Maximum \max()

The maximum function of a matrix yields a 1×1 matrix containing the maximum of all its elements.

Minimum \min()

The minimum function of a matrix yields a 1×1 matrix containing the minimum of all its elements.

Absolute value \abs()

The abs function replaces all matrix elements with their absolute value.

Trigonometric functions \cos(), \sin() etc.

These functions replace all matrix elements with their appropriate trigonometric transformation, in radians.

Exponent \exp()

Any matrix is a legal argument for this function which replaces each element A_{ij} by $e^{A_{ij}}$.

Natural Logarithm \ln()

Any matrix is a legal argument for this function which replaces each element A_{ij} by $\ln A_{ij}$. If an element is less than 1×10^{-30} then the result is $\ln(1 \times 10^{-31})$. Although error messages would be more normal in such a situation, this behavior can be helpful in optimization.

Square Root \sqrt()

Any matrix is a legal argument for this function which replaces each element A_{ij} by $\sqrt{A}_{ij}$. If an element is less than zero, a fatal error occurs.

Diagonal to Vector \d2v()

The leading diagonal of any matrix is placed into a column vector with $\min(r, c)$ rows, i.e. r or c, whichever is less. e.g. if

$$\mathbf{A} = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \end{pmatrix}$$

then

$$\backslash\mathrm{d2v}(\mathbf{A}) = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Vector to Diagonal Matrix \v2d()

A row or column vector is placed in the leading diagonal of a square matrix. e.g. if

$$\mathbf{E} = \begin{pmatrix} a & b & c & d \end{pmatrix}$$

then

$$\backslash\mathrm{v2d}(\mathbf{E}) = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & d \end{pmatrix}$$

Matrix to Vector \m2v

A matrix is placed in a column vector, by rows. Thus if

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

then

$$\backslash\mathrm{m2v}(\mathbf{A}) = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

This is similar to the function `\vec`; which places the matrix into a vector by columns, instead of rows.

Matrix to Vector \vec

A matrix is placed in a column vector, by columns. Thus if

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

then

$$\backslash\mathrm{vec}(\mathbf{A}) = \begin{pmatrix} a \\ c \\ b \\ d \end{pmatrix}$$

Note that it is more efficient to use `\m2v(A)` than `\vec(A')` and more efficient to use `\vec(A)` than `\m2v(A')`. Both functions work for matrices of any shape.

Matrix to Vector \vech

All the elements on the diagonal and below are placed into a vector, by columns. Thus if

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

then

$$\backslash\mathrm{vech}(\mathbf{A}) = \begin{pmatrix} a \\ c \\ d \end{pmatrix}$$

Like its counterparts `vec` and `m2v`, this function will operate on matrices of any shape, terminating at the last row or column, whichever is the lesser. Thus if

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix}$$

then

$$\backslash\mathrm{vech}(\mathbf{A}) = \begin{pmatrix} a \\ c \\ e \\ d \\ f \end{pmatrix}.$$

Standardize \stnd

This operation will convert a covariance matrix into a correlation matrix. Replacement of elements is made according to the formula:

$$\mathbf{A}^*_{\mathrm{ij}} = \frac{\mathbf{A}_{\mathrm{ij}}}{\sqrt{\mathbf{A}_{\mathrm{ii}}\mathbf{A}_{\mathrm{jj}}}}$$

The diagonal elements of $\mathbf{A}$ have to be greater than zero, and $\mathbf{A}$ has to be square.

Real Eigenvalues \eval

The real parts of the eigenvalues of a square matrix are placed in a column vector, in ascending order of size, smallest first.

Real Eigenvectors \evec

The real parts of the eigenvectors of a square matrix are placed in a square matrix, where column j contains the eigenvector corresponding to eigenvalue j, with eigenvalues sorted in ascending order of size, smallest first ($j = 1$).

Imaginary Eigenvalues \ival

The imaginary parts of the eigenvalues of a square matrix are placed in a column vector, in ascending order of size, smallest first.

Imaginary Eigenvectors \ivec

The imaginary parts of the eigenvectors of a square matrix are placed in a square matrix, where column j contains the eigenvector corresponding to eigenvalue j, with eigenvalues sorted in ascending order of size, smallest first ($j = 1$).

3.2.3 Means or M1 Command

Syntax:

```
Means/M1 A {operator B {operator C ...}} /
```

The `Means` command operates in the same way as the `Const` or `M2` command. It exists to facilitate the modelling of means. All the matrix operators and functions (Sections 3.2.1 and 3.2.2) may be used just as when specifying a model for covariances. A `/` must end the command. Currently, Mx will do nothing with models for means when applying the functions `LS, GLS, AWLS, DWLS`. Only the `ML`, `US` or `RM` fit functions make use of models for means.

