<b>5.4</b>	<b>Conditional processing .....</b>	<b>44</b>
<b>5.5</b>	<b>Repetitive commands (DO loop).....</b>	<b>45</b>
<b>6.</b>	<b>Operators and Functions .....</b>	<b>47</b>
<b>6.1</b>	<b>Operators.....</b>	<b>47</b>
6.1.1	Mathematical operators.....	47
6.1.2	Logical operators .....	47
<b>6.2</b>	<b>Functions .....</b>	<b>48</b>
6.2.1	Mathematical Functions.....	48
6.2.2	Logical Functions.....	49
6.2.3	Random Functions .....	49
6.2.4	Additional Functions.....	50

<b>7.</b>	<b>Defining the model: Alternatives, UTIL and SIZE functions, Coefficients..</b>	<b>51</b>
7.1	Alternatives .....	51
7.2	Utility Functions.....	52
7.2.1	Non-random utility functions.....	52
7.2.2	Mixed logit: use of ERROR_COMP .....	53
7.3	Size Functions.....	56
7.4	Coefficients .....	58
<b>8.</b>	<b>The LOG file .....</b>	<b>59</b>
8.1	Introductory and closing material .....	59
8.2	ALOGIT numbered messages .....	59
8.2.1	INFORMATION.....	59
8.2.2	WARNING .....	60
8.2.3	USER ERROR.....	60
8.2.4	PROGRAM ERROR .....	60
8.2.5	OBSERVATION REJECTED.....	61
8.3	ALOGIT reports.....	61
8.3.1	Report of ALO file.....	61
8.3.2	Report of data processing.....	62
8.3.3	Model estimation reports .....	64
8.3.4	APPLY tables.....	66
<b>9.</b>	<b>The initialisation file.....</b>	<b>69</b>
9.1	Dynamic storage.....	69
9.2	Print settings.....	70
9.3	Test and communication settings .....	70
<b>APPENDIX A: ALOGIT system files .....</b>		<b>71</b>
A.1	The F11 file.....	71
A.2	The F12 file.....	71
A.3	The F14 file.....	73
<b>APPENDIX B: Previous versions and obsolescent features .....</b>		<b>74</b>
<b>B.1</b>	<b>Improvements in Versions 4.x .....</b>	<b>74</b>
	Control File.....	75
	Data Files .....	76
	Versions 4, 4.1, 4.2 and 4.3 (also EC variants) .....	76
	Version 4.5.....	77
	Status of Versions 3.8 and earlier .....	77
<b>B.2</b>	<b>Obsolescent features .....</b>	<b>78</b>
B.2.1	Coefficient specification .....	78
B.2.2	Utility and size functions .....	78
B.2.3	\$ELAST command .....	78
B.2.4	ERROR_COMP files.....	79
B.2.5	@USERINPUT .....	79

<b>APPENDIX C: The RU1 specification and ‘normalisation’ .....</b>	<b>80</b>
Example: adding a dummy nest in a 3-alternative model.....	81
Conclusion .....	82
<b>APPENDIX D: Program Size Limits .....</b>	<b>83</b>
<b>APPENDIX E: File Handling Error Messages .....</b>	<b>84</b>

## 1. Introduction

ALOGIT is a powerful computer program for studying and forecasting consumer choices, based on the use of simple and advanced ‘logit’ models.

ALOGIT exploits the ability of logit models to explain and predict many aspects of consumer behaviour, giving insight into the reasons for choice behaviour, finding and quantifying the main variables determining the choices made by consumers and allowing forecasts to be made of what they will choose in the future.

The main functions of ALOGIT are:

- preparing and testing data for logit modelling;
- estimating the unknown coefficients of a logit model;
- applying a logit model to test its validity or make forecasts.

ALOGIT can also be used for some data processing tasks.

Many applications of logit models have been made in the field of transportation planning, where it is essential to obtain forecasts of what travellers will do under changed circumstances. The power of ALOGIT is a major advantage in dealing with the large-scale models that are needed in transportation planning. However, in many other fields of consumer study, models of this type have been recognised as useful tools for gaining understanding and making predictions.

The following topics are covered in this Introduction:

- general function and operation of the program;
- logit models in ALOGIT –
  - generalisations of simple logit models,
  - the use of structural coefficients,
  - defining a tree choice structure and
  - mixed logit: the use of error components;
- development, ownership and origins of the program;
- organisation of the remainder of the documentation.

An overview of ALOGIT some supporting documentation is also given at [www.alogit.com](http://www.alogit.com).

### 1.1 Function and operation of ALOGIT

To help in the specification, estimation, improvement and application of logit models, ALOGIT is organised to perform three main functions, which are defined by the user in the control file:

- **PREPARE**: data is tested for suitability and prepared for estimating the coefficients appearing in the utility functions that describe the attractiveness of the alternatives from which consumers are modelled as choosing;
- **ESTIMATE**: unknown coefficients appearing in the model are estimated from the data;
- **APPLY**: the model is tested and/or applied for forecasting.

In addition, ALOGIT can also be used for simple data processing.

The main tasks of PREPARE are to specify a model for estimation and to amend, check and prepare the data. By ‘specification’ is meant the definition of the utility functions for each of the alternatives; additionally, alternatives may be defined to be unavailable to certain consumers under certain circumstances. The data is input from the base data file(s) to PREPARE. It may then be amended by transformations specified by the user. The transformed data is then checked to ensure its suitability for logit modelling and written out in a special compressed format to a temporary file (F11) for use by ESTIMATE.

The function of ESTIMATE, not surprisingly, is to estimate the values of the coefficients appearing in the utility functions. This is an iterative process, in which initial values are given to the coefficients, the data is read in from the F11 file, and a ‘step’ is calculated that should give better values of the coefficients. This is repeated until the step taken becomes smaller than a defined test criterion. The resulting coefficient values are stored in a file (F12) and a report is given (in the LOG file) of the results of the estimation.

APPLY can be used to test the quality of the model that has been estimated or to make forecasts. The main test that can be made is to apply the model under unchanged circumstances and compare the results obtained with what was actually observed in the data. Additionally, APPLY can be used to make forecasts of consumers’ behaviour if circumstances were to change. Comparisons can be made of predicted behaviour under a range of different circumstances.

In addition, it is possible to use ALOGIT for simple data processing. In particular, the user can take advantage of ALOGIT’s file linking and matrix manipulation abilities.

ALOGIT is generally run from within the ‘Shell’ program AlogShell<sup>1</sup>. This allows for the editing of control files, examination of the results of a series of models and offers other functions. Documentation of the Shell program will be provided separately in due course; at present an outdated but substantially correct version is available.

## 1.2 The logit model and its generalisations in ALOGIT

The logit model is used to analyse discrete choices. The ‘choosers’ may be individuals (consumers, travellers etc.) or households, companies, governments etc.. The essence is that we are able to predict the probability – *not* the certainty – that the chooser will select one of an exhaustive and exclusive set of alternatives. To be useful, the model must contain variables that influence these probabilities. Then, the simplest logit model predicts the probability  $p_j$  of choice of an alternative  $j$  from a choice set  $C$  as

$$p_j = \frac{\exp V_j}{\sum_{k \in C} \exp V_k} \tag{1}$$

Here,  $V_j$  represents the benefit to the chooser of choosing  $j$ ; similarly the values  $V_k$  represent the benefit of all of the alternatives in the set  $C$  available to the chooser. It can be seen from the form of the equation that the probabilities must all be positive, that they must add up to 1 and that the larger  $V_j$  relative to the other benefits the more probable it is that the choice will be  $j$ . Note that the set  $C$  can contain 2, 200 or 20,000 alternatives.

---

<sup>1</sup> This program was updated in 2022.

The model of choice (1) can be related to an underlying model of utility maximisation, in which each chooser picks the alternative that offers the highest utility. For this model, the utility for each alternative  $j$  must be defined to be  $V_j$  plus a random term, which is distributed identically but independently over the alternatives with a Gumbel (also called extreme value) distribution. The random utility interpretation of the model is helpful in a number of contexts and has a very wide application.<sup>2</sup> For consistency with the literature, however, we shall refer to the standard benefit  $V_j$  as “utility”, without being concerned with the full details of the distribution of individuals’ valuations around that number.

While the formula (1) gives only a probability for an individual chooser of making a specific choice, the advantage of the model is that it can give accurate forecasts of the behaviour of a population, while the ability to find the impact on choice probabilities of specific variables can help us understand behaviour in general and to attach values to the variables. This is done by including those variables in the utility  $V$  and estimating the importance of each variable from observations of choice.

In the simplest models,  $V$  is just a sum of variables multiplied by coefficients, so the utility  $V_j$  of alternative  $j$  is given by

$$V_j = \beta_1 x_{1j} + \beta_2 x_{2j} + \beta_3 x_{3j} + \dots = \sum_{k=1..K} \beta_k x_{kj} \quad (2)$$

for as many variables  $x_{1j} \dots x_{Kj}$  as we want to include<sup>3</sup> and for which we have measurements. Because of the simple ‘linear’ way in which the unknown coefficients  $\beta$  are included in  $V$  the model defined by equations (1) and (2) is called ‘linear logit’. However, it must be remembered that this applies to  $\beta$ , not to  $x$ , which may be any transformation, linear or nonlinear, of the basic data. The utility functions of equation (2) are set up in ALOGIT using UTIL() or U() lines in the control file, as is described in detail in Chapter 7 below.

Because of the symmetrical way in which the alternatives appear in equation (1), the model is called multinomial logit, often shortened to MNL, or binary logit (when there are just 2 alternatives). In this User Documentation, we use MNL as shorthand for multinomial or binary linear logit.

### 1.2.1 Size variables

In the MNL model, the variables  $x$  in the utility model describe the quality of the alternatives. The first generalisation in ALOGIT is to allow also variables that describe the size or quantity of each alternative. Size variables  $S$  for each alternative are introduced by extending equation (1) as follows

$$p_j = \frac{S_j \cdot \exp V_j}{\sum_{k \in C} S_k \cdot \exp V_k} = \frac{\exp(V_j + \log S_j)}{\sum_{k \in C} \exp(V_k + \log S_k)} \quad (3)$$

The theory of the way in which size variables work is described in a paper.<sup>4</sup> Essentially, they are variables such that, all other things being equal, the probability of choosing the alternative

<sup>2</sup> See Hess, S., Daly, A. and Batley, R. (2018) Revisiting consistency with random utility maximisation: theory and implications for practical work, *Theory and Decision*, January.

<sup>3</sup> In the utility it is also possible to include constants  $x$  without coefficients (i.e.  $\beta = 1$ ) and/or coefficients  $\beta$  without explicit data items (i.e.  $x = 1$ ).

<sup>4</sup> Daly, A.J. (1982) Estimating Choice Models Containing Attraction Variables, *Transportation Research Vol. 16B*, 1.

is proportional to its size. In ALOGIT, multiple variables can be included in the size of an alternative, using an equation like

$$S_j = s_{1j} + \gamma_2 s_{2j} + \gamma_3 s_{3j} + \dots = s_{1j} + \sum_{l=2..L} \gamma_l s_{lj} \quad (4)$$

Looking back at equation (3), we can see that if the total size of each alternative is multiplied by the same factor, then the choice probabilities will not change. To deal with this, the first size variable does not have a coefficient. So, in equation (4), if there is just one size variable, there are no  $\gamma$  coefficients to be estimated. However, if there are multiple size variables, then we need to estimate  $\gamma$  coefficients to obtain the *relative* importance of the size variables.

Size functions like equation (4) are set up in ALOGIT using SIZE() or S() lines in the control file, as is described in detail in Chapter 7 below.

Note that the total size  $S$  variables must be positive; this implies that  $\gamma$  must be positive and that none of the  $s$  variables can be negative. To keep  $\gamma$  positive, ALOGIT sets  $\gamma$  to be the exponential of another coefficient, and this other coefficient, which can be positive or negative, is used in the optimisation. ALOGIT checks that the individual  $s$  variables are not negative and that at least one of them is positive for each alternative.

A further slight generalisation to the way in which size variables are modelled is to introduce the coefficient  $\phi$ :

$$p_j = \frac{\exp(V_j + \phi \cdot \log S_j)}{\sum_{k \in C} \exp(V_k + \phi \cdot \log S_k)} \quad (5)$$

For obvious reasons,  $\phi$  is called the log size multiplier. In ALOGIT,  $\phi$  is controlled by the \$L\_S\_M command (see section 4.8). In most models,  $\phi$  will be set to 1, i.e. following equation (3) rather than (5), but sometimes this generalisation is useful.

A paper on shopping travel<sup>5</sup> gives a further explanation of these points.

### 1.2.2 Tree structure (nesting)

The second major generalisation of MNL offered by ALOGIT is to allow the alternatives to be related to each other in ways more sophisticated than the symmetry of equations (1), (3) and (5). This is achieved by allowing the alternatives to be grouped in ‘nests’ so that within nests the alternatives relate symmetrically to each other, but they can have a different relationship with alternatives in other nests. The probability of choice of an alternative is then the probability that the relevant nest will be chosen times the probability that the specific alternative will be chosen in the nest.

In a ‘tree’ logit model, the nests are defined so that they do not overlap, i.e. each alternative may appear in only one nest<sup>6</sup>. Then for each alternative  $j$  there is a unique pointer  $t_j$  which indicates which nest the alternative belongs to. The nests then have their own pointers, which may be the ‘root’  $r$  of the tree or may be another nest, giving a multi-level model.

---

<sup>5</sup> Kristofferson, I., Daly, A. and Algiers, S. (2018) Modelling the attraction of travel to shopping destinations in large-scale models, *Transport Policy*, Vol. 68, pp. 52-62.

<sup>6</sup> In ‘cross-nested’ logit models the nests may overlap, so that alternatives can appear in more than one nest. These models are not dealt with in current versions of ALOGIT.

Using the indexes  $t$ , this extension of the model than leads to an extended form of equation (1), e.g. for a two-level model:

$$p_j = \frac{\exp V_{t_j}}{\sum_{t_k=r} \exp V_k} \cdot \frac{\exp V_j}{\sum_{t_k=t_j} \exp V_k} \quad (6)$$

in which the first fraction gives the probability of choosing  $t_j$ , the nest that contains  $j$ , among all the nests of alternatives, and the second fraction gives the probability of choosing  $j$  within nest  $t_j$ . Equation (6) requires the definition of the utility  $V_{t_j}$  for nests and this is given in ALOGIT by

$$V_{t_j} = V_{t_j}^* + \theta \log \sum_{t_k=t_j} \exp V_k \quad (7)$$

In this equation  $V_{t_j}^*$  represents the specific utility of nest  $t_j$  and is shared by all of the alternatives in that nest. Most often, it will not be necessary to define utilities in this way and  $V_{t_j}^*$  then disappears from the equation, but if utility is to be added to nests then UTIL or U functions need to be defined as they are for alternatives – see Chapter 7.

The parameter  $\theta$  plays a key role in the model. First, it can be seen (after a little algebra!) that if  $\theta = 1$  then equation (6) is just the same as equation (1). That is, the possibility that  $\theta \neq 1$  is exactly what allows the nesting to give a more sophisticated model than MNL. Relating the model to the theory of random utility maximisation<sup>7</sup> shows that the model represents correlation between the random terms of the alternatives, but it is fully consistent with utility maximisation only if  $0 < \theta \leq 1$ , so a more sophisticated and valid model is obtained when we can estimate values of  $\theta$  that are positive and less than 1. ALOGIT can estimate values outside the valid range, so the estimates always need to be checked to ensure the model is valid. Estimated values outside the range can give insight into necessary changes in the specification of the model.

Size variables can be included in nested logit models, either by specifying a size function for elementary alternatives or by specifying a size function for the nests. When a nest has a size function, it is also necessary to specify a utility function for the nest, but if there are no utility components to be defined this can simply be zero. Care may be necessary in choosing the level at which size variables are attached in a nested model, i.e. in choosing between

$$p_j = \frac{\exp V_{t_j}}{\sum_{t_k=r} \exp V_k} \cdot \frac{S_j \exp V_j}{\sum_{t_k=t_j} S_k \exp V_k} \quad \text{and} \quad p_j = \frac{S_{t_j} \exp V_{n_j}}{\sum_{t_k=r} S_k \exp V_k} \cdot \frac{\exp V_j}{\sum_{t_k=t_j} \exp V_k} \quad (8)$$

In the first case the proportionality with respect to the size variables applies at the lower level of the nesting and in the second case it applies at the higher level. The user has to consider which of these is more appropriate and code the model accordingly. Size variables cannot be attached at multiple levels.

The model defined by equation (6) can be extended to multiple levels by defining nests that contain other nests. Any number of levels can be defined, subject to the constraints that an alternative or nest can belong to at most one nest and that ‘circularity’ (where nests ultimately

---

<sup>7</sup> One of several papers explaining this, produced around the same time, is Daly, A.J. and S. Zachary (1978) Improved Multiple Choice Models, in D.A. Hensher and M.Q. Dalvi (eds.), *Determinants of Travel Choice*, Saxon House, also available as a 1976 conference paper at [http://alogit.com/papers/Daly\\_Zachary.pdf](http://alogit.com/papers/Daly_Zachary.pdf).

contain themselves) must be avoided. A detailed discussion of the possibilities is given in another paper.<sup>8</sup>

It must be noted that the definition using equations (6) and (7) is only one way of specifying a tree-nested logit model and that some literature uses different specifications. Sometimes the ALOGIT specification is called RU1 (Random Utility 1) and an alternative form is called RU2. These different forms can be made entirely equivalent in practice, although RU1 may need to be adjusted by the use of ‘dummy nests’ as described in Appendix C. In other specifications, there will be an equivalent constraint to  $0 < \theta \leq 1$ , but it may take a different form, e.g. requiring parameters to increase or decrease between the successive levels of the tree.

The nests (i.e. the tree structure) and structural parameters  $\theta$  are defined using \$NEST commands, as described in section 4.9 below.

### 1.2.3 Error components (mixed logit)

The third major extension of MNL offered by ALOGIT is ‘mixed logit’ modelling. The mixed logit model has become very popular in recent years because it offers the ability to approximate accurately any random utility model.<sup>9</sup> The key to mixed logit modelling is the introduction of explicit random elements into the utility, called error components in ALOGIT. These can be used to model:

- variations of preference in the population, i.e. that the coefficients  $\beta$  in equation (2) are not the same for everyone;
- correlation between the utilities of the alternatives, in ways that are more complicated than is possible in a tree-nested logit structure;
- the possibility that some utilities are better defined than others (heteroskedasticity).

In ALOGIT, mixed logit is implemented by the use of error components.<sup>10</sup> These operate by introducing random components<sup>11</sup> into the utility functions. The model is then estimated by simulation, repeatedly making draws for each of the random variables and calculating utility for each set of draws, using equation (1) and an extension of equation (2)

$$V_{jd} = \sum_{k=1..K} \beta_k x_{kj} + \sum_{r=1..R} \gamma_r \xi_{rdj} \tag{9}$$

where  $\xi_{rdj}$  is the value of the  $r$ th random variable for alternative  $j$  at draw  $d$  and  $\gamma_r$  is the coefficient of the  $r$ th random variable.

The calculation of  $V_{jd}$  and the associated probabilities is repeated for each of a large number of draws  $d$ .

---

<sup>8</sup> Daly, A. J. (1987) Estimating ‘Tree’ Logit Models, *Transportation Research*, **21B**, pp 251- 267.

<sup>9</sup> This theoretical point is established by Dalal, S.R. and Klein, R.W. (1988) A flexible class of discrete choice models. *Marketing Science*, 7 (3), pp232-251; also by McFadden, D. and K. Train (2000) Mixed MNL models of discrete response. *Journal of Applied Econometrics*, 15, pp447-470.

<sup>10</sup> The error components approach can reproduce almost all of the models described as mixed logit in the literature. However, ALOGIT cannot estimate latent class models, which could be considered to be mixed logit models.

<sup>11</sup> Obtaining ‘truly’ random numbers is a difficult task, philosophically as well as mechanically. In practice, we work either with pseudorandom numbers, which ‘look like’ true random numbers, or with quasirandom numbers (such as Halton numbers) which are carefully chosen to perform well in this estimation context, but which are not random at all. ALOGIT provides both pseudorandom functions such as `NORMAL()` and `UNIFORM()`, and the quasirandom function `HALTON()`; all of these are described in section 6.2.3.

Note that the other extensions such as estimating size variables and tree structures are not possible with error components. However, size variables can be used if all of the relative coefficients are constrained.

Models defined in this way are called ‘mixed logit’ models. The term arises because, if we imagine the  $\xi$  variables to be fixed, the models would simply be MNL. But  $\xi$  takes a distribution, with an MNL for each value of  $\xi$ , so that the final model is a ‘mixture’ of the MNL models arising for each  $\xi$ .

The error components specification used in ALOGIT can be used to implement random parameters, heteroskedasticity and correlation between the utilities of the alternatives.

- To implement random parameters, the coding required is to set  $V_{jd} = \dots + \beta_k x_{kj} + \dots + \gamma_r \xi_{rdj}$ , with the error component  $\xi_{rdj} = x_{kj} \zeta_{rdj}$ . Then if  $\zeta_{rdj}$  is a random variable with mean 0 and standard deviation 1,  $\beta_k$  gives the mean and  $\gamma_r$  gives the standard deviation of the random parameter.
- To implement heteroskedasticity, the coding required is simply to set  $V_{jd} = \dots + \gamma_j \xi_{rdj}$ , with the error component  $\xi_{rdj}$  being itself a random variable with mean 0 and standard deviation 1; then the parameter  $\gamma_j$  gives the standard deviation of the utility of alternative  $j$  additional to the Gumbel variance  $\pi^2/6$ .
- To implement correlation between alternatives, say between alternatives 1 and 2, the coding is

$$\begin{aligned} V_{1d} &= \dots + \gamma_1 \xi_{1d} \\ V_{2d} &= \dots + \gamma_1 \xi_{1d} \\ V_{3d} &= \dots + \gamma_1 \xi_{2d} \end{aligned}$$

here, the  $\xi$ s are random variables with mean 0 and standard deviation 1; then the parameter  $\gamma_1$  gives the covariance between alternatives 1 and 2; note the way in which homoskedasticity across the alternatives can be maintained by including variation on alternative 3 (and on any other alternatives) that is independent but has the same variance.

Other variations and interactions can also be handled within this coding framework.

A further possibility is to set up some or all of the random variables in such a way that their total contribution is positive. This can be done by using exponentiation, extending equation (9) and introducing new ‘shape’ and scale parameters  $\alpha, \delta$  to be estimated

$$V_{jd} = \sum_{k=1..K} \beta_k x_{kj} + \sum_{r=1..R} \gamma_r \xi_{rdj} + x_{K+1,j} \delta_1 \exp(1 + \sum_{r=1..T} \alpha_r \xi_{rdj}) + x_{K+2,j} \delta_2 \exp(1 + \sum_{s=1..S} \alpha_s \xi_{sdj}) \dots \quad (10)$$

The effect of defining a model as in equation (10) is that the random components multiplied by  $\delta_1$  and  $\delta_2$  follow distributions that are constrained to be positive. Several such distributions can be included in the model, multiplying different data items. By including multiple components within each exponential, flexible patterns of covariance can be represented. The data items  $x_{K+1,j}$  etc. that the exponentials multiply can be negative, if it is required that a negative value should appear in the utility, e.g. for the cost of an alternative.

To estimate an error components model in ALOGIT, the user must take two steps.

First, code must be provided to calculate the random numbers  $\xi$  and store them in the `ERROR_COMP` array. ALOGIT automatically sets up a loop to make the calculations repeatedly, as required by the simulation, so the user has to program these calculations only

once. The number of repetitions is given by parameter NDRAWS on the \$ALGOR command (see section 4.1).

Second, the ERROR\_COMP array must be used in the utility functions. Unlike other terms in the utility function, ERROR\_COMP must appear multiplied by a coefficient *only*, and the coefficient must precede the random number, as in the examples above. So if any scaling is required, for example to represent heteroskedasticity, this must be done in the initial calculation of ERROR\_COMP. More details on the use of ERROR\_COMP in the utility function are given in section 5.3 below.

#### 1.2.4 Limitations of ALOGIT specifications

The most important limitations of the current version of ALOGIT are the following:

- it is not possible to estimate cross-nested logit models, i.e. those in which an alternative can belong to more than one nest;
- it is not possible to estimate latent class models;
- it is not possible to take account of correlations between observations, e.g. when there are multiple observations from a single individual.

It may be possible to include these capabilities in future versions of the software.

Additionally, it may be noted that aggregate data (i.e. using PROP ( . . )), ranked data (i.e. using RANK ( . . )) or sufficient statistics (i.e. using SUFF . STATS), all of which are described in section 5.2, can only be used in MNL models. The reason for this limitation is the estimation procedure using sufficient statistics, which makes it possible to use these facilities, is only possible with MNL models.<sup>12</sup>

#### 1.2.5 Estimation procedure

The estimation procedure in ALOGIT maximises the likelihood function. That is, it finds the values of the coefficients, such as  $\beta$  in the equations above, that maximise the likelihood of observing the data. This statistical criterion is a measure of the correspondence between theory and data, and is used throughout statistics as a basis for estimating unknown parameters. Statistical theory assures us that maximum likelihood is a sensible and robust procedure, in some ways better than *any* other method for parameter estimation.

The likelihood contribution of each observation in the data is given simply by the probability predicted for the chosen alternative, i.e. as indicated in equation (1) when  $j$  represents the observed choice. In maximising the likelihood, it turns out to be simpler to work with the log of likelihood, which is a negative number, because the probabilities are less than 1, so in maximising it, we are trying to get closer to zero. The overall likelihood is the product of the individual contributions, so the overall log likelihood is the sum of the individual log contributions.

The core of the estimation is therefore a mathematical procedure for maximising the log likelihood function. In contrast to several other programs, ALOGIT works with analytical first and second derivatives of that function. These exact calculations mean that ALOGIT iterations are typically considerably more effective than those of other programs, although of course some time is spent in making the exact calculations.

---

<sup>12</sup> Also, the calculation of elasticities, using the obsolescent command \$ELAST, described in Appendix B, is also possible only with an MNL model.

The exact second derivative matrix is used in making the iterations and is also available for use in calculating the estimation error. In ALOGIT 4.5, in distinction to earlier versions of ALOGIT, the estimation errors that are directly derived from the inverse of the second derivative matrix, the so-called ‘classical’ errors, are complemented by ‘robust’ errors, using the ‘sandwich’ matrix. These procedures are explained in more detail in Section 8.3.3.

The estimation of error components models in ALOGIT, as in most professional and academic work on mixed logit models, is by maximum simulated likelihood. That is, the distribution of the random components is modelled by simulation, repeatedly drawing values, and for each simulated value a logit model is applied. The average of the simulated values then gives the best estimate of the likelihood function. It is therefore possible to work with any distributions that can be simulated. ALOGIT provides facilities for simulating normal, logistic and uniform distributions and other distributions such as triangular can be derived simply from those.

MNL and tree logit models do not require simulation.

In either case, the procedure that is used for given values of the coefficients is, for each observation, to calculate its contribution to the log likelihood and first and second derivatives, applying any weight that is specified. The overall log likelihood and derivatives are calculated by summing over the observations. From the derivatives, a ‘step’ is calculated to find new values of the coefficients and the procedure is repeated. Under normal circumstances, the step that is calculated is that of the Newton-Raphson method; however, there are several possibilities for complications to arise; these are explained in a paper<sup>13</sup>, which also gives information about the ‘BHHH’ matrix, which is used in dealing with the complications and also in calculating the sandwich matrix to give robust errors.

When the model is MNL, advantage is taken of the sufficient statistics for that model to speed the estimation. For MNL models, using sufficient statistics also allows the estimation to use aggregate data or directly-input sufficient statistics, using `PROP( . . )` or `SUFF.STATS` respectively, both of which are described in section 5.2.2.

### **1.3 Development, ownership and origin of ALOGIT**

The current version of ALOGIT is 4.5, launched in beta form in January 2019 and in a definitive form in March 2022. Relative to the previous versions 4.3 and 4.3EC, version 4.5 offers an improvement, sometimes very large, in the speed of processing of the control file. The EC capability for mixed logit models is now integrated in the standard version of ALOGIT. Version 4.3 was current from 2006 to 2018 and earlier 4<sup>th</sup>-generation versions were current from 2000; further details are given in Appendix B.

ALOGIT has been developed, owned and supported by ALOGIT Software & Analysis Ltd. since 2007 and was previously owned by Hague Consulting Group bv (HCG). ALOGIT was originally developed by HCG for its own work and was offered for sale, initially in a mainframe version but later for PCs, from 1984 onwards. The development of the program has benefited from extensive professional use over more than 40 years.

---

<sup>13</sup> Daly, A.J. (1982) Estimating Choice Models Containing Attraction Variables, *Transportation Research Vol. 16B*, 1; this paper references Berndt, E.U., Hall, B.H., Hall, R.E. and Hausman, J.A. (1974) Estimation and inference in nonlinear structural models, *Annals of Economic and Social Measurement* 3-4, pp. 653-665. These authors are often referred to as BHHH.

In 2019-20 a project was undertaken to improve the FORTRAN standardisation of ALOGIT, which achieved standardisation for 98-99% of the code and focussed the non-standard elements into a separate source file. Transfer of ALOGIT to a new compiler should be straightforward.

Further information is available on the website <http://www.alogit.com/>.

## **1.4 Purpose and organisation of the User Documentation**

The purpose of the User Documentation is to give ALOGIT users sufficient information to make full use of the features of the program.

The operation of ALOGIT is determined by the control file, which the user will normally prepare to have the extension ‘.ALO’. The control file is summarised in the following chapter and details of its various components are described in Chapters 3-7. Reports of the run are given in the LOG file, details of which are given in Chapter 8. Chapter 9 describes the ‘initialisation file’, which gives some of the information necessary for setting up an ALOGIT run.

System files F11, F12 and F14 may be produced to support an ALOGIT run. Details of the system files are given in Appendix A. The LOG file and all of the system files will have the same name (i.e. before the file extension) as the ALO file, allowing files to be organised conveniently.

Appendix B gives information about previous versions of ALOGIT and obsolescent features that were available in those versions of ALOGIT.

Appendix C gives information about the normalisation of tree-nested logit models using ALOGIT. This appendix explains how the RU1 normalisation used in ALOGIT can be made equivalent to the RU2 normalisation used in many academic papers.

Appendix D gives the size limitations that apply to ALOGIT. These do not generally affect the size of the problem that can be analysed, but relate to the length of input lines etc.

Finally, Appendix E lists file handling errors that may be generated by the operating system.

## 2. The ALO File

The control file for ALOGIT contains the information to run the ALOGIT models, that is the necessary commands to read in data files, carry out data transformations, specify the logit model structure and ESTIMATE or APPLY the models. Usually the control file will have the extension ALO and for that reason this documentation refers to it as the ALO file.

This chapter summarises the components of the ALO file and gives the specification for the lines and characters of which the file is composed.

### 2.1 Components of the ALO file

The control file may contain a number of components

- `incl.file` commands, pointing to files containing further control commands
- comment and blank lines, not processed by ALOGIT but useful for documentation of the run and legibility of the file
- `$` commands, processed initially, when the control file is read<sup>14</sup>
- file specifiers and data transformations, processed for every observation
- utility and size functions, processed initially to set up structures *and* for every observation

Not every ALO file will contain all of these elements, but at least one file specifier *must* be present in every ALOGIT run.

Utility functions must come before size functions and these lines must be at the end of the file, except as described in the following paragraph. Otherwise, lines may be mixed in any order, but the rule in ALOGIT is that data items must be defined before they are used. Generally it is good practice to put the `$` commands early in the file and `$TITLE` and `$$SUBTITLE` first.

In an `APPLY` run transformations placed after the utility/size commands are carried out `AFTER` the logit model calculations. In `PREPARE` or `PREPARE & ESTIMATE` runs, transformations after the utility/size functions are not allowed. Otherwise, transformations placed before or among the utility/size functions are carried out before the logit model calculations are made.

No transformations, file specifiers or utility or size functions are allowed in `ESTIMATE` runs.

#### 2.1.1 *INCL.FILE Control*

The `INCL.FILE` control may be used to add control lines from another file to the main ALO file. Often, using these control commands clarifies the structure of the control file, particularly if running similar models where only parts of the control file differ between segments.

This control has two forms; the first is simply:

```
INCL.FILE file_name
```

---

<sup>14</sup> The command `$COEFF` replaces the coefficient lines used in earlier versions of ALOGIT. However, for compatibility with older files, these lines are still accepted and processed at the initial stage – that is, they function like `$COEFF` commands. See Appendix B for the specification of these and other obsolescent features.

In this case, `file_name` is a file containing further control lines, which will be processed at the point where the `INCL.FILE` command is included in the main control file. An `INCL.FILE` may contain any of the control commands, transformations etc.; the only command that an `INCL.FILE` cannot contain is another `INCL.FILE` command. Note that continuations of control lines or transformations cannot be made *between* the main file and the include file.

The second form is:

```
INCL.FILE COEFFS file_name
```

In this case the file indicated by `file_name` is expected to contain lines that are coefficient specifiers but may also contain other lines not acceptable as control lines. This form of `INCL.FILE` is intended for the input of F12 files (which give the results of model estimations, see Appendix A), from which the coefficient specifiers will be extracted and the other lines will be ignored.

### 2.1.2 *Comment and blank lines*

Comment lines, indicated by a minus sign ‘-’ in the first column, can be inserted anywhere in the ALO file. These lines have no impact on the processing but are very useful in documenting the ALO file.

Similarly, blank lines can be inserted anywhere in the ALO file to improve legibility without affecting the processing.

### 2.1.3 *\$ Commands*

Many of the most important instructions to ALOGIT are presented in the ALO file in the form of ‘\$ commands’. These are commands introduced by a keyword that begins with \$, such as \$TITLE or \$ESTIMATE. \$ commands are processed one time and are the first to be processed in an ALOGIT run, while most other commands are processed for each observation. Details of the \$ commands are given in Chapter 4.

### 2.1.4 *File specifiers*

At least one file specifier must be present in an ALOGIT run. These specifiers point to the data that ALOGIT is required to process.

Multiple data files may be input to ALOGIT. This feature allows the user to store data efficiently and merge the information during an ALOGIT run. This facility can save a lot of storage space and run time, while helping to keep an overview of the model structure. The operation of file specifiers and the various ways in which information can be linked are described in Chapter 3.

Files may also be output from ALOGIT, allowing the user to check intermediate calculations or to pass results from ALOGIT runs to other software. Output files are also described in Chapter 3.

### 2.1.5 Data transformations

One of the most powerful features of ALOGIT is its ability to transform data. ALOGIT can convert, combine, or create new data items from the base data input. The general form of a data transformation is that a specifically-named data item is set equal to a mathematical expression. Transformations may use system data items, operators or logical or mathematical functions.

ALOGIT performs the series of data transformations programmed by the user for each observation. These are organised as follows.

- In ESTIMATE runs, no transformations are allowed.
- In data processing runs, the transformations, \$ commands and file specifiers are the only components of the ALO file.
- In PREPARE and PREPARE-ESTIMATE runs, the transformations, \$ commands and file specifiers are placed before the utility and size functions.
- In APPLY runs, the transformations, \$ commands and file specifiers necessary to apply the model are placed before the utility and size functions. The utility and size functions may be followed by further transformations that use the results of the model application: in particular the DEMAND array (see 4.2) and the LOGSUM data item (see 5.2.2) are then available.

The ALO language is procedural, like FORTRAN, Pascal, Basic or C. It has only one data type (double precision) and all variables are initialised to 0 for each observation (unless this is explicitly prevented by \$KEEP). Compared with other procedural languages, it has many logical functions but few control structures.

Further details of ALOGIT's data transformations are given in Chapter 5.

### 2.1.6 Utility and Size Functions

The specification of the model to be estimated or applied in ALOGIT is largely done in the utility functions (and, where necessary, the size functions). In general terms, utility functions describe the quality of an alternative, the benefit a choice maker can get from choosing that alternative. Size functions are required only in some models, typically those involving spatial choice (e.g. destination choice) and describe the number of possibilities that are represented by an alternative (e.g. the size of a zone, measured in some way such as the number of jobs it contains).

The functions are specified by lines in the ALO file starting with

```

    U (nnn) =
or
    UTIL (nnn) =

and

    S (nnn) =
or
    SIZE (nnn) =

```

In these lines nnn is the label of the alternative whose utility or size function is being defined. The short form (U or S) can be used freely as an abbreviation of the longer form (UTIL or SIZE).

A size function can be specified only if a utility function for the same alternative has been specified previously.

An obsolescent form of these functions (Uxxx, UTILxxx, Sxxx and SIZExxx) is not recommended but is still recognised. Further information is given in Appendix B.

More details of the specification of models using utility and size functions are given in Chapter 7.

## 2.2 Characters and Lines

### 2.2.1 Characters

The set of characters recognised by ALOGIT in the ALO file is as follows:

- letters: A-Z and a-z (upper and lower case are considered indistinguishable<sup>15</sup>);
- numbers: 0-9
- tab characters, which are changed to blanks for processing
- operators: + - \* / ^ & | ~ # ! how these work is described in detail below
- 5 other separators: blank , ( ) = how these work is described in detail below
- 18 other ('printing') accepted characters: \$ < > % [ ] : @ { } ; " ' ` ? . \_ \ these function like letters
- other ('non-printing') characters<sup>16</sup>, which are changed to blanks, with a WARNING message to the user.

The processing of characters follows effectively the following steps to simplify the input for processing.

1. Continuations of a line are added to form longer lines, with a blank inserted to separate each continuation line (as described in the following section).
2. Non-printing characters are converted to blanks, with warning messages (except for tab characters).
3. Blanks next to (before or after) any other separator or operator are removed.
4. Any remaining blank or sequence of blanks is converted to a single comma.

These specifications give the user freedom to prepare the ALO file with maximum clarity, while giving precise instructions to the software.

### 2.2.2 Lines

Lines are read in by ALOGIT from the ALO file. Each line is processed separately, except that a line may be continued onto two or more further lines, under the following conditions.

The continuation of a \$command line is identified by a plus sign '+' as the first non-blank character on the line. If a plus sign is found, following a \$command line, then what follows is added to the \$command line before the whole line is processed. There may be several continuations, subject to the overall limits given below.

<sup>15</sup> The first mention of the name defines the capitalisation of a data item that is used in output; subsequent mentions are treated as identical irrespective of capitalisation.

<sup>16</sup> The full set of accepted characters is: tab (ASCII code 9) and all those with ASCII codes between 32 and 126 inclusive, sometimes described as 'printing characters'. 'Other characters' here refers to codes outside this range.

When the previous line was *not* a \$command line, then the next line is taken as a new line, i.e. it is **not** a continuation, if it is

- part of a logical structure (DO, THEN, ELSE or END),
- a \$command line,
- an INCL .FILE command,
- a FILE command or
- if it contains an equals sign '='.

**Otherwise** what follows is added to the previous line, before the whole line is processed. Again, there may be several continuations, subject to the overall limits below.

However, a line will always be treated as a continuation if more left brackets '(' have been opened than right brackets ')' closed.

Lines in ALOGIT control files may contain between 0 and 200 characters. A maximum of 20,000 characters and 5,000 strings are possible on a continued line.<sup>17</sup> There is no limit to the number of continuation lines that are processed together.

---

<sup>17</sup> These limits are considerably extended relative to earlier versions of ALOGIT.

## **3. File Commands**

ALOGIT can deal efficiently and conveniently with data in a wide range of formats and organisations. The key to addressing data is provided by the file commands, which are explained in this chapter.

This chapter first explains the concept of the ‘main’ data file, which defines the processing to be done by ALOGIT and to which any other files are related. Then a summary of FILE commands is given and finally detailed documentation of the subcommands is presented.

### **3.1 The main data file**

ALOGIT works by processing one data record at a time. This is a key feature of the program and enables it to deal with very large problems as it does not attempt to hold all the data in computer memory at one time.

The records to be processed are defined by the main data file. In simple data structures, there is often only one data file. However, with more complicated data ALOGIT takes the first file defined in the ALO file to be the main data file and files defined later in the ALO file are linked to the main file in various ways:

- files may contain further records to be processed after those of the first file; these files are indicated by the FOLLOWS subcommand and these files take over the role of the main file once the first file has been processed completely; when there are multiple FOLLOWS files they are processed in the order they appear in the ALO file;
- files may be linked to the main file by keys, possibly identified by the KEY subcommand as described in 3.3.5 below;
- files may contain data which relates to all the records to be analysed – this is called common data and is indicated by having neither FOLLOWS nor KEYS and not being indicated as OUTPUT;
- output files are defined by the OUTPUT subcommand – the first file in the ALO must not be an OUTPUT file but otherwise these files may be placed anywhere in the ALO and they do not have a special connection with the main file.

Thus, at any time during the processing, there is a main file from which each record defines an observation to be processed. Initially, the first file specified will be the main file; when all the records of this file have been processed, the first FOLLOWS file will be processed, then the second FOLLOWS file etc.. This process forms the ‘heartbeat’ of every ALOGIT run, since the commands in the ALO file are carried out once for each record.

A useful feature in some contexts, such as simulation, is to define ‘dummy’ files which cause ALOGIT to carry out the processing of the ALO file without reading in any data. For example, in simulation runs it might be useful to generate all of the data in the ALO file. Dummy files can be the only main file or can be included in a sequence of main files indicated by FOLLOWS. They are indicated by not having a file name or a list of variables to input, but must specify the number of ‘records’ to be processed using the NOBS subcommand as described in 3.3.8 below.

FILE commands control data input and output. They may be placed anywhere in the ALO file and are processed among the transformations in the order implied by their position in the ALO file. No more than 50 input and output files can be specified.

## 3.2 File command syntax

File commands take the form:

```
FILE (file_specification) variable_list
```

These components are described below.

### *File specification*

The file specification, i.e. the part of the file command contained in brackets, contains one or more subcommands and may also contain a format specification. These components have to be separated by commas (or spaces) and are described in detail in sections 3.3 (subcommands) and 3.4 (format).

### *Variable list*

The variable list is simply a list, separated by commas or spaces, of

- single data items and/or
- whole arrays

which are input or output in the order they appear on each record of the file.

If no variable list is given the file will be taken as a dummy file. In this case NOBS must be specified. Note that array elements (e.g.  $x(6)$ ) CANNOT be referenced, only entire arrays (e.g.  $x$ ).

Details of the variable list syntax are given in section 3.5.

## 3.3 File subcommands

The following 11 file subcommands may be used in ALOGIT<sup>18</sup>:

```
BINARY  
FOLLOWS  
HANDLE = label  
INTEGER  
KEY = KEY1 [, KEY2 [...] ]  
MATRIX = item  
NAME = yyy  
NOBS = xx  
OUTPUT  
SCENARIO [=label]  
TRANSPOSE
```

Some subcommands require further information as indicated in the list above; the others simply require the subcommand itself.

---

<sup>18</sup> The ERROR\_COMP subcommand is now obsolescent, see Appendix B.

The order of the subcommands is not important, and usually only a couple of subcommands are required. The subcommands are now described in turn, followed by a summary of the ways in which the subcommands can be used together.

### 3.3.1 *BINARY subcommand*

The `BINARY` subcommand causes the file to be read or written in binary form, either as 4-byte integers (if `INTEGER` is also specified) or as 8-byte (double precision) real values.<sup>19</sup>

It is not expected that this subcommand will be used extensively, but it could help in getting maximum accuracy and possibly a speed advantage. It could be useful for transferring information to other software. However, the disadvantage that the user cannot easily read the file will usually outweigh these possible advantages.

If `BINARY` is not specified (and `MATRIX` is not specified, see 3.3.6 below), the file will be formatted and able to be viewed in an ordinary text editor.

### 3.3.2 *FOLLOWS subcommand*

The `FOLLOWS` subcommand is used to identify a file that will become the main file when the initial main file (defined by the first `FILE` command in the `ALO` file) has been fully processed. `ALOGIT` thus processes all the observations in the first main file, then all the observations in the `FOLLOWS` file. In the same way, further `FOLLOWS` files can become the ‘main’ files until all have been read. `FOLLOWS` files therefore define the whole processing procedure.

The file structure may be different between different `FOLLOWS` files: this is very useful for processing multiple data files, possibly from different sources.

Note that variables not defined in the current main file are set to 0, unless they are specified as `$KEEP` (see 4.7).

### 3.3.3 *HANDLE subcommand*

The `HANDLE` subcommand gives a label for `ALOGIT` and the user to refer to a file, as indicated in `HANDLE = label`. This label is used by `ALOGIT` to refer to the file in messages in the `LOG` file and it can also be used by the user, e.g. in commands involving `FILE_OPEN ( . . )`.

If `HANDLE` is not given then a file number is allocated, which is the sequence of the file in the control file, and this is used in messages. It is good practice to use the `HANDLE` subcommand for all files. A `HANDLE` is of course clearer than a sequence number, but also the sequence number may change if the `ALO` file is edited.

The subcommand `HANDLE` is also recommended for dummy files.

### 3.3.4 *INTEGER subcommand*

The `INTEGER` subcommand indicates an integer file.

---

<sup>19</sup> `BINARY` specifies an `UNFORMATTED` file, specified to the operating system as `FORM='UNFORMATTED'`, not `FORM='BINARY'`. If the subcommand `BINARY` is not given, files are specified as `FORM='FORMATTED'`.

An INTEGER file cannot also have a format, so it must be free-format or BINARY. A free-format INTEGER file will give an error on input if a decimal point is encountered. On output, the values will be rounded to the nearest integer and output without decimal points.

### 3.3.5 *KEY subcommand*

A useful way of inputting additional data is by linking the information on different files. For example, data relating to the alternatives may be linked to data relating to the agent. Linking is done by matching KEY data items between the main file and one or more other files, using the `KEY = data_item[s]` subcommand or by using implicit KEYS.

Up to five data items may be listed after the `KEY=` subcommand. These data items must be defined and given values earlier in the ALO file, often by input from the current main file, and they must appear in the list of items to be read in from the keyed file. Data items can also be calculated from main file data items for matching with a linked file and that there can be multiple linked files, where items on the main file and/or on a first linked file (or calculated from items on those files) are used to link to a second linked file.

If no `KEY=` subcommand is specified, and there is no `FOLLOWS` subcommand, then any variables on the list that are already defined will be taken as key variables, with the order of the keys given by the order in which they appear on the variable list. If there are no variables in the list that are already defined, the file will be taken as giving COMMON data that applies to all the records processed by ALOGIT. Just a single record is read a from COMMON file.

For linking to work efficiently and reliably, the main file and the keyed files must be sorted with respect to the keys: the order of sorting is ascending and the first key variable is given highest priority, then the second, etc. If the files are not sorted, it is possible that linking will work, but the files will need to be rewound, possibly multiple times, increasing the run time, perhaps substantially.

Data items on keyed files and COMMON files that are changed by assignments will give rise to WARNING messages. A WARNING message will also be given if a file needs to be rewound.

### 3.3.6 *MATRIX subcommand*

The `MATRIX = item` command indicates a binary ‘matrix’ file and defines a key data item. Matrix files are intended for use in transport analysis contexts, where they contain data organised on an origin-destination basis. The key item represents the origin, so that ALOGIT reads the data for one origin at a time.

Several matrices can be stored on one file, as indicated by the names given in the variable list part of the file command. These names need to relate to arrays, all of them with at least the length required to store a row of the matrix. The matrices on the file are square and errors will be generated if the arrays are not defined or are too small; if the arrays are longer than the data rows on the file, only the first items of the arrays will be processed.

Matrix files refer to ‘MinUTP’<sup>20</sup> format.<sup>21</sup>

### 3.3.7 *NAME subcommand*

The subcommand `NAME = file_name` gives the name of the file to which the `FILE` command refers.

File names in ALOGIT must have names not exceeding 250 characters, File names may not contain ALOGIT separators (tab characters, ALOGIT operators or other separators, see 2.2.1) or ASCII characters outside the standard range (32-126 inclusive). Also, characters that are not allowed by the operating system for file names (?\* etc.) must not be used or problems will be caused. Of course, the character \ is permitted.

If `NAME =` is missing for an output file, the indicated outputs will go to the LOG file.

Note that the `NAME = sub-control` must be omitted for dummy files but `NAME =` is essential for all other files.

### 3.3.8 *NOBS subcommand*

The `NOBS = nnn` subcommand restricts the number of observations input or output to or from ALOGIT on the file to which the command refers. Input or output will be stopped after the indicated number of observations, after which the file will be closed as if the end of the file had been reached. If `NOBS` is not specified, ALOGIT will continue to read an input file until it reaches the end of the file or write an output file until the end of the run.

`NOBS` is the only way of setting the number of ‘observations’ on a dummy file.

If `NOBS` is 0, the file will be closed as soon as it is opened, without reading any data from it; this facility can be useful in testing data processing. `NOBS = 0` can also be used on a dummy file.

### 3.3.9 *OUTPUT subcommand*

By default, ALOGIT tries to input data from files. The `OUTPUT` subcommand causes the items in the data list to be written to the relevant file, using the format given in the file format specification. All of these data items must have been defined and given values previously in the ALO file.

`OUTPUT` cannot be applied to the first file in the ALO file.

### 3.3.10 *SCENARIO subcommand*

The `SCENARIO [= label]` sub-command indicates that the file is to be a SCN output file. If `= label` is present, the scenario file will be labelled as indicated; otherwise a label will be constructed from the name of the scenario file.

---

<sup>20</sup> These files are in the format of the MinUTP software, which is probably now defunct. However, the file format remains compact and quick to read and write, so the format remains useful.

<sup>21</sup> Previously, it was also possible to use VISUM format. However, these formats have been changed and ALOGIT is no longer able to process them.

The data items in the variable list should then be either ARRAYS of length ALTS, or scalars. The first output will always be DEMAND (see 4.2) and up to 3 additional variables can be specified. If the data items are arrays, the first dimension of the output is the index of the arrays; if they are scalars, the first dimension is fixed to 1 and dimensions 2-10 will be defined by the first 9 entries on the \$DIMENSION command.

Note that the \$TAB.VAR command is normally used to define scenario output (see section 4.12).

### 3.3.11 SKIP1 subcommand

This command can be given to cause ALOGIT to skip over the first line of formatted input files. Some software, such as Excel writing CSV files, writes an initial record on data files that contains header information. This information cannot be processed by ALOGIT but the use of the SKIP1 subcommand allows ALOGIT to access the remaining information.

### 3.3.12 TRANSPOSE subcommand

This subcommand is available only with OUTPUT MATRIX files. It has the effect that the rows and columns of all the matrix files are reversed. So, for zonal matrices of transport data, the origins and destinations are interchanged.

The MATRIX subcommand must precede the TRANSPOSE subcommand.

### 3.3.13 Interaction of FILE subcommands

Some FILE subcommands can be used with other subcommands, while other combinations are not allowed. The table below clarifies the interactions that are and are not permitted.

	INTEGER	NOBS	OUTPUT	FOLLOWS	MATRIX	HANDLE	KEY	NAME	TRANSPOSE
BINARY	yes	yes	yes	yes	no	yes	yes	yes	no
INTEGER		yes	yes	yes	no	yes	yes	yes	no
NOBS			yes	yes	no	yes	no	yes	no
OUTPUT				no	yes	yes	no	yes	yes
FOLLOWS					yes	yes	yes	yes	no
MATRIX						yes	no	yes	yes
HANDLE							yes	yes	yes
KEY								yes	no
NAME									yes

SCENARIO files are for output only. These files may not be INTEGER, BINARY, FOLLOWS, MATRIX or KEY, all of which follow from the nature of SCENARIO files.

## 3.4 File format specification

ALOGIT can deal with two different file format specifications:

- Free format, where no explicit file format is given; in this case:
  - input file records must have the data items separated by spaces, tab characters or commas<sup>22</sup>;
  - output file records are written with comma separation; or

---

<sup>22</sup> The CSV format for spreadsheets often works well to exchange information with ALOGIT free format.

- Explicit fixed format, where the user specifies the file structure in the file `formatstring` sub-command.

Note that these formats do not apply to files specified as BINARY or MATRIX files.

### 3.4.1 Free format

Free formats are easy to use, but the user must be sure that input files are set up with the variables properly separated and that output files with comma separation will be acceptable in any further processing that is to be done.

### 3.4.2 Explicit fixed format

Explicit formats may be defined for either input or output files. An explicit file format consists of a list of format components, which are used in sequence to input or output the items in the variable list. The different format elements are best illustrated by way of examples. Note that it is not possible to mix real numbers (with decimals) and integers (whole numbers) on one data file.

#### Examples 1: real numbers in output file with space between data items

```
file (name=example.dat, output, 10.0,2x,10.0,2x,3.0,2x,8.2)
sample_n home_stm person_n weight
```

The output format elements work as follows<sup>23</sup>:

1. the first element, `10.0`, is used for the first output data item, `sample_n`, and puts that data item into the first 10 characters of each record; the next two characters are spaces (`2x`);
2. this is followed by 10 characters for `home_stm`; followed again by two spaces,
3. then `person_n` in 3 characters and 2 more spaces; because of the specification `' .0'` for all these items, they are output with a decimal point but no decimals;
4. however, the final item, `weight`, is output in 8 characters, with a decimal point and two decimals as implied by the specification: `8.2`.

This specification will lead to a file that looks like

```
123456.      70956.   3.      456.78
789012.      63451.   1.      1097.32
etc.
```

This format could have been achieved slightly more economically using brackets:

```
file (name=example.dat, output, 2(10.0,2x),3.0,2x,8.2)
sample_n home_stm person_n weight
```

The effect of the `'2'` in front of the bracket is to get the format inside the bracket used twice before moving on to the remainder of the format. Brackets can also be nested (not more than 10 deep), but some experimentation will usually be necessary to get the desired results.

To read this file again in another ALOGIT run, a command is needed like

```
file (name=example.dat) sample_n home_stm person_n weight
```

<sup>23</sup> ALOGIT's format specifications are in the style of FORTRAN, but are more limited.

In this case no format specification is required because the input file is in default (space-separated) format. Space-separated is a common format for input files. Comma and tab separation work just as well; with space separation it is necessary to be certain that the fields are actually *separated* by one or more spaces, as they were in this case by the use of ‘x’ format to obtain 2 spaces between each data item. However, separation by semi-colon (;) is not possible; note that this format can be used by spreadsheets outputting so-called CSV files.

**Examples 2: more real number specifications**

Real data files can also be formatted using the G format specification, which allows formats to be repeated without the use of brackets. These file types can be specified using the nGw.d specification, where n is a repeat factor (omitted if 1), w is the width of the field, and d is the number of decimals.

For example, if we specify the format (g4.0, g2.0, 11g6.3) for output, we would get a file that looks like

```

1 3 .295 .609 .553 .297 .227 .945 .766 .536 .455 .770 .232
2 2 .453 .368 .738 .738 .315 .577 .670 .451 .496 .592 .536
...

```

Alternatively, the file can be specified by the field widths alone, e.g. format 4,2,11\*6 would give the same file format

```

1 3 .295 .609 .553 .297 .227 .945 .766 .536 .455 .770 .232
2 2 .453 .368 .738 .738 .315 .577 .670 .451 .496 .592 .536
...

```

**Example 3: Explicitly formatted integer data files:**

These file types can be specified using the nIw format string, where n is a repeat factor (omitted if 1) and w is the width of the field. Alternatively, the file can be specified using the field widths alone, as shown in the previous example. So we could use either

```
i4,i2,2i1,i2
```

or, providing the file is specified as INTEGER (see section 3.3.4),

```
4,2,2*1,2
```

to get

```

1020 11151
1140 130 0
.....

```

Note that this file could not of course be input to a different run of ALOGIT in default format, because the values are not separated by spaces.

The key distinction given by INTEGER files is that decimal points are not given on output, thus allowing the file to be smaller or clearer for reading.

**Example 4: Explicitly formatted file with non-numerical data:**

The / symbol can be used to skip onto the next line of the data file to skip over non-numerical

data, as demonstrated by the first example. The `nx` specifier can also be used to instruct ALOGIT to skip over `n` characters.

For example, the format `g4.0, g2.0/11g6.3` could be used to read a file like

```
1 3 London 1984
.295 .609 .553 .297 .227 .945 .766 .536 .455 .770 .232
2 2 Paris 1983-88
.453 .368 .738 .738 .315 .577 .670 .451 .496 .592 .536
...
```

In many cases it will be necessary to test formats until the required results are obtained.

### 3.5 Variable list syntax

Items in the variable list can be single data items, arrays or (for output only) constants.<sup>24</sup>

Single data items may have user-specified names or be system-defined items such as `CHOICE`, `WEIGHT`, etc..

Array items for input or output must relate to the whole array. Parts of arrays or single items in arrays are not permitted here. The array must be defined in a previous `$ARRAY` command by the user or be a system array (`RANK`, `PROP`, `AVAIL`, `EXCLUDE`, `FILE_OPEN` (output only) or `DEMAND` (output only))<sup>25</sup>. If the length of an array is changed after it is referenced in a file command, errors may occur and it is not recommended to do this.

When items input in this way are already known to ALOGIT (either by input in a previous file or by calculation by user commands), they will be treated as keys for the file. This implicit specification of keys may be used instead of or in addition to the explicit specification using the `KEY` subcommand, see 3.3.5 above, which also explains how the keys work.

---

<sup>24</sup> Ranges of data, which were permitted in earlier versions of ALOGIT, are no longer accepted in 4.5 (see Appendix B). Use arrays instead.

<sup>25</sup> Note that `ERROR_COMP` cannot be input or output in this way. See section 5.3 for more details

## 4. \$ Commands

ALOGIT has 15 \$ commands, which are used to define many of the most important features of each run. In this chapter, these commands are documented in full, in alphabetic order.<sup>26</sup>

So that ALOGIT can recognise these commands, they need to be the first characters on the line and to be followed by a space or comma.

### 4.1 \$ALGOR command

The \$ALGOR command provides some controls for the likelihood optimisation procedure.<sup>27</sup> The command operates through the specification of subcontrols:

\$ALGOR subcontrol= , subcontrol= , ... etc.

The subcontrols are as follows.

Subcontrol	Range	Default	Explanation
MAXIT	0 – 32000	20	Maximum number of iterations to be made
CON.CRIT	1 – 3	3	Defines the convergence criterion to be used: <ol style="list-style-type: none"> <li>1. <math>\max_k(\delta\beta_k/\beta_k)</math>, comparing the changes <math>\delta\beta</math> with the current (non-zero) coefficient values <math>\beta</math></li> <li>2. <math>\sqrt{\sum_k(\delta\beta_k/\beta_k)^2}</math>, again using only non-zero values <math>\beta</math></li> <li>3. <math>\sqrt{\sum_k(\delta\beta_k/\sigma_k)^2}</math>, comparing the changes <math>\delta\beta</math> with the current standard errors <math>\sigma</math>, but setting a large value if <math>\sigma</math> is zero</li> </ol>
CON.LEVEL	> 0	0.01	Convergence is reached when the selected criterion falls below this value
CON.BHHH	$\geq 0$	0	When CON.BHHH > CON.LEVEL, BHHH iterations <sup>28</sup> are used until this level of convergence is achieved, then standard iterations are used until CON.LEVEL is reached
ALPHA	$0 < \alpha \leq 1$	$10^{-6}$	Used to test whether step that is proposed is nearly perpendicular to the first derivative by testing whether $\cos(\phi) < \alpha$ where $\phi$ is the angle. If the test fails, ALPHA times the first derivative is added to the step.

<sup>26</sup> \$TITLE and \$SUBTITLE are documented together. \$ELAST is an obsolescent feature and is documented in Appendix B rather than in this Chapter.

<sup>27</sup> Note also the sub-commands LIN.FIRST and GOGO on the \$ESTIMATE command.

<sup>28</sup> That is, the BHHH matrix is used to calculate steps, rather than the true second derivative matrix. This can save a lot of time per iteration, though the iterations are less effective in getting closer to the optimum.

ZETA	no limits	0.15	When CUBIC is FALSE or CUBIC factoring does not work, the step size calculated in the algorithm is reduced by the factor ZETA. Note that a constant step reduction satisfies the Berndt et al. (1974) <sup>29</sup> requirement for convergence.
STEPS	1 – 32000	4	The maximum number of ZETA reduction steps to be made before admitting defeat. Note that $0.15^4 = 5 * 10^{-4}$ approximately.
EPSILON	$\geq 0$	$10^{-6}$	This value is used in matrix inversion to test whether the matrix is near singularity <sup>30</sup>
NDRAWS	$\geq 0$	0	Indicates the number of random draws to make in mixed logit runs
CUBIC	$>0$ or $\leq 0^+$	TRUE	When CUBIC is TRUE and the function value is declining, instead of factoring by ZETA, a step size $x$ is determined to solve the equation $0 = f' = s(1 - x - 3cx^2)$ , where $s$ is the slope at the current $\beta$ values and $c$ is calculated from the cubic equation $\Delta f = s(t - t^2/2 - ct^3)$ where $\Delta f$ is the observed change (decrease) in log likelihood and $t$ is the current step length. If this does not work, i.e. $\Delta f$ remains negative, ZETA factoring is resumed. CUBIC steps seem to work faster in many cases.

<sup>+</sup> Note that FALSE and TRUE are equivalent to 0 and 1 respectively.

## 4.2 \$ARRAY command

The \$ARRAY command is used to indicate that a single data item name, followed by a bracketed index, refers to a series of values: the ‘array elements’. The \$ARRAY command also defines the length of the series.

Any number of \$ARRAY controls may be included in the control file, each defining any number of arrays of any positive length. Array elements may not be used, nor may whole non-system arrays, before they have been defined in this way. However, after the array has been defined, any element (up to the length of the array) may be used and ALOGIT is unable to check whether that specific element has been assigned a value.

The special value ALTS may be used instead of a constant, in which case ALOGIT will define the array to have a length equal to the number of alternatives. In this case the array may be indexed by the alternative identifier (as indicated in section 7.3). Note that ALTS includes any composite alternatives which are defined.

Seven arrays are predefined by the system and do not require \$ARRAY commands:

<sup>29</sup> Berndt, E.U., Hall, B.H., Hall, R.E. and Hausman, J.A. (1974) Estimation and inference in nonlinear structural models, *Annals of Economic and Social Measurement* 3-4, pp. 653-665. These authors are often referred to as BHHH.

<sup>30</sup> This value is considerably smaller than in previous versions of ALOGIT and should reduce the number of unnecessary reports of inversion failure.

- EXCLUDE, which is set to TRUE to exclude an observation from the model, with a default value of FALSE; EXCLUDE is defined to have the length of the highest constant value that is set;
- AVAIL, which is set to FALSE to make an alternative unavailable, with a default value of TRUE (unlike all other ALOGIT default settings, which are zero, i.e. FALSE); AVAIL has length ALTS;
- FILE\_OPEN, which is set to TRUE when a file is open; FILE\_OPEN has a length equal to the number of files defined and can be indexed by the sequence number of the file in the ALO file or by the file HANDLE;
- ERROR\_COMP, which is used in setting up mixed logit models as described in section 7.1; this array has the length of the highest defined index;
- DEMAND, which gives the expected demand for an alternative, i.e. the choice probability times the WEIGHT, after application of the model in an APPLY run;
- PROP, which gives the proportion of an observation that chooses each alternative in an MNL model; this can be used where appropriate instead of CHOICE;
- RANK, which gives the ranking of choice for the alternatives in ranked data, replacing a single CHOICE specification; RANK can only be used in MNL models.

The last three arrays have length ALTS.

If an ARRAY is redefined, the later definition will have effect. However, redefinition in this way is not advisable.

### 4.3 \$COEFF command

This control is used to define the coefficients that will be estimated in an ESTIMATE run or that will be applied in an APPLY run.

Coefficients are defined simply by listing them on the \$COEFF command. Additionally, values may be given to the coefficients, which are used as initial values for estimation or as values to be used in APPLY runs. These values are defined in two ways:

- `coefficient = value`, which means that the coefficient is to be constrained to the value indicated, which may be zero;
- `coefficient # value`, which means that the value is to be used as the initial value for the coefficient, but it is not constrained.

Coefficients may also be defined by reading them from a file of previously determined values, usually an F12 file, using an INCL.FILE COEFFS command, as mentioned in section 2.1.1. A coefficient that was constrained on the run that produced the F12 file can be unconstrained by simply setting coefficient #, which will use the value on the F12 file as a starting value but not hold the coefficient to that value. Similarly, a coefficient that was unconstrained can be constrained by setting coefficient =.

If no explicit value is given in the \$COEFF command or in an input file, a default value will be used to initialise the coefficient. This value is 0 for ordinary, size variable and random component coefficients, and 1 for nest coefficients and the log-size-multiplier. Coefficients first specified in this way will not be constrained.

If a coefficient is specified by more than one \$COEFF or INCL.FILE COEFFS command, the last specification is used. This feature can be useful, for example in changing the constrained status of a coefficient on an F12 file.

## **4.4 \$DIMENSION command**

This command specifies the data items defining columns for APPLY tables<sup>31</sup>. The command simply lists data items: each data item listed will have a table whose columns are defined by the values of the data item. Each data item may take the values 1 to 20 as 20 is the maximum number of columns that can be included in a table. Data items over 20 or zero or negative values will cause the observation to be omitted from the relevant tables. Up to 100 data items may be given.

## **4.5 \$ESTIMATE command**

The \$ESTIMATE command controls the main operation of ALOGIT.

If \$ESTIMATE is present then (subject to the requirements of subcommands) PREPARE and ESTIMATE steps will be run. Otherwise, if UTIL functions are defined, then an APPLY run will be performed. If no \$ESTIMATE command line is given and no UTIL functions are defined, then a simple data processing run will be performed.

\$ESTIMATE has five subcommands, as follows.

- LATER        The LATER subcommand indicates that ALOGIT is to perform the PREPARE stage only and terminate before starting estimation. This subcommand can be used to check the setup of a model, for example.
- DONE         This subcommand indicates that model estimation has already been performed, i.e. that an APPLY run should be performed. Note that this explicit specification of an APPLY run (rather than implicitly, by omitting the \$ESTIMATE command) is necessary to call for a check comparing the likelihood value in APPLY with what was obtained in model estimation. However, to get this check it is necessary to specify at least one table using \$DIMENSION.
- MODEL        If \$ESTIMATE model=model\_name is specified, ALOGIT will read in the data from model\_name.F11. If an F12 file named model\_name.F12 exists the coefficient values in this file will be read in, and if an F14 file named model\_name.F14 exists this file will also be read in. Only \$NEST, \$COEFF and \$ALGOR dollar commands can be used – note therefore that it is possible to specify a new tree structure. F11 files cannot be read in to APPLY runs.
- LIN.FIRST    If LIN.FIRST=TRUE, a linear model will be estimated before proceeding to non-linear modelling. When this option is enabled, the optimal coefficients of the linear model will be written to a file called xx.L12, where xx is the base name of the ALO file. The default is to make the linear run first, but sometimes it is preferable to prevent linear optimisation and begin immediately with the full model, i.e. LIN.FIRST=FALSE. Note that LIN.FIRST=FALSE will be applied automatically if there is no equivalent linear model, for example if a nest alternative has a non-zero utility function.
- GOGO         Setting this subcommand causes the run to go ahead even if an apparent error is discovered. For example, estimation will be undertaken even if an error is

---

<sup>31</sup> These column specifications are also used in SCENARIO files, see 3.3.10.

discovered in the preparation stage, or an apply run will be made even if the estimation has not converged. The default is GOGO=FALSE, while the GOGO option can be initiated by setting GOGO=TRUE or simply by placing GOGO on the \$ESTIMATE control line.

## 4.6 \$GEN.STATS command

The \$GEN.STATS command causes basic statistics to be produced for a list of variables for all observations. It can also be used to produce statistics for the utilities, for available alternatives only, or for error components. The statistics are reported in the LOG file.

\$GEN.STATS causes the non-zero percentage, minimum, maximum, mean and standard deviation to be produced for each variable that is selected; mean and standard deviation are printed only for variables that are not constant. Additionally, correlations can be produced by the subcommand CORREL. The statistics are based on non-zero observations only.

The facilities of \$GEN.STATS are invoked by subcommands and/or a list of data items, which are presented on the \$GEN.STATS command line. The possible subcommands are as follows.

**UTILITIES** This subcommand causes all of the data items appearing in UTILITY and SIZE functions to be printed. This is often a very useful check on the data processing that is done to set up a model. Note that these statistics are produced for data items appearing for available alternatives only, which is not the case for any of the other options.

**ALL** This subcommand causes statistics for all of the named data items in the run to be printed.

**CORREL** This subcommand causes correlations to be produced between all the items in the list defined by UTILITIES, ALL or the explicit list of data items. This can result in a lot of output!

**ERROR\_COMP** This subcommand produces statistics on error components, reporting within observation and between observation separately. This command can be used in conjunction with CORREL, but correlations between error components and other variables are not given. Note that error component statistics are accumulated even if the relevant alternative is not available.

**list of data items** Statistics are produced for each data item listed. These can be any user-defined or system data item, including arrays, in which case statistics are produced for each element of the array.

## 4.7 \$KEEP command

In ALOGIT, all data items are initialised to 0 (i.e. the same as FALSE) by default for each observation. This includes user-defined and system-defined single data items and array elements. Exceptions are:

- the array AVAIL, which is initialised to TRUE (i.e. 1);
- COUNT, which is set to the current observation sequence number (and cannot be changed by the user);
- ID, which is initialised to COUNT;

- WEIGHT, which is initialised to 1;
- the array FILE\_OPEN, which indicates whether the file is open for input or output and cannot be changed by the user;
- the array DEMAND, which is available *after* the model processing and is set to the number of choices of each alternative;
- data items specified in \$KEEP.

The \$KEEP command is used to prevent data items from being initialised. It is possible for the user to change \$KEEP data items, but a warning is given in the LOG file. Both single data items and arrays can be specified in \$KEEP.

The \$KEEP command can also be used to set data items and arrays to given values, as illustrated by the following example:

```
$KEEP nprimes=5, isgood
$ARRAY primes(nprimes)
$KEEP primes = 2 3 5 7 11
```

The first \$KEEP command sets the value of nprimes, so that it can be used to set the length of an array. The second \$KEEP command gives the nprimes (i.e. 5) values to be stored in the array primes().

The first \$KEEP command also indicates that the data item isgood is not to be re-initialised for each observation. While this data item will have the initial value 0 for the first observation, this is not exactly the same as specifying isgood=0 on the \$KEEP command. When a value is given, the data item is considered to be defined, and can be used on the right of a data transformation; when no value is given, a single data item is considered undefined and it needs to be assigned a value, otherwise using it on the right of a transformation will cause an error; this may be useful as a check on the logic of the ALO file. Array elements are always considered defined, once a \$ARRAY command has been given.

## 4.8 \$L\_S\_M command

The \$L\_S\_M specification identifies the log-size multiplier for SIZE functions. This appears as the parameter  $\phi$  in the specification of utility including size:

$$V_j^* = V_j + \phi \cdot \log S_j = \sum_{k=1..K} \beta_k x_{kj} + \phi \cdot \log(s_{1j} + \sum_{l=2..L} \gamma_l x_{lj})$$

As explained in the Introduction,  $\phi$  will often be 1, but if a value other than 1 is to be used, the coefficient must be specified in a \$L\_S\_M command. If there is no \$L\_S\_M command and size variables are present, the log size multiplier will be constrained to 1.

The log-size multiplier can be specified as a number or a label. If a number is used, the coefficient will be given the name GammaXXX, where XXX is the number given. In either case, unless a \$COEFF specification is also given, the initial value will be 1 and the coefficient will be unconstrained.

Only one \$L\_S\_M command may be given in an ALOGIT run.

## 4.9 \$NEST command

The \$NEST command is used to define the tree structure. Each nest is given a name, a structural coefficient (in brackets), and then the components of each nest are specified, as shown in the example.

```
$NEST Nest1 (Theta_1) Alt1 Alt2 Alt3
```

This command indicates that the three alternatives Alt1, Alt2 and Alt3 are to be nested into the composite alternative Nest1 and that the coefficient Theta1 is to be used to multiply the utility contribution of these three alternatives to the utility of Nest1.

Referring to equation (7) in the Introduction (slightly simplified),

$$V_n = V_n^* + \theta \log \sum_{t_k=n} \exp V_k \quad (7a)$$

the example \$NEST command indicates that the three listed alternatives are to be used to form the logsum, i.e. the log of the sum of their exponentiated utilities, and that this logsum is to be multiplied by Theta\_1 and added to the utility of the nest, as illustrated in equation (7a);  $V_n^*$  is the utility (if any) explicitly attached to the nest alternative and specified in the utility function Util(Nest1).

By default, alternatives (elementary or composite) not appearing in a \$NEST command are connected to the root of the tree. However, it is possible to define:

```
$NEST ROOT () list ...
```

If this is done then only the alternatives in the list will be connected to the root. Other unconnected alternatives will be reported as errors. The brackets MUST be empty in the \$NEST command specifying the ROOT nest. Sometimes it is useful to define the alternatives connected to the root in this way so that their names can be used in other commands.

\$NEST commands may be input in any order. Consistency checks will be made.

The nest coefficient can be specified as a number or a label. If a number is used, the coefficient will be given the name ThetaXXX, where XXX is the number given. In either case, unless a \$COEFF specification is also given, the initial value will be 1 and the coefficient will be unconstrained.

## 4.10 \$PRINT command

The \$PRINT command controls many of the printing options of ALOGIT.

As usual with \$ commands, the command is formatted as follows. Note that all of these subcontrols must be set equal to an explicit value.

```
$PRINT subcommand = value [subcommand = value ...]
```

The subcommands for \$PRINT are as follows.

Subcommand	Range	Default	Explanation
MAT.STAT	0 – 2	1	0 no 'matrix statistics' print will be produced 1 alternative availability and ranges of independent variables will be output as 1, plus mean, s.d. and correlation of independent variables 2 Output is made for available alternatives only.
ITER	-1 – 6	1	0 no output is produced from iterations before the estimation has converged 1 the likelihood and convergence value are output at each iteration (in the log file as in the DOS window) 2 important changes in coefficient values are reported at each iteration 3 coefficient values, standard errors and t values, plus correlations, are produced at each iteration (as at convergence) 4 as 3, plus the first derivatives are output at each iteration 5 as 3, additionally giving details of the algorithm operation <i>values 6 and -1 are intended for special purposes only</i> 6 as 5, additionally giving first derivatives, the second derivative matrix and its inverse -1 the first derivatives are produced at convergence (when they are zero, in principle)
PROB	0 – 1	0	The test value for output of outliers in APPLY. Observations with a probability for the chosen alternative less than PROB are output; so the value 0 produces no output.
REJ.OBS	-1 – 99999	25	The number of rejected observations for which output is made. Observations are rejected when the chosen alternative is invalid, there are no unchosen alternatives, all the size variables are zero, etc.. -1 all rejected observations are output
TRANSFORMS	>0 or ≤0 <sup>+</sup>	FALSE	Controls whether to print data transformations (which can be lengthy). Positive or TRUE to call for output; FALSE, 0 or negative to suppress output.
ERR.FLAG	>0 or ≤0 <sup>+</sup>	FALSE	Controls whether the rejection of an observation will be treated as an error (run terminates) or that the run continues without that observation. Positive or TRUE to call for termination; FALSE, 0 or negative to continue.
CH.RANGE	0 – 1	0.1	Gives an error range for the total choice probability indicated by PROP (when this is used). Observations with total probability more than CH.RANGE away from 1 are rejected, otherwise the PROP values are scaled so the total is 1.

OBS.RANGE	-1 – 99999	3	Indicates the maximum number of observations per iteration that are output when the utility values are out of range during estimation. -1 all out-of-range observations are output
TREE	>0 or ≤0 <sup>+</sup>	FALSE	Controls whether to print the tree (which can be lengthy). Positive or TRUE to call for output; FALSE, 0 or negative to suppress output.
CHOICE	>0 or ≤0 <sup>+</sup>	FALSE	Controls whether to print alternative availability for all alternatives (which can be lengthy) or just those that are chosen. Positive or TRUE to restrict output; FALSE, 0 or negative to get full output. Not relevant if MAT.STAT is zero.
EST.ERROR	1, 2 or 3	1	Controls final printing of robust and classical errors in estimation <sup>32</sup> : 1 robust errors only 2 robust and classical errors 3 classical errors only

<sup>+</sup> Note that FALSE and TRUE are equivalent to 0 and 1 respectively.

#### 4.11 \$ROW command

The \$ROW command is used to specify rows for tables and aggregations for scenario files<sup>33</sup>. The default is to put each alternative in its own row, labelled by its name. There are three forms of this command.

1. A fixed row for each alternative can be given:

```
$ROW (row-id) list
```

row-id must be a new identifier and will appear as the label of the row  
list is a list of elementary alternatives that are aggregated to form the row

2. The row for each alternative can be indicated by a variable.

```
$ROW alt-id = data_item
```

The alternative identified will be put in the row indicated by the data item. Rows of this form will be identified by the value of the data item.

3. The rows for all of the alternatives can be specified by an array of dimension alts, as illustrated below. The rows will then be labelled by the value of the array.

<sup>32</sup> Printing at each iteration is of classical errors only as these relate to both the optimisation process and the final estimation error, whereas robust errors relate only to the estimation error. Please note that the F12 file will give robust errors unless EST.ERROR=3. For a discussion of the relationship of classical and robust errors and of other approaches to error estimation, see Daly, A. and Hess, S. (2016) Simple approaches for random utility modelling with panel data, *to be made available online*.

<sup>33</sup> These aggregations are also used for the obsolescent \$ELAST calculations; see B2.3.

The uses of these forms of the command are illustrated by the following examples. For models with large numbers of alternatives, the third form can save many lines of code in the control file.

Example 1:

```
$row (active) walk cycle
```

`walk` and `cycle` are aggregated to form the ‘active’ mode and presented as one row in the tables.

Example 2:

```
$row dest01 = LClass01
```

The alternative `dest01` will be put in the row number defined by the data item `LClass01`. Other alternatives can have their rows indicated in a similar way, possibly using the same variable to aggregate alternatives into a smaller number of rows.

Example 3:

```
$array dist(alts) tlclass(alts)
$row alts = tlclass

do i = alts
  tlclass(i) = ifge(dist(i), 0, 10, 20, 50, 100)
end
```

Each alternative is put in a row (number 1 to 5) according to the distance for that alternative, defined by the array `dist(i)`, which has to be set using other data.

If there is no `$row` command, but tables are requested by the use of `$dimension`, each alternative will be put in its own row, labelled by its name. If partial row definitions are given, user-defined rows will appear first in the tables, in the order they are defined, followed by alternative-specific rows.

If required, user-defined labels can be given to the rows, see `$VAL.LABEL` in section 4.14.

## **4.12 \$TAB.VARS command**

The `$TAB.VARS` command is used to define variables added to the tables in `APPLY`. These variables appear as rows, one for each variable defined, under the main part of the tables and can be used to show how continuous variables are influenced by the choice. Up to four variables may be defined, but the first of these is always `DEMAND`.

In each table, each of these rows has an entry in each column, giving the average value of the variable for the observations falling in that column.

All the variables must be specified as arrays with length `alts`.

Dimension 1 is always ROWS, other dimensions are defined by the first 9 variables on the \$DIMENSION command or are set to 1 if that is not sufficient.

Example:

```
$array cost(alts) dist(alts)
$tab.vars cost dist
```

If the user fills the arrays `cost` and `dist` with the correct values for each alternative, the mean values will then appear in two rows under the main part of each table, distinguished by the columns for that table.

User-defined labels can be given to the variables if required, see `$VAL.LABEL` in section 4.14.

`$TAB.VARS` can also be used to produce scenario files containing observed and predicted choices from an `APPLY` run, useful when presenting the fit of a model. Predicted choices can also be used to present the impact of changes in the variables.

Example:

```
$tab.vars (obs=model04o.scn,pred=model04p.scn,
+ scenario=model04) distance time cost
```

This produces both observed and predicted scenarios, with the file names indicated (`obs=model04o.scn` and `pred=model04p.scn`). These files will contain the demand for each alternative and the data items (`distance`, `time` and `cost`) as indicated. These data items will also be added to the `APPLY` tables.

If the file names are omitted, but the subcommands `obs` and/or `pred` are present, the files will be named `obs.scn` and `pred.scn` by default.

### 4.13 \$TITLE and \$SUBTITLE commands

It is good practice to put these commands at the start of the ALO file, as this ensures that the LOG file is better identified, as well as documenting the ALO file itself. The title should identify the model run being made and the subtitle (which is optional) can be used for further identification.

The commands are formatted as follows

```
$TITLE title for the run

$SUBTITLE subtitle for the run
```

The string `@FILE.ALO` will be converted to the name of the control file (`*.ALO`). This string must be preceded and followed by separators (e.g. blanks or brackets) so that `ALOGIT` can recognise it.

The default `$TITLE` is `@FILE.ALO`, the default `$SUBTITLE` is empty.

The use of @USERINPUT is obsolescent and is not advised, as it may fail in newer Windows environments. See Appendix B for a brief description.

#### 4.14 \$VAL.LABEL

\$VAL.LABEL allows the user to input labels for the rows of tables (defined by \$ROW) and for table variables (defined by \$TAB.VARS). This command can be useful when tables are being used extensively to demonstrate aspects of model fit or responsiveness.

\$VAL.LABEL can also be used for scenario output. The value labels are read in from file, and output for use by the Graphical Reporting System (GRS)<sup>34</sup>.

The three possible forms of this command are:

```
$val.label file1 file2    files are input and output respectively
$val.label file           file is input
$val.label * file        file is output
```

The user is responsible for giving suitable names to the input and output files. However, it should be noted that the GRS requires input to be provided in a file named VALUE.LBL.

The file format is best obtained from examples. A tip is to use ALOGIT without a value label input file to generate an output file in correct format, then edit this file manually to label the output as required.

The input file should be in the required format for the Graphical Reporting System, except that:

- TITLE, VARIABLES and DEFAULTS keywords and their subsequent lines may be omitted, i.e. only EXSYS and DIMENSIONS are essential;
- The XY/NOXY identifier on the specifier of each dimension may be omitted and the title for the total of the dimension is not used by ALOGIT (but must be present).

Each of the dimensions in the input file must also be specified in a \$DIMENSION control but all of the \$DIMENSION identifiers need not be given value labels. The name of the dimension will be given the form in the value label file (i.e. regarding upper and lower case) and blanks may be included in the label, which must correspond to underline characters in the \$DIMENSION control specification.

The output file will be in the appropriate format for input to the Graphical Reporting System. However, only the first 10 items from the \$DIMENSION command will be used. Also, value labels will be given in default form if either there is no specification in the value label input or if there are more values than specified in that input.

---

<sup>34</sup> The GRS is a function of the Shell program, which is documented separately.

## 5. Data Transformations

One of the strengths of ALOGIT is the freedom and simplicity of the ways in which data can be transformed. These facilities are provided by the ALOGIT programming language, which is designed to help users adjust their data to define models in a simple way, without worrying about the technicalities of computation.

This chapter describes the basics of the language and then discusses the data items in which data is stored and manipulated. Three final sections present in turn: arrays, logical structures and loops.

### 5.1 Basics of the ALOGIT language

The objective of the ALOGIT ‘language’ is to allow the user to define the model or data processing task for the run conveniently and clearly. The basic concept is that each line, other than \$ commands, in the ALO file is processed in turn (subject to conditional processing and loops, described later in this chapter). This includes the FILE commands described in Chapter 2 and the data transformations which are the subject of this chapter.

The data processing by ALOGIT, other than file input and output using FILE commands, comprises data transformations. In these transformations, a new value is assigned to a data item. Transformations can be made conditionally, using IF... [ELSE... ] END commands or repetitively, using DO... END commands; these are described in sections 5.4 and 5.5 respectively.

The components of data transformations are data items, operators and functions. Data items may be single items (described in 5.2) or array elements (described in 5.3). Operators may be either mathematical (e.g. +, – etc.) or logical (e.g. AND, OR etc.), which are described in section 6.1. There is also a wide range of mathematical and logical functions, described in section 6.2.

Data transformations have a simple syntax

```
a = expression
```

Here a is a data item, either a user-defined single item or array element, or a system-defined single item (among those that are allowed to be changed) or array element. `expression` is a simple or complicated calculation involving data items, array elements, coefficients or numerical or system constants. Data items and arrays used in `expression` must have been defined by a previous transformation, FILE command, \$KEEP specification (with a value), a system data item, or in a \$ARRAY command (which makes all the elements in the array defined).

All of the data items in ALOGIT are stored in the same way, whether they are integers, real numbers or logical values. To help this simplification of the language, a ‘tolerant’ calculation process is made, so that rounding errors can be kept to a minimum. In particular, numbers are considered equal if they differ by less than one part in a trillion, mathematically:

$$ifeq(a, b) = if\left\{\left|\frac{a}{b}\right| < 10^{-12}\right\}$$

(of course, care is taken with zero values).

In processing logical data items, any positive value is taken to be TRUE, zero or negative values are taken as FALSE. When values are the result of a logical operation or function, the value is set to 1 for TRUE and 0 for FALSE.

## 5.2 Data items

### 5.2.1 *User-defined data items*

Users have a wide range of freedom in giving their own names to data items. Letters, numbers and other accepted characters (see section 2.2.1) may be used as parts of user-defined data item identifiers. However, the following rules must be observed.

- User defined data items cannot be more than 10 characters in length.
- Control line identifiers, e.g. \$NEST, FILE, IF, may not be used (\$ command subcommands like `root` or `maxit` are acceptable).
- ALOGIT separators may not be used within names, e.g. `.-`, `+`, `/`, blank.
- ALOGIT function names, e.g. SQRT, RECODE, NORMAL may not be used.
- Identifiers containing only characters used in numbers, (i.e. `+`, `-`, `e`, `E` and `.`) are taken as numerical constants and must obey the rules for those constants (e.g. ‘e’ must not appear twice).
- Identifiers beginning with UTIL or SIZE (or U, S), on their own or followed only by numbers, or BETA or PARAM (or B, P), followed only by numbers, have special functions. These special functions are obsolescent and are described in Appendix B.
- User data items must always be defined before they are used on the right side of a transformation. Definition of a data item takes place when it is given on a \$ command or FILE command or appears on the left side of a transformation.

### 5.2.2 *System-defined data items*

ALOGIT uses some data items for special functions. In some cases it may be possible for the user to use these items in a different way, but this is not advisable as it may cause confusion. The data items commonly used in this way are as follows.

<code>choice</code>	the user must set <code>choice</code> to be the chosen alternative; this is defined by the name or number used in the utility function, or the sequence number of the relevant alternative in the utility functions
<code>count</code>	this is set automatically to the sequence number of the current record and cannot be changed by the user
<code>id</code>	the user can set this to be an observation identifier; the default is to use <code>count</code>
<code>weight</code>	this can be used to assign a weight for each observation; the default value is 1 and the observation will be rejected if the value is 0 or negative

Choice has to be defined in APPLY runs (either by `CHOICE` or `PROP`) if `$DIMENSION` is used (unless `NO_CHOICE` is set) or if `$PRINT PROB` is used. In PREPARE runs, a check is made that the choice is defined properly.

A useful tip in APPLY runs with large data sets is to set `WEIGHT` to a small value such as 0.01 or 0.001 so that APPLY tables do not overflow their formats. Conversely, in small data sets or

when there are many alternatives a large WEIGHT (e.g. 100) can be used to obtain more significant digits.

The following system-defined items are less commonly used and are recommended for use by advanced users only:

<code>suff.stats</code>	this data item is FALSE by default but can be set to TRUE so that the observation will be used in calculating sufficient statistics only, and will not be used for calculating choice probabilities <sup>35</sup>
<code>no_choice</code>	this data item is FALSE by default but can be set to TRUE so that the observation will not be used in sufficient statistics but only in calculating choice probabilities; the data item can also be used to avoid having to set choice in APPLY
<code>logsum</code>	in APPLY runs this data item gives the total utility derived from the choice set; however it is not set until the model has been calculated, so it can be used only in data transformations placed after the utility (and size) functions in the ALO file
<code>jack_knife</code>	for use in jack-knife runs – see the Shell documentation

### 5.2.3 *User-defined constants*

ALOGIT recognises simple numbers as constants and these cannot be redefined. Numbers may use

- digits, giving a value in the usual way, either with or without a decimal point
- powers of 10 may be used to multiply a decimal value, e.g. 1e6 is equal to 1 million – as usual in ALOGIT an upper-case or lower-case E may be used
- a negative sign may be used for either (or both) the value or the exponent, e.g. -1E-6 is minus one millionth.

Note that ALOGIT will accept quite high accuracy in the specification of numbers and will store a numerical value to high accuracy. However, only the first 10 characters of the number are used for purposes of recognition. For example if we put

```
a=1234567890.12
b=1234567890e2
```

then a will be stored to the full 12-digit accuracy, but the number to which b is set will be recognised as being the same as a (because the first 10 characters are the same) and b will therefore not have the correct value. This can be solved by setting

```
b=123456789.0e3
```

The first 10 characters of b are then different from the first 10 characters of a. This type of correction will not be needed often.

### 5.2.4 *System-defined constants*

A small number of constants are pre-defined by ALOGIT for the convenience of the user. These names cannot be redefined:

---

<sup>35</sup> `suff.stats` and `no_choice` work with MNL models only.

alts            the number of alternatives, including both elementary and nest alternatives, but excluding the root  
 true, false= 1, 0 respectively; the longer names may be useful for legibility  
 pi\_root3      = 1.813799364234218  $\approx \pi/\sqrt{3}$ ; this value may be useful in simulations  
 pi\_root6      = 1.282549830161864  $\approx \pi/\sqrt{6}$ ; again may be useful for simulation

## 5.3 Arrays

### 5.3.1 User-defined arrays

The user can define arrays using the \$ARRAY command (see section 4.2). Once an array has been defined, *elements* of the array can be used just like other data items, either on the left side of a data transformation, to assign a value to that element, or on the right side, to be used in assigning values to other data items. For example,

```
$array cost(alts)
...
cost(car) = distance * costperkm
cost_diff = cost(car) - cost(bus)
```

However, in contrast to other data items, it is not checked that a value has been assigned to an array element before it is used. The user must therefore make sure that all array elements that are used do have previously assigned values or that the value zero (set initially) is satisfactory.

### 5.3.2 System-defined arrays

In ALOGIT, eight arrays are reserved for the system and have particular meanings. Seven of these (RANK, PROP, AVAIL, EXCLUDE, FILE\_OPEN and DEMAND) are listed and their meanings are given in section 4.2. The other (ERROR\_COMP) is described in section 7.2.

Generally, these arrays do not have to be defined using \$ARRAY commands and they can be used directly by the user.

If the arrays are used, values have to be given by the user to RANK, PROP, AVAIL, EXCLUDE and ERROR\_COMP, while FILE\_OPEN and DEMAND cannot be changed by the user from the values given to them by the system. Additionally, the indexes of ERROR\_COMP *must* be constants.

Otherwise, system-defined arrays are used exactly like user-defined arrays.

## 5.4 Conditional processing

Often, it is required that some data transformations are performed only under specific conditions. For example, data items needed for the model may be calculated in different ways, depending on the values of particular inputs. This type of processing is handled in ALOGIT by making logical tests. These commands take the form

```
test THEN
..
[ELSE
..]
end
```

Here, `test` may be a data item or a function. Often, a logical function (see section 6.2.2) will be used, but in fact any data item or function may be used here. The only restriction is that `THEN` must appear on the same physical line in the ALO file, so that ALOGIT can recognise that conditional processing is required.

Following the `.. THEN` line, a number of data transformations (including `FILE` commands, for instance) are given. These transformations will be carried out if `test` evaluates as `TRUE`, i.e. if its value is positive.

As indicated in the framework illustrated above, it is possible to insert an `ELSE` command, followed by other data transformations. These will be carried out if `test` evaluates as `FALSE`, i.e. if its value is zero or negative.

## 5.5 Repetitive commands (DO loop)

Particularly in large models, it is often convenient to perform commands repetitively and this facility is offered by ALOGIT in the form of ‘DO loops’. A DO loop contains a number of commands, which are each carried out on each iteration of the loop, until a prescribed limit is reached. An example will clarify:

```
DO dest = 1, ndests
  cost(dest) = dist(dest) * costperkm
              + park(dest) * hourstay
END
```

This loop makes a simple calculation of the cost of travelling to a destination as a function of the distance, the cost/km of travelling, the cost/hour of parking and the number of hours to be parked. These calculations are made once for each destination (indexed by `dest`) up to the total number of destinations `ndest`. At each step `dest` is increased by 1 until it reaches `ndests`, which is the last time the calculations are made. We also see that DO loops are very often used with arrays indexed by the DO loop counter.

To set up a DO loop over all the alternatives in a model, it is possible to specify

```
DO ialt = alts
  ...
END
```

In this construction, the variable `ialt` takes all the values from 1 to `alts` (the number of alternatives, elementary and composite) in succession.

The DO loop set-up can be extended to use a third control parameter:

```
DO counter = start, finish, step
  ...
END
```

This construction can be used to make steps that are more than 1 or even to make negative steps. When `step` is negative, the initial point `start` will of course usually be lower than the end value `finish`. Note that the commands in the loop are always executed at least once, even if `start` is already beyond `finish`, because the test for termination is made at the end of the loop, not at the beginning.

If `step` is zero, an error will be reported.

## 6. Operators and Functions

ALOGIT offers a rich set of operators and functions, which are the main ways in which data items can be transformed.

### 6.1 Operators

ALOGIT recognises 10 *operators*, which typically combine two data items to obtain a result. There are five mathematical and five logical operators.

Note that, although logical operators have an outcome that can be interpreted numerically, they cannot be mixed with mathematical operators without using brackets.

#### 6.1.1 Mathematical operators

Operator	Usage	Notes
+	$a + b$	
- binary - unary	$a - b$ $- b$	
*	$a * b$	
/	$a / b$	fails if b is zero, observation rejected
^ or **	$a ^ b$ , $a ** b$	always 0 if a is 0 otherwise 1 if b is 0 b is rounded to an integer if a is negative

Mathematical operators in sequence are processed according to a hierarchy, whereby operators in a lower row in the table above are processed first, and then in sequence left to right, so

$a + b / c ^ d$  is the same as  $a + (b / (c ^ d))$

$a / b * c$  is the same as  $(a / b) * c$

$a / b / c$  is the same as  $(a / b) / c$

However, unary minus is processed first, so

$a ^ - b$  is the same as  $a ^ (-b)$

#### 6.1.2 Logical operators

Logical operators have single-character and longer forms.

Operator	Usage	Notes
& AND	$a \& b$ $a \text{ AND } b$	TRUE (set to 1) if both a and b are TRUE otherwise FALSE (set to 0)
OR	$a   b$ $a \text{ OR } b$	TRUE if either a or b is TRUE, or if both are TRUE
~ XOR	$a \sim b$ $a \text{ XOR } b$	TRUE if either a or b is TRUE, but not both
# EQV	$a \# b$ $a \text{ EQV } b$	TRUE if both a and b are TRUE, or if both a and b are FALSE (opposite of XOR)

! NOT	! a NOT a	TRUE if a is FALSE
-------	--------------	--------------------

The precedence of logical operators *must* be indicated by brackets, except that sequences of ‘AND’s and sequences of ‘OR’s are allowed. So either of

a OR b AND c

a & !b

will cause an error and a failure during the reading of the control file.

## 6.2 Functions

ALOGIT has 28 *functions*, which give the result of calculations that depend on the *arguments* of the function. These functions are divided into

- mathematical,
- logical,
- random and
- additional functions.

### 6.2.1 Mathematical Functions

The following table lists the 11 mathematical functions available in ALOGIT.

Function usage	Meaning	Failure conditions <sup>36</sup>
exp(a)	exponential of a, i.e. $e^a$ gives zero if $a < -700$	$a > 700$
log(a)	natural logarithm of a	$a \leq 0$
sqrt(a)	square root of a	$a < 0$
abs(a)	absolute value of a	
mod(a,b)	modulus, the remainder when a is divided by b	$b \leq 0$
dim(a,b)	the diminution of a by b, i.e. $\max(a-b, 0)$	
sign(a,b)	a gets the sign of b	$b = 0$
int(a)	a rounded towards zero	
nint(a)	nearest integer to a	
min(a,b,c,...)	smallest number in list	more than 101 items
max(a,b,c,...)	largest number in list	more than 101 items

<sup>36</sup> Function arguments that are out of range will cause an error and rejection of the observation during data processing; argument lists that are too long will cause an error and stop the run during control file processing.

### 6.2.2 Logical Functions

The following table lists the 9 logical functions available in ALOGIT.

Function usage	Meaning
<code>if(a)</code>	TRUE (= 1) if $a \geq 0$ ; otherwise FALSE (= 0)
<code>ifeq(a,b,c...)</code>	TRUE if $a = b$ ; 2 if $a = c$ etc.; otherwise FALSE
<code>ifne(a,b)</code>	TRUE if $a \neq b$ ; otherwise FALSE
<code>ifgt(a,b,c...)</code>	TRUE if $a > b$ ; 2 if $a > c$ etc.; otherwise FALSE
<code>iflt(a,b,c...)</code>	TRUE if $a < b$ ; 2 if $a < c$ etc.; otherwise FALSE
<code>ifge(a,b,c...)</code>	TRUE if $a \geq b$ ; 2 if $a \geq c$ etc.; otherwise FALSE
<code>ifle(a,b,c...)</code>	TRUE if $a \leq b$ ; 2 if $a \leq c$ etc.; otherwise FALSE
<code>ifin(a,b,c,d,e...)</code>	TRUE if $b \leq a \leq c$ ; 2 if $d \leq a \leq e$ etc.; otherwise FALSE; error and observation rejection if $b > c$ etc.
<code>ifout(a,b,c)</code>	TRUE if $a < b$ or $a > c$ ; otherwise FALSE error and observation rejection if $b > c$

**Note:** In multi-argument functions the *first* true condition gives the value. Argument lists that are too long (more than 101 items, for functions with ... in the table above) will cause an error and stop the run during control file processing.

### 6.2.3 Random Functions

The following table lists the 6 random functions available in ALOGIT. These functions return pseudo-random or quasi-random numbers from distributions:

Function usage	Returns a value from the distribution...
<code>uniform(seed)</code>	uniform between 0 and 1
<code>normal(seed)</code>	standard normal (0, 1)
<code>logistic(seed)</code>	standard logistic, i.e. $\log(x/(1-x))$ , where $x$ is uniform (0,1)
<code>gumbel(seed)</code>	standard Gumbel <sup>37</sup> i.e. $-\log(-\log(x))$ , where $x$ is uniform (0,1)
<code>halton(a,b)</code>	gives the $b^{\text{th}}$ number from the Halton sequence generated by $a$ ; $a$ should be a prime to generate non-repeating sequences; will fail if $a \leq 1$ or $b < 1$
<code>randsel(a,b...seed)</code>	multinomial (discrete) with probabilities $a, b, \dots$

Note that `seed` is optional, but brackets must always be given, e.g. `uniform()`. If there is no explicit `seed`, ALOGIT will generate a value from the current time and date. It is usually preferable to use an explicit `seed`, which can easily be done using `$keep`.

RANSEL operates as follows, if the function is called with arguments

`RANSEL(p1, p2, .. list of up to 101 items)` then

- If  $\sum\{p\} = 1$  it returns 1 with probability  $p_1$ , 2 with probability  $p_2$  etc.

<sup>37</sup> Also known as extreme value type 1. This function was erroneously called `weibull` in earlier versions of ALOGIT.

- If  $\sum\{p\} < 1$  it will also return 0 with probability  $(1-\sum\{p\})$ .
- If  $\sum\{p\} > 1$  and  $n$  is the last value for which  $\sum\{np\} \leq 1$  then it will return
  - $i$  with probability  $p_i$  for  $i \leq n$  and
  - $n+1$  with probability  $(1-\sum\{p_i | i \leq n\})$ .
- If  $p_1 \geq 1$  it will always return 1.

The user is responsible for ensuring that  $\sum\{p\}$  has a suitable value. If there are more than 101 arguments an error will be generated in processing the control file and the run will be stopped. RANDSEL will give an error and reject the observation if  $a > 2*10^9$ .

Seeding with RANDSEL works on the LAST argument. If the last argument is a constant, a further argument will be generated from the time and date and used like the seed in the other random functions<sup>38</sup>. Otherwise, the last argument will be updated each time RANDSEL is called and the user will be warned if it is not a \$KEEP item.

#### 6.2.4 Additional Functions

There are two additional functions in ALOGIT, which cannot be classified into the categories above. They both have a maximum of 101 arguments and exceeding this number will cause an error and stop the program in processing the control file.

### RECODE

RECODE allows new codes to be given for specific values of a data item.

```
recode(a, b, c, ...)
```

returns:

- if nearest integer to  $a$  is 1 then  $b$
- if nearest integer to  $a$  is 2 then  $c$ , etc.
- otherwise 0, if the nearest integer to  $a$  is zero, negative or greater than the highest value indicated in the arguments

### DUMP

The DUMP function offers the possibility of writing variable values to the LOG file. It is very useful for checking that data items are being calculated correctly. For example:

```
dump(count, a, b)
```

will report the values of data items `count`, `a`, `b` to the LOG file with their labels, 3 per line. It is quite useful to include `count` or `id` in `dump` commands to identify the record generating the message.

DUMP will not be executed if the observation has a previous error.

---

<sup>38</sup> Note that this behaviour can be forced by using 0 as the last argument.

## 7. Defining the model: Alternatives, UTIL and SIZE functions, Coefficients

In this Chapter the key elements defining the model are described.

### 7.1 Alternatives

Alternatives in ALOGIT are defined primarily by having utility functions. Nest alternatives may also be defined by being specified in \$NEST commands.

Either elementary or nest alternatives may be chosen, but for any alternative to be eligible as a chosen alternative it must have a utility function, though this can simply be set to zero. For example, a nest alternative that was sometimes chosen would usually have a zero utility function.

Alternatives in ALOGIT can be referenced by name or number. This reference is used in UTIL and SIZE functions, as well as in \$NEST and \$ROW commands<sup>39</sup>. Generally names are preferred because they are less liable to error than numbers. When numbers are used, they have a maximum of 6 digits, but there is no limit on the number of alternatives that have names – names like A1234567 can be used. Changes may be needed in the ALO4.INI file to allow large numbers of alternatives (see Chapter 9). Numbers and names may be used together,

In setting CHOICE and in dealing with arrays defined for the alternatives (i.e. arrays with length ALTS), we may have to associate numbers with alternatives, even if the primary definition of the alternatives is by name. A simple way of doing this for small models is to use the RECODE function; for example, we can code as follows:

```
- mode takes a value of 1 if bus is chosen, 2 if car is chosen
choice = recode(mode, bus, car)
avail(car) = licence and car_owned
...
util(bus) = ASC_BUS + .....
util(car) = .....
```

In larger models with many alternatives, this approach is not practical. Then:

- when the alternatives have names in their utility function definitions (i.e. in the brackets on the left of the equation), the numbers associated with them are given by the order of the utility functions;
- when the alternatives have numbers in their utility function definitions, these numbers can be used for setting choice and in arrays.

So, in the example above, for named alternatives, we could use

```
- mode takes a value of 1 if bus is chosen, 2 if car is chosen
choice = mode
avail(car) = licence and car_owned
...
util(bus) = ASC_BUS + .....
util(car) = .....
```

<sup>39</sup> Also obsolescent \$ELAST, see Appendix B.

Here the numbers 1 and 2 are implicitly associated with `bus` and `car` respectively, because of the order of the utility functions. This method is more error-prone than using `recode`, but would be necessary in large models because `recode` cannot be used with more than 101 arguments.

Alternatively, the alternatives can be defined by numbers alone

```
- mode takes a value of 1 if bus is chosen, 2 if car is chosen
choice = mode
avail(2) = licence and car_owned
...
util(1) = ASC_BUS + .....
util(2) = .....
```

This approach is less clear than using names.

## 7.2 Utility Functions

The utility of an alternative is made up of a number of components, like its cost and its quality, perhaps measured in several different ways. These different characteristics are combined together, in a way that is not usually known *a priori*, to give the total utility of the alternative as seen by the average individual. The function of model estimation in ALOGIT is to determine how the characteristics should be combined in the utility function.

Utility functions may be non-random or contain random components.

### 7.2.1 *Non-random utility functions*

Utility functions can be introduced in either of the following ways<sup>40</sup>:

```
U(nnn) or
UTIL(nnn)
```

Here `nnn` is the alternative name or the alternative number.

This is followed an equals sign and by a series of terms, e.g.:

```
UTIL(Car) = bcost * CarCost + btime * CarTime + ASC_Car +
           correction
```

The terms are separated by plus signs (“+”). Each term may consist of:

- a coefficient identifier (e.g. `bcost`, `btime`) multiplying a data item, in which case the coefficient and the data item are separated by a multiplication sign “\*”,
- a coefficient identifier (e.g. `ASC_Car`) or
- a data item (e.g. `correction`).

Coefficients used in utility functions must have been specified previously (see section 7.4). Data items may be single items (user-defined single items, array elements, constants) or the result of calculations; in the latter case the calculations should be enclosed in brackets.

<sup>40</sup> The format `UTILnnn` or `Unnn`, used in ALOGIT 3, is obsolescent, see Appendix B.2.

Even if no utility terms are defined, utility functions must be specified for any alternatives that can be indicated as chosen and for any alternatives for which size variables are defined (see section 7.3). These functions can simply be set to zero if no terms are required, e.g.

```
$NEST CarModes (b_t1) Drive Passenger
...
UTIL(CarModes) = 0
```

This would allow the composite alternative `CarModes` to be indicated as chosen.

### 7.2.2 *Random utility (mixed logit): use of ERROR\_COMP*

ALOGIT implements ‘mixed logit’ through the use of ‘error components’ to make utility functions random. The random terms are set up as additions to the utility function. In the introduction this was shown as

$$V_{jd} = \sum_{k=1..K} \beta_k x_{kj} + \sum_{r=1..R} \gamma_r \xi_{rdj} \quad (9)^{41}$$

In this equation, the first summation gives the non-random part of the utility, while the second part is random. The  $\xi$  variables are the error components (`ERROR_COMP`) described in the following sections, while the coefficients that multiply them ( $\gamma$ ) are to be estimated. The  $\gamma$ s must therefore be indicated in `$COEFF` commands. Error component coefficients cannot also be used as ‘ordinary’ coefficients  $\beta$  used for non-random components.

The ways in which error components can be used to meet the various requirements of mixed logit modelling (random parameters, heteroskedasticity and correlation between the utilities) are described in section 1.2.3.

To apply these methods the user must set error components *before* the utility functions are defined. This is done through commands like

```
error_comp(4) = normal(seed) * cost
```

The index of each error component (i.e. 4 in the example above) must be a fixed positive integer. It is not sufficient to set the index in a `$keep` command. It is good practice to number the error components from 1 up to the number of components required, because ALOGIT stores values on the F14 file (and reads them at each iteration of the estimation) up to the highest index used, so any missing numbers add to storage space and input/output time without adding information to the model.

Error components require the use of pseudorandom or quasirandom functions (i.e. `normal()` in the example, see section 6.2.3). These functions return values with standardised mean and standard deviation, often 0 and 1 respectively. If they are used without further transformation (e.g. without multiplying by `cost` in the example above), then the standard deviation of the term  $\gamma_r \xi_{rj}$  will be equal to the estimated coefficient (i.e.  $\gamma_r$ ); this would be the usual way to model heteroskedasticity or correlation between the alternatives. However, when it is required to model random parameters, it is necessary to multiply the error component by

---

<sup>41</sup> Note that it is not possible to estimate size variable coefficients in an `ERROR_COMP` run, although a fixed size can be specified and a log size multiplier estimated. This feature is omitted here for simplicity.

the relevant variable (e.g. `cost` in the example above). The relevant part of the utility function would then be

$$\beta_{cost}cost_j + \gamma_{cost}\xi_{cost,j}$$

Because  $\xi_{cost,j}$  is equal to the standard random variable multiplied by the relevant cost, we obtain  $\beta_{cost}$  as the mean value of the random cost parameter and  $\gamma_{cost}$  as its standard deviation.

This could be coded in the ALOGIT utility functions as

```
Util(train) = ... + bcost*cost + gcost*error_comp(4)
```

The ... represent other components in the utility function. Note that it is essential to place the coefficient `gcost` **before** the error component.

Correlation between utility functions can be obtained by incorporating the same error components in the utility functions of different alternatives, for example:

```
Util(train) = ... + btrain*error_comp(1) + bcorrel*error_comp(2)
Util(bus)    = ... + bbus* error_comp(3) + bcorrel*error_comp(2)
```

Here, the covariance between the train and bus utilities is given by the square of `bcorrel`, while provision is made for the two alternatives to have different utility variances<sup>42</sup>.

When processing the error components, ALOGIT sets up a loop over all the `ERROR_COMP` specifications. The length of the loop is `ndraws`, which is set on the `$ALGOR` command (see section 4.1). This feature relieves the user of concern about the way in which a loop over draws works. `ERROR_COMP` settings should be kept near each other in the code to reduce the number of commands included in the `ndraws` loop. Note that no calculations relating to `ERROR_COMP` may be made in the `UTIL` functions. An `ERROR_COMP` must not be in any other `DO` loop, and ...`THEN` structures must be opened or closed either entirely within or entirely outside the implicit `ndraws` loop.

Because the loop over draws runs exactly from the first setting to the last setting of `ERROR_COMP`, it may be necessary to include a ‘dummy’ setting in order to get a loop counter incremented. For example, when using the quasirandom `HALTON` function, an index has to be submitted to the function to get the next Halton number, for example:

```
error_comp(1) = 0
draw = draw + 1
error_comp(1) = halton(2,draw) - 0.5
```

This code ensures that the Halton sequence (based on the prime 2) gives *successive* numbers in the Halton sequence, because the incrementing of `draw` is included in the loop. This coding (`error_comp(1) = 0` and `draw = draw + 1`) is not necessary when using pseudorandom numbers based on a seed value, because the seed is incremented automatically by the pseudorandom function.

---

<sup>42</sup> This structure would probably not be estimable unless there were other alternatives in the model, because `btrain` and `bbus` could not be identified separately from each other and from the ‘white noise’ error.

### 7.2.3 Exponentiated error components

A further possibility is to set up some or all of the random variables in such a way that their total contribution has a sign that is controlled (e.g. always negative). This can be done by using exponentiation, extending equation (9) and introducing new ‘shape’ and scale parameters  $\alpha, \delta$  to be estimated, as in the Introduction:

$$V_{jd} = \sum_{k=1..K} \beta_k x_{kj} + \sum_{r=1..R} \gamma_r \xi_{rdj} + x_{K+1,j} \delta_1 \exp(1 + \sum_{r=1..T} \alpha_r \xi_{rdj}) + x_{K+2,j} \delta_2 \exp(1 + \sum_{s=1..S} \alpha_s \xi_{sdj}) \quad (10)$$

The effect of defining a model as in equation (10) is that the random components multiplied by  $\delta_1$  and  $\delta_2$  follow distributions that are constrained to be positive. The data items  $x_{K+1,j}$  etc. that the exponentials multiply can be negative, if it is required that a negative value should appear in the utility, e.g. for the cost of an alternative.

In equation (10), the following coefficients can be estimated by ALOGIT:

- $\beta$  are ‘ordinary’ non-random linear coefficients, as described in 7.2.1;
- $\gamma$  are linear random coefficients, as described in 7.2.2;
- $\alpha$  are ‘shape’ parameters that determine the form of the distribution of the exponentiated coefficients;
- $\delta$  are ‘scale’ parameters that determine the scale of exponentiated coefficients.

To obtain correlation between coefficients that are distributed log-normally it is possible to include multiple error components within the exponential function, placing the same error components in different utility functions to indicate correlation. These variables work in the same way as in the non-exponentiated functions: the correlation between two lognormal variables is not quite the same as between the underlying normal variables, but the difference is not large. In equation (10) this possibility is shown by the summation within each exponentiation term.

To maintain the expected sign of the impact of  $x$  (e.g. that cost should have a negative impact on utility), ALOGIT moves the scale coefficients into the exponentiation, setting

$$x_{K+1,j} \delta_1 \exp\left(1 + \sum_{r=1..T} \alpha_r \xi_{rdj}\right) = x_{K+1,j} \exp\left(\zeta_1 + \sum_{r=1..T} \alpha_r \xi_{rdj}\right)$$

where  $\zeta = \exp \delta$  and the  $\alpha$  coefficients are scaled by  $\zeta$ .

To code exponentiated random terms in ALOGIT, the user must first set up the ERROR\_COMP data items as described in 7.2.2, then code utility functions with terms like

```
Util(train) = ...
+ cost_train *
  exp(cost_mu + cost_sig*error_comp(1)
    + tc_covar*error_comp(2))
+ time_train *
  exp(time_mu + time_sig*error_comp(3)
    + tc_covar*error_comp(2))
```

In these functions, provided the `error_comp` variables take a normal distribution, the time and cost coefficients follow lognormal distributions. To form these distributions, the underlying normal distributions of cost and time have:

- means at `cost_mu` and `time_mu` respectively;
- standard deviations of `cost_sig` and `time_sig` respectively;
- covariance given by the square of `tc_covar`.

To achieve these effects, the signs of the cost and time variables must be reversed, so that the positive exponential function means that the total effect of the term on utility is negative, as it should be.

The most common example of such distributions is to take the underlying variable as being normally distributed, in which case the exponential is distributed log-normally. But in fact the discussion applies equally to any underlying distribution, such as uniform, which would yield a log-uniform distribution. In this context, it may be useful to note that if two coefficients are distributed log-normally, then their ratio (i.e. the value of one in terms of the other) is also distributed log-normally; however, such a simple result may not apply for other underlying distributions.

Note that, within the exponential functions, the shape coefficients must **precede** the error components variables and that the data item (`cost_train` in the first part of the example above) must **precede** the exponential function.

Coefficients cannot be used as for both shape and scale, nor can shape or scale coefficients be the same as linear error component coefficients or ordinary coefficients.

Models including exponentiated error components can be difficult to estimate, because the exponentiation can cause very large values to appear in the utility function. Ingenuity and flexibility may be required to obtain successful estimates!

### 7.3 Size Functions

Size functions are used to incorporate in the utility of an alternative variables that relate to the *quantity* of the alternative, rather than to its *quality* (the latter variables are included in the ordinary utility equations). This type of variable is frequently used in spatial choice (e.g. transportation planning) when the alternatives are zonal aggregates of many elementary alternatives.

Size functions are specified in an analogous way to utility functions, i.e.<sup>43</sup>:

`S(nnn)` or  
`SIZE(nnn)`

Here `nnn` is the alternative label or the alternative number. Size functions may be defined only for alternatives for which utility functions have already been given. If the user does not want to assign utility to an alternative but does want to assign a size function, then a ‘null’ utility function should be used, such as

`UTIL(nnn) = 0`

<sup>43</sup> The format `SIZEnnn` or `Snnn`, used in ALOGIT 3, is obsolescent, see Appendix B.2.

The size functions **MUST** appear in the same order as the utility functions and all the size functions **MUST** follow all the utility functions.

In MNL models with size variables every alternative must have a `SIZE` specifier. Size variables do not necessarily need to be specified for every alternative in tree logit models. It must however be the case, when any size variables are used, that for every alternative one and only one of the set of nodes leading from the elementary alternative to the tree root has a size variable defined. ALOGIT checks that the size variables are properly specified in this sense.

If the desired model does not include a size variable for a specific available alternative, ALOGIT may be given the specification that the size for that alternative is 1, by

```
size(nnn) = 1
```

which will effectively eliminate the size function for that alternative (since after taking the log, there remains nothing to add to the utility function).

The log function is not explicitly stated when defining size functions. The size functions are set out in the same format as the utility functions, e.g.

```
SIZE(Zone01) = POP(01) + b_emp * EMP(01) + b_retail * RET(01)
```

Note that one term must appear without a coefficient, `POP(01)` in this case, because size functions can be multiplied by any factor without changing the model.

To keep the coefficients of the size terms positive, ALOGIT works with the logarithms of the actual coefficients, which can take any value. Then the actual coefficients can be expressed as exponentials of the working coefficients, thus ensuring they are positive. Both internal and actual coefficients are reported, with appropriate error measures.

When all the size variables are zero for an alternative, it will be set as unavailable. If that alternative was chosen, the observation will be rejected. This would imply (in a tree logit model) that ‘lower’ alternatives (if any) would also be unavailable.

Like the coefficients of the utility function, size coefficients may be constrained. If all of the size coefficients are constrained, ALOGIT will ‘pre-calculate’ the total size and treat it as an ordinary variable. It is necessary to constrain all of the size coefficients if the model contains error components (i.e. for mixed logit), since ALOGIT cannot estimate size coefficients simultaneously with error component coefficients.

As indicated in the Introduction, it is possible to multiply the whole log size function by a coefficient. This coefficient is defined by the `$L_S_M` command (see section 4.8 above). If there is no `$L_S_M` command, the log size function is used exactly as indicated. Care must be taken in the use of this option, as it has implications for the meaning of the model, explained in a paper<sup>44</sup>. Usually there is no `$L_S_M` multiplier.

---

<sup>44</sup> Kristoffersson, I., Daly, A. and Algers, S. (2018) Modelling the attraction of travel to shopping destinations in large-scale models, *Transport Policy*, Vol. 68, pp. 52-62.

## 7.4 Coefficients

The coefficients to be estimated by ALOGIT need to be specified as such before they appear in utility or size functions. This can be done in four ways<sup>45</sup>:

- by using the `$COEFF` command (see section 4.3), which is the standard way of defining coefficients;
- by using the coefficient in a `$NEST` command (see section 4.9);
- by defining the coefficient to be the log-size multiplier in a `$L_S_M` command (see section 4.8);
- by reading the coefficients from an F12 file using an `incl.file coeffs` command.

The `$COEFF` command also permits the user to indicate an initial value that a coefficient is to take and to indicate whether the coefficient is to be constrained, i.e. not changed in the estimation procedure. Unless indicated otherwise in a `$COEFF` command, coefficients indicated in `$L_S_M` or `$NEST` commands have the initial value of 1 and are not constrained. Coefficients input from an F12 file have the value and constraint status indicated on that file, unless they are changed by a subsequent `$COEFF` command.

The default starting values are 0 for utility, size and all types of error component coefficients, and 1 for tree coefficients and log-size multiplier coefficients. All coefficients are unconstrained by default and any constraints have to be applied explicitly.

If repeat coefficient specifications are used, the last specification given is used. This feature means that `$COEFF` commands can be used to change default constraint and value settings.

In `$COEFF` commands, the symbols `#` and `=` following the coefficient name are used to indicate unconstrained and constrained respectively. The value follows the relevant symbol. For example

```
$COEFF btime#-0.03, bcost=-1
```

means that `btime` is defined as a coefficient, it has the starting value `-0.03` and it is not constrained, while `bcost` is defined as a coefficient, constrained to the value of `-1`. The symbols `#` and `=` may be used without values to start or constrain coefficients at the current or default value. However, values may not be given without using `#` or `=`.

Note that a coefficient can be used on the right-hand-side in transformations, when it will have its initial value. However, a coefficient *cannot* be changed in value using transformations; only `$COEFF` can be used for this purpose.

---

<sup>45</sup> Obsolescent commands that also define coefficients are `$ELAST` and the coefficient input format used in ALOGIT 3 and earlier with numbered coefficients. See Appendix B for a description of these.

## 8. The LOG file

This Chapter describes the contents of the LOG file. The LOG file contains introductory and closing material, including file documentation, numbered messages issued by ALOGIT and reports. These are described in turn.

Additionally, messages may be generated by the operating system relating to file handling on input or output. These messages are explained in Appendix E.

### 8.1 Introductory and closing material

The first line of the ALOGIT LOG file gives a copyright notice, the version number of ALOGIT (e.g. 4.5) and the licence serial number of the current EXE file. When pagination is requested (see section 9.2 below), this line is repeated at the top of every page, along with the page number. The second line indicates to whom the current EXE is licensed. The licence information is incorporated in the EXE file itself and cannot be changed by the user.

The following lines of output give the title and (if it exists) sub-title of the run, which are controlled by the user through \$TITLE and \$SUBTITLE commands – see section 4.13 above.

The following line gives the directory information for the EXE file being used.<sup>46</sup>

This is followed by a line giving the start time and date of the run.

At this point the program will complete the reading of the ALO file, which will usually give rise to a number of messages (see 8.2). Then a report will be given of the content of the ALO file, the results of the data processing stage, model estimation and APPLY tables (see 8.3).

Directory information is given for all of the input files encountered in the ALO file. This gives documentation of the run for quality control purposes.

Finally, the program gives an overview of the time taken for the run, separated into ALO file processing, data processing (including APPLY tables) and model estimation. Model estimation time will be zero in APPLY or data processing runs.

### 8.2 ALOGIT numbered messages

ALOGIT issues messages of five types. Apart from some hopefully helpful information, each message also includes a specific number, which can be used to ask for help from [helpdesk@alogit.com](mailto:helpdesk@alogit.com).

#### 8.2.1 INFORMATION

INFORMATION messages simply inform the user in ways that may be helpful, such as giving the time taken for particular processing steps. These messages do not require any action from the user. An example is

---

<sup>46</sup> This may not give the date on which the EXE file was created, as directory information can be changed by various operations. However, the creation (compilation) date is given in the output to screen and this may be used in case of uncertainty.

INFORMATION 002: data processing time .02 secs.

### 8.2.2 *WARNING*

WARNING messages indicate a condition that may require action from the user. They do not stop the run but the user is advised to review any WARNING messages to be certain that there is no problem that needs to be solved. An example is

```
WARNING 152: coefficient log_price not used in model
```

This message indicates that a coefficient has been defined but not used in the model. This may be intended but it may also be an indication of an error in the setup, e.g. a typing error. Other WARNING messages indicate similar conditions that are useful for the user to review.

### 8.2.3 *USER ERROR*

A USER ERROR message indicates that a situation has been found where ALOGIT cannot continue. The run will be terminated, though it may continue for a few more steps, possibly indicating several errors that the user needs to correct. A typical (and common) USER ERROR message is

```
USER ERROR 351: unknown code x in transformations
```

This message indicates that the data item *x* is used as if its value was known but it has not yet been defined. The user will need to ensure that all the data items used are defined, e.g. by being input from a file or set in a data transformation, before they are used.

Other USER ERROR messages may be caused not so much by a mistake on the part of the user as by a misunderstanding of the requirements of ALOGIT. If the message is not sufficiently self-explanatory, and reference to Messages.pdf does not help, then [helpdesk@alogit.com](mailto:helpdesk@alogit.com) may be able to suggest an answer.

### 8.2.4 *PROGRAM ERROR*

Like other substantial computer programs, ALOGIT contains errors, though of course every effort has been made to remove them and in particular to remove errors that could cause erroneous results. In some cases, ALOGIT is able to detect that a condition has occurred that should not be possible and in these cases it will terminate with a PROGRAM ERROR message.

When a PROGRAM ERROR message is given, it will usually be accompanied by some diagnostic information, in addition to the error number. This information is not usually helpful to the user, but ALOGIT Software & Analysis Ltd. would be glad to receive reports of these conditions arising, so that corrections can be made in future versions of the software.

Reports of other errors, in particular unexplained termination of the program, are also gratefully received.

### 8.2.5 *OBSERVATION REJECTED*

In the PREPARE phase of operation, ALOGIT tests observations to see whether they are acceptable and useful for model estimation. When they are not, a message may be generated like the following.

```
OBSERVATION REJECTED 518:          565: choice drive          unavailable
```

Note that only a limited number of such messages are printed (as indicated by control \$PRINT REJ.OBS, see section 4.10). After reaching this number, further rejections do not cause specific messages to be printed.

Reasons for the rejection of observations may be any of the following:

- the chosen alternative is defined to be unavailable, as in the example above;
- the defined choice indicator is not an alternative (i.e. has not got a utility function);
- the number of chosen alternatives is not positive and/or the number of unchosen alternatives is not at least 1;
- the number of chosen alternatives in a non-linear model is not exactly 1;
- the weight attached to the observation is not positive;
- a size variable is negative, or all of the size variables are zero;
- an error is encountered in the data processing (e.g. an attempt to take a square root of a negative number)<sup>47</sup>.

The user should check these rejections and ensure that they are consistent with correct working of the code.

## 8.3 **ALOGIT reports**

ALOGIT works in three main steps: processing the ALO file, processing the data (including PREPARE and APPLY steps) and estimating the model. The reports produced by these steps are described in the following sections, with the APPLY tables covered in the last section.

### 8.3.1 *Report of ALO file*

A series of reports may be generated from information input in the ALO file, in the following order.

- If ERROR\_COMPONENTS have been defined, a listing is given of which components are attached to which alternatives and what their coefficients are.
- If a Value Label file is present for input (see section 4.14), a report is given that lists, for each DIMENSION, the label attached to each value.
- If a \$PRINT control is present in the ALO file (see section 4.10), all of the print controls are listed (i.e. even those that are not changed from their default values).
- If a \$GEN.STATS control is present (see section 4.6), information is output about the variables that are selected.

---

<sup>47</sup> This error may also occur in a data processing run.

- If a \$ESTIMATE control is present (see section 4.5), all of the estimation controls are listed (i.e. even those that are not changed from their default values).
- If a \$ALGOR control is present (see section 4.1), all of the algorithm controls are listed (i.e. even those that are not changed from their default values). However, CON.BHHH and NDRAWS are not printed if they still have their default values 0.
- If a \$DIMENSION command is present (see section 4.4), the reporting dimensions are listed. However, this report is skipped if the more detailed report from a value label file (see above) is given.
- If a \$TAB.VARS command is present (see section 4.12), the continuous variables that will appear at the bottom of tables are listed.
- If scenario files are used (see section 3.3.10), either for observed choices or for predicted choices, then this is reported, together with the labels assigned to those files.
- If one or more \$KEEP commands are present (see section 4.7), all of the \$KEEP items are listed, together with items input from files, which are protected in the same way as \$KEEP because changing them can cause unexpected results.
- Any \$ELAST commands are listed; however, these are obsolescent (see Appendix B.2.3) and should not be used.
- If one or more \$ARRAY command is present (see section 4.2), all the arrays are listed, together with their lengths.

If \$PRINT TRANSFORMS is set (see section 4.10), the data transformations will be printed in the form in which they are interpreted by ALOGIT. Both the initial transformations and the final transformations are printed, if the latter are provided in an APPLY run (see section 2.1.5). These prints may be helpful in understanding exactly how ALOGIT is interpreting the ALO file, but the user is warned that the output may be lengthy.

If coefficients are defined (using \$COEFF, see section 4.3, or other specifications), these are listed, giving the label, constraint indicator, initial value and type, i.e. the way in which the coefficient is used in the model.

Finally, if a tree is defined (using \$NEST, see section 4.9), a diagram is printed showing the connections that have been defined. While this may be useful in understanding issues concerned with the nesting, the user is warned that each alternative produces one line of output and the output can be lengthy.

Utility functions are not reported at this stage but may be output together with summary statistics about their components (and \$ROW indicators, if relevant) at the conclusion of data processing.

### *8.3.2 Report of data processing*

During data processing, any observations giving rise to exceptions will be indicated (see section 8.2.5). At the conclusion of the processing, two types of reports may be printed:

‘matrix’ and ‘vector’ statistics. Following these reports, messages are printed summarising the numbers of observations rejected for various reasons.

### **Matrix statistics**

‘Matrix’ statistics are so named because each variable is considered across the alternatives and observations. Only variables that appear in utility functions as linear additions or in size functions can be included (i.e. variables appearing in random components cannot be presented in this way, but see their presentation as vector statistics in the following section). The outputs are controlled by \$PRINT MAT.STAT (see section 4.10). If this parameter is set to 0 no matrix statistics output will be produced (the default is 1).

If MAT.STAT is set to a positive value, 1 or more, the first set of matrix statistics will be produced.

- First, output is produced for each alternative, indicating the number of times it was chosen, available but unchosen, available and unavailable. If parameter \$PRINT CHOICE (see section 4.10) is set TRUE, then only alternatives that are chosen at least once are included in this output. The user is warned that the output can be lengthy and the CHOICE parameter is provided to help in controlling that.
- Second, for each variable included in the utility functions, output is provided giving the ranges: minimum and maximum values for the chosen alternative, minimum and maximum differences between the value for the chosen alternative and any other alternative. The differences must include both positive and negative values, otherwise the coefficient relating to that variable will become infinite. Zero difference is not sufficient: the values must include strictly positive and strictly negative numbers. These ranges are tested and an error is generated when there is no overlap.

If MAT.STAT is set to 2 or more, further statistics concerning the variables included in the model are produced.

- First, the mean and standard deviation of the variables attached to the chosen alternative and the differences between other alternatives and the chosen alternative are produced. These statistics, in particular the standard deviation of the differences, indicate the amount of information available to estimate each parameter.
- Second, correlations of the differences between unchosen and chosen alternative values are given. These statistics are useful to identify excessive correlation in the data that may cause issues in estimation.

Note that each unchosen alternative and each observation makes a contribution to these statistics, so that interpretation of the outputs may be imprecise. Nevertheless, these statistics have proven their worth in helping to find problems that impede estimation.

### **Vector statistics**

‘Vector’ statistics are so named because each variable relates to a single data item across the observations. They are controlled by \$GEN.STAT (see section 4.6), which generates a list of the data items to be included in the statistics. The key subcontrols of \$GEN.STAT are:

- a list of data items;
- the keyword ALL, which means that all named data items are included, which extends to system data items; this can of course give large numbers of data items;
- the keyword UTILITIES, which has the effect that statistics are given for variables included in the utility function; variables identified in this way are handled for available

alternatives only, while variables identified in other ways are handled irrespective of alternative availability;

- the keyword CORREL, which gives correlations between variables identified in the previous ways; note that this can produce a large amount of output;
- the keyword ERROR\_COMP, which produces statistics on the random variables in the model, separately from the non-random variables, except that the keyword CORREL controls both random and non-random correlations.

For non-random variables, the simple statistics produced are:

- the percentage of non-zero values in the data;

then, for non-zero values only:

- the minimum and maximum values;

then, if the minimum and maximum values are different:

- the mean and standard deviation of the values.

When variables are specified by utility functions, the values are preceded by a header giving the alternative name, number of observations for which it is available and the indication of the row to which it is assigned in tabular outputs (see section 4.11).

If keyword CORREL is specified, these simple statistics are then followed by the correlations, in which zero values are included.

Statistics for error components are also presented first as simple statistics and then as correlations. The simple statistics are:

- the index number of the error component;
- the overall mean value;
- the standard deviation within observations;
- the standard deviation of the observation means between observations;
- the minimum and maximum values.

If CORREL is specified, the correlations of the error components are then presented, first within observations and then of the observation means between observations.

### 8.3.3 *Model estimation reports*

Reports on the estimation process are produced either at each iteration or at the termination of the estimation process. At termination, the report depends on whether the estimation has converged or not. Further, termination reports depend on whether the user has requested classical or ‘robust’ error estimates, or both. These functions are controlled by \$PRINT ITER and \$PRINT EST.ERROR (see section 4.10).

An initial print reports any coefficients that are newly freed from constraint, e.g. when an F12 file has been input or when moving from a linear model to a non-linear model.

At each iteration before termination, in addition to ERROR etc. messages as described in 8.2 above, prints are produced depending on the value of the ITER control. Specifically, if this parameter takes the value:

- 0: no print is produced until termination;
- 1: (default value) the screen message indicating progress in convergence is echoed to the LOG file;
- 2: the screen message (as for 1) is echoed and a print is given of coefficients with substantial change, i.e. a change of more than the coefficient standard error times 0.1

or 20 times the convergence limit, whichever is larger – these messages are *not* printed for higher values of ITER as their contents are included in the more detailed information that is produced;

- 3: a report is given of the use of an approximate matrix, if this matrix is needed; on each iteration, a report is given of the current function value, coefficient values and the error matrix; the error prints (standard errors and correlations depend on the value of EST.ERROR on the \$PRINT command (see 4.10): the value 1 (also default) gives robust errors only, the value 2 gives both robust and classical errors and the value 3 gives classical errors only;
 

‘T’ values are given to 1 decimal place. For nest coefficients these are calculated for the difference between the coefficient value and 1, as well as for the difference from 0. For size coefficients, the values used in the utility function are given with their standard errors, as well as the logs of these values and their errors.

Correlations are printed multiplied by 1000 but without decimals (i.e. an output of -345 means that the correlation is -0.345), accurate to the number of digits shown. A correlation output of 1000 therefore means that the correlation exceeds 0.9995. The width of the output depends on the print width selected in ALO4.INI (see section 9.2) and can be as much as 23 columns.
- 4: output is given as for option 3, plus the first derivatives are output at each iteration; the value ITER=-1 can be used to obtain first derivatives at convergence only;
- 5: output is given as for option 3, plus details are given of the operation of the algorithm; also the step change in the coefficients is printed and, if this is found to be unacceptable (too nearly perpendicular to the first derivative) the adjusted step is also printed; details of adjustment of the step by the CUBIC process are also given, if this is used;
- 6: output as for option 5 is produced and in addition a print is given of the second derivative matrix, or, if the matrix is not being calculated (e.g. because CON.BHHH is set), or is not invertible, the BHHH matrix; the inverted matrix is also printed; matrix elements are printed with 5 significant figures, each row of the matrix is labelled with the coefficient identifier and the columns of the matrix are then output in print rows of up to five columns; first derivatives are also output.

The values ITER=-1 and ITER=6 are intended for special situations only.

Robust errors give a measure of the error of estimation *even when the model specification may be in error*. In contrast, classical errors are based on the assumption that the model is correct. Robust errors are calculated using the ‘sandwich’ matrix<sup>48</sup>, which involves the inverse of the classical error matrix and the BHHH matrix, so that neither classical nor robust errors can be calculated when the second derivative matrix cannot be inverted. In these cases, the BHHH matrix is used, as this can always be inverted (in the absence of linear dependency). The sandwich matrix gives the errors in the case when the criterion that is maximised is not the true likelihood of the behaviour on which the model is based. This is a statistical criterion and will not give satisfactory results when the model is simply wrong, i.e. it improves the error estimations, but the coefficients may still be biased if the model is not correct.

At termination, reports are printed giving the cause, i.e. convergence or error conditions. If convergence has occurred or if the specified number of iterations has been made without reaching convergence, summary messages are given of the log likelihood (zero coefficients, constants only, initial and final) and the  $\rho^2$  statistic with respect to zero and constants-only

<sup>48</sup> Specifically  $R = H^{-1}BH^{-1}$ , where  $H$  is the classical error matrix of second derivatives (the Hessian) and  $B$  is the BHHH matrix.

models.<sup>49</sup> The format of the likelihood is adjusted to try to retain the maximum useful information.

If a linear logit model is being estimated as a preliminary to a nested or mixed logit model (LIN.FIRST is set TRUE by default or on the \$ALGOR control, see 4.1) then a full report is given of the linear model convergence before starting on the non-linear model.

If the model estimation fails and the ITER parameter is set to a value less than 6, then it is set to 6 and a ‘diagnostic iteration’ is made that may indicate more clearly what the difficulty is that has caused the run to fail.

Note that when JACK\_KNIFE is set (see 5.2.2 and the Shell documentation) the sign of error component coefficients will be changed as necessary to be positive in the F12 file. This is to ensure that the jack-knife calculations made by the Shell are not distorted by arbitrary sign changes. The values printed to the LOG file have the signs as estimated.

#### 8.3.4 *APPLY tables*

In an APPLY run, the main output to the LOG file is a series of tables. First, however, the probabilities for the chosen alternative may be output, as indicated by the subcontrol PROB on the \$PRINT command. Observations with a probability for the chosen alternative less than PROB are output, so the value 0 (which is the default) produces no output. Messages produced during data processing, e.g. those given by error messages or produced by the DUMP command (see 6.2.4) will be interspersed in this output.

If tables are being printed *and* the model coefficients are input using the \$ESTIMATE MODEL= construction (see 4.5), a test is made that the log likelihood of the chosen alternatives is the same as was found in estimation (to within 0.001). This test can be used to be certain that the model specification and coefficients reproduce the values present at the conclusion of model estimation. Of course, if variables in the model are changed to make forecasts, it will not be possible to reproduce the likelihood found in estimation. The result of this test is printed before the tables.

The tables are defined by the \$DIMENSION command (see 4.4), one table for each data item listed on the command; these data items should be exogenous variables. In each table, the rows are defined by the choices being modelled, which can be done in several ways as described below. The columns correspond to the values of the relevant exogenous data item from the \$DIMENSION command. If the data item is not positive for a given observation, that observation is dropped from the table, as it is when the value exceeds 20, which is the maximum number of columns. The number of observations dropped from each table is reported.

The choice indicators are by default the alternative numbers, as defined by the order of the utility functions in the ALO file. More sophisticated grouping and arrangement of choices into rows can be defined in the \$ROW command (see 4.11). Three different specifications can be used, which are explained in 4.11, which also indicates how the rows will be labelled in each case.

---

<sup>49</sup> The  $\rho^2$  statistic is calculated as  $\rho^2 = 1 - L/L_b$  where  $L$  is the optimum likelihood that has been achieved and  $L_b$  is the base likelihood, with coefficients either all zero or with alternative-specific coefficients only. So if the model gives an (impossible) perfect fit,  $L = 0$  and  $\rho^2 = 1$ , whereas if the model gives no improvement on the base,  $L = L_b$  and  $\rho^2 = 0$ .

In each cell of the table, three numbers are printed:

- the ‘number chosen’, which is the observed number of records with the relevant values of the exogenous variable and the choice indicator;
- the ‘standard deviation chosen’, which refers to the distribution to be expected of the number chosen on the assumption that the model is correct – that is, the fit of the model to the data is assessed by testing how likely it is that the observed data would be obtained if behaviour followed the model; the calculation is made by summing the variance for each observation, then calculating the standard deviation;
- the ‘number predicted’, which is obtained by summing the probabilities of the relevant choice indicator for all the observations with the relevant value of the exogenous variable.

All of these calculations are weighted, using the WEIGHT defined for each observation. If the total weight of the observations is less than 10,000, a single decimal is printed; otherwise, the nearest whole number is printed. WEIGHT can be reduced if necessary (e.g. by a factor of 100) to preserve legibility in the tables.

Additionally, in each cell, when the numbers observed and predicted are not equal, a plus or minus sign is printed to indicate the direction of difference: plus means that the prediction exceed the observed. Stars may also be printed, indicating the magnitude of the difference in terms of the standard error: one star means 1-2 standard errors, two stars means 2-3 and three stars means more than 3 standard errors difference. These star prints can be used to indicate the importance of differences between observed and predicted numbers, but they do not have any specific statistical meaning.

In addition to the numbers in the cells, row and column totals are calculated. Because of the way the calculations are made, the column total observed and predicted must match exactly, but the row totals may not, and standard errors and stars are therefore printed for these also. The overall total (bottom right in the table) can be used to check whether any observations have been excluded from the table because the exogenous variable did not have a value between 1 and 20 inclusive. Observations are excluded when the column is out of range for each table independently, but when the row is not properly defined the observation is excluded from all the tables and a specific message is given before the tables are printed.

To improve presentation, rows and columns of the tables can be labelled. This is primarily achieved by the command \$VAL.LABEL. The possibilities for using this command and the associated file are described in section 4.14. In many cases it will be sufficient to use the default labelling, which is generally clear.

In addition to the comparisons in the cells of the table, it is possible to compare the observed and predicted values of continuous variables, separately for each value of the exogenous variables defining the columns. These results are printed below the tables and are useful for comparing results like expenditure in models of purchase choice, trip length in transport models, etc.. They are controlled by the \$TAB.VARS command: see 4.12, which gives details of the arrays that are needed.

The features offered for continuous variables, dimensions and labelling are designed in part to interact with the reporting system provided in the Shell through the use of scenario files. See the relevant sections of this document (4.4, 4.11, 4.12 and 4.14) and the Shell documentation for more details.

Following each table, summaries are given of the fit in terms of the root-mean-square difference between the observed and predicted numbers in each cell (normalised by the relevant standard error) and the total number of stars printed in the table. These results should be treated as indicative only.

Finally, for each table, a summary is given of the number of cells for which each number of stars has been printed (0, 1, 2 and 3, negative and positive). A  $\chi^2$  test is then made against the null hypothesis that this distribution of stars could be obtained from a normal distribution of the differences. The  $\chi^2$  value and an indication of its significance level are then printed. These values can be used to determine whether the model is giving reasonable results with respect to the exogenous variable, but care must be taken as deviations from the normal distribution can be obtained if the model fits the data unexpectedly well, e.g. because the exogenous variable used to define the columns has been adequately included in the model specification.

## 9. The initialisation file

ALOGIT can use sufficient dynamic storage (i.e. core or RAM) to deal with the size of problem being addressed. When the problem is large, the user must request sufficient storage for the problem and this is done in the initialisation file, named ALO4.INI. The INI file is also used to initialise printing parameters and for the setting of certain test and communication parameters.

If the file ALO4.INI is present in the current directory, it will be used. Otherwise ALOGIT will look for the ‘global’ ALO4.INI in the directory where the program itself (EXE file) is stored. If neither of these directories contain a file, the default values for all the parameters will be used.

The most convenient way to change these parameters is by using the Shell program, which can edit ALO4.INI in either the local or global directories, under the Edit or Tools menus respectively. However, the file may also be edited by a simple text editor, noting that the variables are stored on 3 lines in fields of 5 characters, with the first 5 characters reserved for a label.

### 9.1 Dynamic storage

Five parameters are used to define the storage that is pre-allocated for reading and processing the ALO file and processing the data. Estimation of the model requires different storage allocation, in which the number of parameters is the key, but this is done using the actual values for each run, which are known after data processing, rather than predefined numbers.

parameter	default (thousands)	description	storage requirement per unit
mdata	50	maximum number of data items	26
mp	10	maximum number of parameters	8
malt	10	maximum number of alternatives	24
mut	20	maximum number of terms in utility functions	8
mtran	80	maximum number of transformation codes	4

For large models it will be necessary to increase these numbers. A degree of trial and error may be necessary in doing this, as it may be difficult to determine the numbers of data items and transformation codes *a priori*. Also, it is not advisable to choose very high numbers, as the allocation of RAM on modern machines is not transparent and errors can be caused by requesting too much RAM. Such errors are not easy to eliminate, as clear messages are often not given.

The RAM requirement for each unit in bytes is shown in the final column of the table. It can be seen that the main issue is most likely the number of data items mdata, as this needs 26 bytes for each item and the number of items is likely to be large. Most care should therefore be taken in setting mdata.

Other program limits are not expected to impact on problem size: see Appendix D.

## 9.2 Print settings

Three parameters are used to define print settings for the LOG file, as shown in the table, although only the first two of these are accessible through the Shell program.

parameter	default	description
maxcol	86	maximum width of print lines
maxlin	0	number of lines printed on a page the value 0 suppresses pagination
kpcon	1	print conversion, the value 2 uses simpler characters in printing APPLY tables

It is not expected that the user will need to change these parameters from their default values.

## 9.3 Test and communication settings

These parameters control some test outputs and also allow communication with the user to be adjusted.

parameter	default	
qlist	false	set to true to list all the codes used in the run to codelist.txt
qmem	false*	set to true to report use of RAM to LOG file
qwarn	true	set to true to give a before overwriting an existing F12 file
kinfo	0	set positive to give a screen report every kinfo lines when reading the ALO file and every kinfo records when processing data
qiter	false*	set to true to offer a manual extension of iterations if the run has not converged after the indicated number
qkill	false	set to true to delete the scratch files (F11 and F14) after the run has completed

\* Note:

- qmem can be set to true only if qlist is also true;
- qiter is operative only if kinfo > 0.

In ALO4.INI only 5 values are stored. The first value gives qlist and qmem, with 0 indicating both are false, 1 indicating qlist is true and qmem false, 2 indicating both are true.

## APPENDIX A: ALOGIT system files

ALOGIT works with three ‘system files’, which are designed to be processed efficiently by the program but which are not intended to be used in other applications.

- the ‘.F11’ file, a binary file which contains a compressed form of the data needed to estimate a logit model,<sup>50</sup>
- the ‘.F12’ file, a text file in a specific format<sup>51</sup> which contains the coefficient estimates and other statistics about the model estimation;
- the ‘.F14’ file, only used for ‘mixed logit’ modelling, which contains the random numbers used to support model estimation.<sup>52</sup>

### A.1 The F11 file

The F11 file contains the data needed for model estimation. All of the information for each observation that is accepted in the PREPARE stage is stored in a single record on F11, so that each observation can be processed separately. This organisation is the key to allowing ALOGIT to process data sets of indefinite size.

The data that is stored is processed to retain only the minimum required for model estimation; for example, data items that are zero are simply not stored; alternatives that are not available are similarly omitted. The data is then recorded in binary format. These features contribute substantially to the speed of ALOGIT.

F11 is read, record by record, at each iteration of the estimation procedure but is not changed by that procedure.

The way in which the data is organised on F11 is complicated and this, together with the binary format, make it difficult for the user to read the file or use it in any way other than as the basis for model estimation by ALOGIT itself.

### A.2 The F12 file

The F12 file gives the main results of the model estimation: run title, coefficients, standard errors and their correlations, together with summary information about the final, null and constants-only log likelihood values, number of iterations and convergence condition, the number of observations and the date and time of the run. This file is stored in plain text and gives both coefficient estimates and standard errors with more accuracy than is given in the LOG file. Details of the F12 file are shown in the table below.

The information in the F12 file relates to the last iteration at which a correct calculation of the likelihood and its derivatives could be made. Typically, this is one iteration before the termination of the run and in principle there can be small differences between the F12 file and the final best-estimate values reported in the LOG file. The reason for this is that it is not known whether the coefficient values given in the LOG file can be used to make valid function

---

<sup>50</sup> For very large models, extension files may be needed, which will have extensions like ‘.FA1’, ‘.FB1’ etc., as the operating system does not allow files larger than 4.2 Gigabytes (4.2 \* 10<sup>9</sup> bytes, i.e. 2<sup>32</sup> bytes).

<sup>51</sup> The format is designed to be input to the ALOGIT Shell program or to future runs of ALOGIT itself. Some users have also programmed Excel to accept F12 files for processing in spreadsheets for various applications.

<sup>52</sup> As for F11, further files may be needed, with extensions like ‘.FA4’, ‘.FB4’ etc..

calculations and it is essential that the F12 file can be used to restart a run. If the run has converged with a reasonable convergence criterion (such as the default value) it is unlikely that differences will be apparent between F12 and the LOG file. But if the run fails on the first iteration no new values will be put on the F12 file.

The estimation errors given on the F12 file can refer to either the classical or robust errors. Usually, robust errors are used, unless the user has required classical errors using the EST.ERROR subcontrol on the \$PRINT control (see section 4.10). Approximate errors using the BHHH matrix will be provided if the exact second derivative matrix cannot be inverted.

Files L12 and J12, if produced, have formats identical to F12. The L12 file is produced when a run is made using the linear-first option and it gives the results of that first linear estimation. A J12 file is produced when the Jack-knife option is used in the ALOGIT Shell program: it contains the Jack-knife estimates and their statistics.

Apart from being used to re-start runs, the key use of these files is as input into the ALOGIT Shell program option for comparing models. In this context it may also be useful to note that files in the F12 format can also be produced by Biogeme and Apollo.

**Specification of F12 file**

Line 1, title, characters 1-79

Line 2, subtitle, characters 1-27, and time-date, characters 57-77

Line 3, "END" (this is historical!)

Lines 4-(K+3), coefficient values, K is the number of parameters  
 characters 1-4, “ 0” (again historical)  
 characters 6-15, coefficient name  
 characters 16-17, “ F” or “ T” (indicating whether or not the coefficient was constrained)  
 characters 19-38, coefficient value  
 characters 39-58, standard error

Line K+4, characters 1-4, “ -1” indicates end of coefficients

Line K+5, statistics about the run  
 characters 1-8, number of observations  
 characters 9-27, likelihood-with-constants  
 characters 28-47, null likelihood  
 characters 48-67, final likelihood

Line K+6, more statistics  
 characters 1-4, number of iterations made  
 characters 5-8, error code on completion, 0 indicates success  
 characters 9-29, time and date

Lines (K+7)-however many we need, correlations\*100000, output as integers  
 10 per line, fields of width 7, with correlation  $i, j$  ( $i > j$ ) in position  $(i - 1) * (i - 2) / 2 + j$ ,  
 i.e.: correlation (2,1), (3,1), (3,2), (4,1) etc.

### **A.3 The F14 file**

The F14 file contains the error components used in mixed logit estimation. These numbers are stored in binary format and in double precision.

In addition to some simple header information, the numbers that are stored are

- for each observation,
- for each draw (up to `ndraws`) and
- for each error component up to the highest numbered error component.

Eight bytes are required for each of these random numbers, so the file can become quite large.

The file is read observation by observation, draw by draw, so that ALOGIT can work with an indefinite number of observations and an indefinite number of draws. The file is created in the PREPARE stage and is not changed during estimation. It may also be read by APPLY.

## APPENDIX B: Previous versions and obsolescent features

The current version of the program is 4.5; this Appendix lists the main improvements that have been made over the various versions. First, the improvements offered by 4<sup>th</sup>-generation ALOGIT versions (i.e. 4.x against 3.x) are described, then the various improvements that have been made by successive 4<sup>th</sup>-generation versions (i.e. different versions of 4.x). Some comments are made about versions 3 and earlier. Finally the obsolescent features in ALOGIT, i.e. features that may not be supported in future versions, are also enumerated and explained.

### B.1 Improvements in Versions 4.x<sup>53</sup>

As implied by the numbering, ALOGIT versions 4.x are much improved and extended relative to versions 3.x and earlier. They can perform all of the functions of previous versions and contain many improvements, such as:

- incorporation of ‘mixed logit’ models through the definition of ‘error components’ (included in version 4.5 and in earlier versions that have the EC suffix);
- greatly improved control file, with IF..THEN..ELSE.. logic, ‘DO’ loops, better exclusion controls and alternative availability, the use of arrays, labels as well as numbers for alternatives, improved NEST specification and ‘include’ files;
- accepting one, several or many data files in differing formats, either for sequential processing (e.g. mixed RP and SP data) or for linking (e.g. data describing alternatives); binary matrices can also be input;
- much greater flexibility of the program, it can be used for general data processing and many forecasting tasks;
- the problem sizes have been greatly increased and are now limited effectively only by the size of the computer.

Many other improvements and error corrections have also been made.

ALOGIT 4 is provided to operate under 32-bit Windows, with a Shell program, newly revised in 2022. ALOGIT 4 has been used since 2001 by many organisations (and since 1995 in preliminary versions by Hague Consulting Group) and is thoroughly tested.

The ALOGIT 3 user will find ALOGIT 4 a great step forward. All the familiar features are there, often in a simplified and improved form, together with many new facilities.

Much of ALOGIT has been completely rewritten for Version 4: in particular, the way in which the control file is handled and the data processing sections. The objective was to produce a program that:

- would implement mixed logit models, as described in section 1.2.3;
- was easier to use and to understand;
- that had much improved capabilities, particularly with respect to the handling of data files; and
- which could accept very much larger problems.

---

<sup>53</sup> The addition of EC to the version indicator, as in 4.3EC, means that the program is capable of running mixed logit models using the error components specification. EC versions are otherwise identical to the versions without the EC suffix. Version 4.5 incorporates the EC capability.

Effectively **all** of the features of ALOGIT 3.8 have been retained and many more have been added.

The improved ease of use is achieved through including almost all of the controls for ALOGIT in the *ALO* file<sup>54</sup>. ALOGIT is then run by opening the *ALO* in the Shell or giving the name of the control file on the command line; the functions to be operated and the data files to be input and output are indicated within the control file. Reports are given in an improved *LOG* file.

### *Control File*

The control file has been completely re-specified and some extensions to data processing have been made. An important improvement is the *INCL.FILE* command to allow lines to be ‘imported’ from other files, including coefficient (F12) files.

All the control lines are identified by starting with a \$. There are 16 of these commands (ALOGIT 3.8 had 20), giving access to many new features. Among these are a new *\$NEST* command which gives a much more intuitive definition of tree models and their structural coefficients (replacing *TREE* and *THETA* commands in 3.8) and a completely new *\$KEEP* command which allows processing to link information across records by preserving data between observations. The *\$ESTIMATE* command is mainly used to indicate which of the main functions of ALOGIT is to be operated; this command includes a *LIN.FIRST* option so that the ‘linear equivalent’ of tree logit and mixed logit models can be estimated as a first step.

A new construct is the use of **arrays**, set up by using the new *\$ARRAY* command. An array can be used to refer to a block of data, containing a specified number of items, with an indexing to refer to individual items within the array. An important example of this use is the array *EXCLUDE(.)*, the means used to eliminate observations from processing, when it is often very useful to know how many observations are excluded for each reason: this can be achieved by using *EXCLUDE(1)* for the first exclusion, *EXCLUDE(2)* for the second exclusion, etc..

Another important use of arrays is *AVAIL(.)* to indicate the availability of alternatives. In this array, as well as in the utility functions *UTIL(.)*, names can be given to the alternatives (e.g. *car*, *buy\_one*), although numbers can still be used as in ALOGIT 3. Other alternative arrays are *PROP(.)* to indicate the proportion choosing each alternative in aggregate data, *RANK(.)* for ranking data input and *DEMAND(.)* for the output of forecasts.

To control the data processing, the standard *if. THEN. ELSE. END* construction has been introduced. Loops can also be set up, using *DO. END*, often very useful in conjunction with arrays.

To facilitate simulations, a range of random functions has been introduced, such as *UNIFORM(.)*, *NORMAL(.)*, *GUMBEL*<sup>55</sup>*(.)* etc.. These mean that most simulations can be performed entirely within ALOGIT. Some constants are defined (e.g. *pi\_root3*) to help this processing. A ‘seeding’ facility means that runs can be reproduced exactly. ‘Halton’ quasi-random numbers are easy to specify and use.

---

<sup>54</sup> A few controls are set in the file *ALO4.INI* – see Chapter 9.

<sup>55</sup> Originally introduced under the incorrect name *WEIBULL*.

## Data Files

The improved data file capabilities allow files to be linked to the main data file in three distinct ways.

- It can be specified that a data file *follows*, indicating that it is to be processed as the main file after the first main file is processed. Several such files can be specified to be processed in succession. A key point is that the successive main files do not have to have the same format or even to have the same variables, meaning that completely distinct data – e.g. from RP and SP sources – can be analysed together.
- A data file can be *keyed*, meaning that information is to be taken from it to match the records of the main files, according to the values of certain *keys*. This feature has two principal uses in transport planning analyses: data from household surveys containing (for example) household, person and trip data can be linked together to get **all** the information about a trip at the same time; or information from ‘level of service’ matrices (whether in ASCII files or in compressed binary form) can be linked according to the origin and destination of the records on the main file. Quite complicated structures of linking can be specified.
- Data input from a file (or from several files) can be *common*, meaning that it is held unchanged throughout the processing and applies to every observation.

Output files can be defined in most respects equivalently to input files, so that ALOGIT can be used for a wide range of forecasting and data processing tasks. File formats are easily controlled.

Additionally, *dummy files* can be defined, containing no data but very useful to control the generation of records in simulation processes. All of these file features are operated by the very powerful *FILE* command.

### *Versions 4, 4.1, 4.2 and 4.3 (also EC variants)*

Versions 4, 4.1 and 4.EC of ALOGIT were available from 2001 to 2005. These versions introduced the features of ALOGIT 4 listed in the previous section. The further changes offered by later versions are as follows.

Version 4.2 was available from 2005 to 2011, with improvements:

- extension of the possible size of the scratch files to over 2 Terabytes each;
- elimination of spurious error introduced into ‘Jack-knife’ runs (controlled by the Shell program) caused by arbitrary sign changes to random coefficients;
- numerous other minor improvements and bug fixes, both to ALOGIT and to the Shell program, in particular giving the user the option in the Shell program to set up automatic deletion of ALOGIT’s working files.

Version 4.3 has been available from 2012; while it is superseded by 4.5 it is still made available for users with perpetual 4.3 licences. Version 4.3 naturally includes the features introduced in versions 4.1 and 4.2 and further adds:

- the ability to read and write files in the formats used by the VISUM transportation planning software;<sup>56</sup>

---

<sup>56</sup> These formats have been changed by the manufacturer of VISUM and are therefore no longer useful in ALOGIT.

- the facility of transposing matrices, often needed in transportation applications;
- the provision of a test (using the  $\chi^2$  statistic) of the overall fit of the model, based on the tables produced in APPLY runs;
- reports from the Shell program to the user's screen give better indications of failure conditions;
- the ability to recognise 'environment' variables to allow users to use ALOGIT control files (i.e. ALO files) in different contexts with minimal changes.

### *Version 4.5*

Specific improvements in 4.5 relative to versions 4.3 and earlier are the following.

- The processing of large ALO files has been greatly speeded up. This is a substantial improvement, but has made it impossible to maintain the option of data range inputs (e.g. d1-d25). The 'special' treatment of dataXX has been dropped, so that (for example) d4 and DATA0004 are no longer equivalent. Arrays give a more logical means of achieving the data range.
- The output of 'robust' errors has been added to the capabilities of the program. This has also involved a revision of the format of output of coefficients and errors in the LOG file. Control is given to the user through \$PRINT EST.ERROR. Potentially more useful t statistics are given for hierarchy coefficients and standard errors are given for size variables, as well as the previous values.
- The standardisation of the program code has been greatly improved, giving better possibilities for maintenance of ALOGIT over the long term.
- Maximum and minimum values in GEN.STATS are now calculated without including zero values.
- The name of the random function WEIBULL has been corrected to GUMBELL.
- Input limits have been changed:
  - maximum input line to 20000 characters (was 10000)
  - the number of input strings on a line to 5000 (was 2000)
  - input one-line buffer to 200 (was 80).
- Printing alternative availability and choice is now controllable by the user as \$PRINT CHOICE.
- The facility of skipping the first record on a file has been added, using the FILE subcommand SKIP1.
- As usual, numerous minor corrections and improvements have been made, including the print of timing for the different stages of the run.

### *Status of Versions 3.8 and earlier*

Version 3.8 was current from 1995 to 2000, superseding 3.2 which was current from 1992. The program MEGALO resembles 3.8: MEGALO and 3.8 run with 'DOS extenders' to allow quite

large problems to be run, but operate only on 16-bit machines and under DOS or with early versions of Windows. In contrast, 3.2 and 3.1 (current from 1990 and quite similar to 3.8) are DOS programs and will run on most machines, but deal only with problems up to 125 coefficients and with corresponding limitations on other dimensions. The differences between versions 3.x and 4.x are described in detail above. A translation program is available to convert control files for version 3 to version 4.

Versions 1 and 2 of ALOGIT were current from 1986 to 1990 and were much more limited, with different and much less flexible control files.

## B.2 Obsolescent features

Some of the older features of ALOGIT are not consistent with the improved approaches that are now used to specifying and estimating models. While these have been retained in version 4.5 to maintain compatibility with older control files, the user is discouraged from using these. It is not guaranteed that they will be retained in future versions.

### B.2.1 *Coefficient specification*

Coefficients have previously been entered into the program using lines like

```
12 car_cost F -0.03
```

This indicates that the name `car_cost` refers to a coefficient, is not constrained, has the initial value `-0.03` and can be referred to by coefficient number 12. The replacement for this is the more intuitive

```
$coeff car_cost # -0.03
```

This revised specification gives exactly the same information, except that no coefficient number is defined. Coefficient numbers are now suppressed, as the identification of the coefficient is by its name, i.e. `car_cost`, and the number can cause confusion.

### B.2.2 *Utility and size functions*

In previous versions of the program, utility and size functions were introduced by lines like

```
Uxxx= or UTILxxx=  
Sxxx= or SIZExxx=
```

These lines can cause confusion (e.g. if the user wants to use a variable called U2) and are therefore now replaced by specifications like

```
U(xxx) = or UTIL(xxx) =  
S(xxx) = or SIZE(xxx) =
```

### B.2.3 *\$ELAST command*

The command `$ELAST` has been available, for MNL models only, to obtain point elasticity values for all alternatives with respect to a specific variable appearing in the utility function of

one alternative. However, it is clearer to obtain these values in an APPLY run, which can be used for all model types.

#### *B.2.4 ERROR\_COMP files*

ERROR\_COMP files were provided, using a subcommand of the FILE command, as a means of re-using previously created error components in a further run. This saves the time taken to calculate the random numbers and ensures that new runs are exactly consistent with the run that created the error components. However, these effects are more easily obtained by using the MODEL= subcontrol on the \$ESTIMATE control (see section 4.5). Such files were and still are named using the extension '.F14' as described in Appendix A.

#### *B.2.5 @USERINPUT*

@USERINPUT was a subcontrol of \$TITLE and \$SUBTITLE that operated in the DOS environment to allow the user to enter text into the run title or subtitle. It will probably not work in most modern Windows environments and has therefore been declared as obsolescent.

## APPENDIX C: The RU1 specification and ‘normalisation’

It has been known since the mid-1970's that MNL and tree logit models can be consistent with the principle of individual utility maximisation (abbreviated here as Umax). Consistency with Umax implies consistency with the principle that each individual chooses the alternative that appears best to him or her. There are of course other possible bases for defining models, with which logit models may or may not be consistent. The choice of Umax is a simple and intuitively appealing basis which has been used in the majority of theoretical modelling studies; it also has remarkable flexibility<sup>57</sup>. Adhering to the Umax principle and keeping all models within a system consistent with it may at first seem of mere theoretical value, however it has clear practical advantages as well.

In particular, if a model can be guaranteed to be consistent with the Umax principle, we can be certain that its behaviour will be consistent with the intuitive notions embodied in that principle. For example, if the price of an alternative is increased (and we assume that cost has a negative impact on utility!), then the demand for that alternative will decrease and the demand for all other alternatives will increase, or at least will not decrease. The same reasoning is appropriate in transport models for longer travel times resulting in lower utilities and decreased demand. Other considerations, e.g. increased comfort and higher frequency (assuming these affect the utility positively), can counter the loss of utility due to higher cost or longer travel times.

It has been known for many years<sup>58</sup> that tree logit models are consistent with Umax, for all data values, providing the nesting coefficients lie within specific ranges. The practical effect of a failure of this condition is that the change in the price of one alternative can cause the demand for other alternatives within the same ‘nest’ to change in an inappropriate direction. For instance, a price increase for an alternative would result in a decreased demand for that alternative, but it could also result in a *decreased* demand for other alternatives within the same nest. This counter-intuitive effect would not always occur, so that it would not be sufficient to make a series of tests on a model to ensure its consistency with intuition. The advantage of keeping models consistent with Umax is that it can be *guaranteed* that counter-intuitive results such as described above will not occur. In the formulation of the model used in ALOGIT, the requirement is that the nest coefficients lie between 0 and 1 (the value 0 is not acceptable, but the value 1 is OK). Other software, such as Biogeme or Apollo, may have different formulations and different numerical limits may apply.

However, a more subtle issue may be hidden in the way in which utility is passed between alternatives and nests. In the Introduction, the utility for a nest  $t_j$  to which an alternative  $j$  belongs was defined by calculating the ‘logsum’ over the alternatives which feed into nest  $t_j$ :

$$V_{t_j} = V_{t_j}^* + \theta \log \sum_{k=t_k=t_j} \exp V_k \quad (11)$$

In this equation,  $\theta$  is the nest coefficient and, as indicated in the previous paragraph, this must satisfy  $0 < \theta \leq 1$ . The formulation of equation (7) is simple to program and this simplicity

<sup>57</sup> See Hess, S., Daly, A. and Batley, R. (2018) Revisiting consistency with random utility maximisation: theory and implications for practical work, *Theory and Decision*, January.

<sup>58</sup> e.g. Daly, A.J. and S. Zachary (1978) Improved Multiple Choice Models, in D.A. Hensher and M.Q. Dalvi (eds.), *Determinants of Travel Choice*, Saxon House; a more complete theory for the ‘GEV’ family of models is given by McFadden, D. (1981) Econometric models of probabilistic choice, in Manski, C. and McFadden, D. (eds.) *Structural Analysis of Discrete Data: With Econometric Applications*. The MIT Press, Cambridge, Massachusetts.

may help in making ALOGIT fast. It may also give important flexibility in setting up systems of models.

However, most of the theoretical literature and some software uses a different formulation

$$V_{t_j} = V_{t_j}^* + \theta \log \sum_{t_k=t_j} \exp(V_k/\theta) \quad (12)$$

The effect of this formulation, which introduces some complication into the programming, is that it ensures that the scale of  $V_{t_j}$  is the same as the scale of  $V_j$ .

The formulation (11) used in ALOGIT is sometimes called RU1, whereas formulation (12) is called RU2.

In simple cases, the change of formulation has no real effect: the RU2 coefficients (i.e. the  $\beta$ s) are simply  $\theta$  times the RU1 coefficients. But when the model is more complicated and different values of  $\theta$  appear in different parts, then real differences can arise and care must be taken. The essential requirement for the RU1 specification to be fully consistent with a basic interpretation of  $U_{max}$  is that the overall scaling implied by the nesting coefficients  $\theta$  should be equal across the alternatives. The automatic scaling given by RU2 means that this equality is always achieved.

Specifically, to achieve consistency with  $U_{max}$  across the model, it is sufficient to arrange that, for every elementary alternative (i.e. alternatives that can be chosen), the product of the nest coefficients  $\theta$  between the alternative and the root should be the same. Then, variables that have a generic coefficient across the alternatives have the same impact on the consumer's utility. For example, this means that the utility impact of marginal cost changes is identical, irrespective of which alternative the additional cost is spent on: a euro is worth a euro, whatever you spend it on.

The ALOGIT formulation, using RU1, can always be adjusted to give equivalence to an RU2 formulation by adding additional nests to the model. These additional 'dummy' nests, containing just one alternative, allow nesting coefficients  $\theta$  to be added so that every alternative is affected by every nesting coefficient and the total scaling is constant over the alternatives. A simple example is given below.

However, this adjustment of adding dummy nests is not necessary in some important special cases:

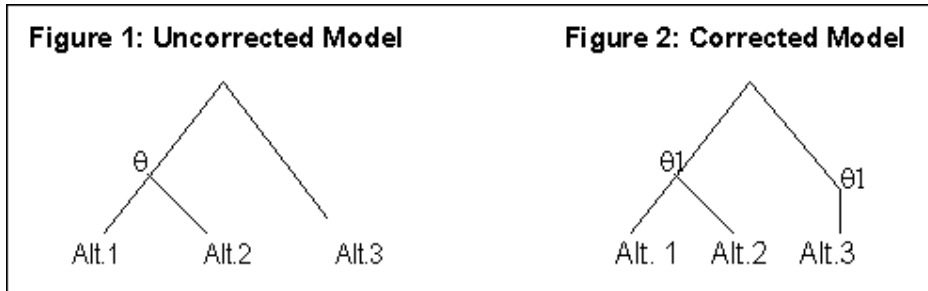
- for alternatives that are not available for the same observations, e.g. because data sets are being merged;
- when there are no generic coefficients across the alternatives;
- when alternatives that have different nesting coefficients have zero utility.

Other special cases may also be found.

*Example: adding a dummy nest in a 3-alternative model*

Figure 1 shows the simplest of all possible tree structures. Alternatives 1 and 2 form a nest, at which a logsum is calculated and multiplied by the nest coefficient  $\theta$ . But because of the RU1 calculation used in ALOGIT, if the cost of all three alternatives is increased by a constant value, the cost of the nest will be increased only by  $\theta$  times that constant and (assuming  $\theta < 1$ ) the demand for the nest alternatives will increase at the expense of alternative 3. Obviously, the

same considerations apply to any variable in the utility function, not just the cost. To counter this effect a correction is needed.



In Figure 2 the necessary correction is introduced in which alternative 3 is also included in a nest, this time containing only one alternative, but that nest is also assigned *the same* structural parameter  $\theta_1$  as is used in the nest for alternatives 1 and 2 (this will have a different value from  $\theta$  in Figure 1 when the model is estimated). Now if the cost of all the alternatives is increased by the same amount, the cost of both the nests is increased by  $\theta_1$  times that amount and the demand remains constant. Thus the consistency of the scale of utilities can be reinstated across all three of the alternatives. The additional nest introduced in Figure 2 is called a dummy nest because it contains only a single alternative and its sole function is to multiply the utility of that alternative by  $\theta_1$ .

This type of correcting dummy alternative can be introduced into more complicated trees also to maintain the required consistency. In these cases it may be necessary to define several dummy nests to multiply the utility of an alternative. To keep the complexity of the model within bounds, the dummy nests can be added at the ‘bottom’ of the model tree. Changing the model structure in this way will always allow an RU1 specification to be adjusted to give the same results as an RU2 specification.

*Conclusion*

There are two standard ways of defining the structural coefficients in a theoretical presentation of the tree logit model. The way used in ALOGIT is often called RU1 and the other specification is called RU2.

The specification used in ALOGIT is more efficient for programming, may be easier to exploit for special purposes and can always be adapted to ensure consistency throughout the tree by achieving consistency with RU2.

It is not clear that an RU2 specification can always be adapted to give consistency with RU1.

## APPENDIX D: Program Size Limits

This table indicates size limits in ALOGIT 4.5 that are not controlled in ALO4.INI (see section 9.1). These limits should have no impact on the size of problem that can be considered.

Length of file name	250
Length of file name for SCN or VAL.LABEL	80
Dimensions (e.g. for tables in APPLY)	100
Dimensions on SCN file and in DATA	10
Printed columns in TABLES	20
Output variables in APPLY tables	10
Output variables in SCN files or VAL.LABELs	4
Elasticities	50
Keys for file linking	5
Files	50
File format nesting	10
Strings on a logical line	5000
Characters on a logical line	20000
Characters on a physical line	200
Length of title/subtitle	80
Depth of DO and ...THEN structures	10
Brackets in a line	500
Arguments for a function	101
Numbered coefficients up to	9999
Numbered rows up to	99999
Numbered alternatives up to	999999
Depth of tree for printing	20
Sequence length in HALTON for prime p	$p^{31}$
Number of random terms <sup>+</sup>	10000
Log-linear terms per alternative (100 in TABLES) <sup>+</sup>	50

Note: + this limitation will be removed shortly.

## APPENDIX E: File Handling Error Messages

The table below shows the most common errors that can be generated by the operating system with respect to input and output files.

number	message	meaning
6103	invalid REAL	This message may be generated when a data file contains invalid characters such as letters or punctuation marks. Sometimes word processing packages insert codes that lead to this message.
6413	file already connected to a different unit	This message may be generated when the same file name is used for different ALOGIT files. Different file names <b>must</b> be used for all the ALOGIT files.
6414	access not allowed	This message can be caused when the file specified is actually a directory, or the file is specified as read-only when ALOGIT would have to write to it, or when there are sharing conflicts (e.g. on a network).
6416	file not found	This message is caused by the non-existence of an essential file, such as the F11 file.
6417	too many open files	The FILES= command in CONFIG.SYS must be changed to increase the number of files that can be open at any time.
6422	no space left on device	The disk is full!
6504	invalid number in input	This message may be generated when a data file contains invalid characters such as letters or punctuation marks.
6981	initial left parenthesis expected in format	The format supplied by the user for reading data must begin with a left bracket '(' and end with a right bracket ')'.
6982-6991		These are more complex format errors relating to the reading of data files.