3.2.4 Threshold Command

Syntax:

```
Threshold A {operator B {operator C ...}} /
```

The `Threshold` command operates in the same way as the `Means` to specify means. It exists to facilitate the modelling of thresholds when fitting to contingency table data. All the matrix operators and functions (Sections 3.4 and 3.5) may be used just as when specifying a model for covariances. A `/` must end the command. `Threshold` cannot be used with any fit function other than `CTable` (see Section 4.2.3).

Special restrictions apply to the dimensions of the matrix calculated in the `Threshold` command. The result must have 2 rows and must have at least d columns where $d = \max((r-1),(c-1))$, in other words at least one less than the number of rows or the number of columns in the contingency table, whichever is the greater. The first $(r-1)$ elements of the first row of the matrix will contain the thresholds that separate the rows. The first $(c-1)$ elements of the second row of the matrix will contain the thresholds that separate the columns. In fact, these are unstandardized row and threshold estimates, which may be standardized by dividing by the square root of the product of the two diagonal elements of the expected covariance matrix calculated by the `Covariance` or `Constraint` statement. This allows the testing of models that predict differences in variance between groups, but have equal thresholds.

The user should take care to supply starting values for thresholds that increase from left to right in both rows of the matrix calculated by the `Threshold` command. Ideal starting values are those that, when standardized, mark the $z-$scores on the normal distribution corresponding to the cumulative frequencies of the normal distribution of the row totals (first row of the calculated matrix) or the column totals (second row of the calculated matrix). For example, if the following contingency table was supplied as data:

```
CTable 3 2
20 180
40 360
20 180
```

then appropriate starting values for 2 row thresholds would be -.67 and +.67 (z-scores corresponding to the lower 25% and 75% of the normal distribution), and -1.28 would be appropriate for the starting value of the column threshold (z-score corresponding to the lower 10% of the normal distribution).

3.3 Putting Numbers in Matrices

This section describes three methods of entering numbers into matrices (see Section 3.4 for how to specify elements of matrices to be free, fixed or constrained parameters). In Section 3.1, we saw how matrices could be defined as one of 9 types, such as identity, symmetric, diagonal or full (see Table 3.1), and how their dimensions (rows, r and columns, c) were specified. On inspection of the table, we see immediately that types `ZE ID IZ` and `ZI` have no free elements at all. For example, there is nothing more to know about an `IZ` matrix which has 2 rows and 4 columns. It looks like this:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

and it *cannot be changed at all.* If it was altered, then it would no longer be an `IZ` matrix.

All 5 remaining matrix types have *modifiable* elements which may be altered with the commands `Matrix Start` and `Value`. The number of modifiable elements varies according to:

- The number of rows and columns in the matrix
- The *type* of the matrix

All modifiable elements of a matrix are initialized at zero. The order of elements in a matrix is left to right, by rows. For example, a symmetric (3×3) matrix would be read as:

```
1
2 3
4 5 6
```

See Table 3.3 for more examples on the patterning of matrices.

3.3.1 Matrix Command

Syntax:

```
Matrix <matrix name> {File=filename}
<numlist>
```

where `<numlist>` *is a free format list of numbers. Note that different syntax is required in multiple fit mode:*

Syntax:

```
Matrix <group number> <matrix name> {File=filename}
<numlist>
```

The list length required varies according to matrix type, and size as described at the start of this Section, on page 35. For example, suppose we specify a diagonal matrix **A** with 3 rows and 3 columns. The fourth column in Table 3.1 shows that the number of free elements is equal to r for diagonal elements, so we supply r elements. The command lines

```
Matrix A
.3  5  9
```

or, equivalently

```
Matrix-I-would-like-to-change-is A
0.3D+00
5  9.00000000
```

would result in matrix **A** as:

$$\begin{pmatrix} .3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

The `Matrix` will operate on elements regardless of whether or not they have been specified as fixed or free parameters.

`Matrix` will read its elements from a file with a FORTRAN format on the first line. Such files may have been produced by an earlier run of Mx, or by another program. LISREL matrix output files (produced by commands such as gamma=filename on the LISREL OU line) are fully compatible. The files must contain at least as many numbers as required to fill the changeable elements of the matrix specified (see page 35).

3.3.2 Start and Value Commands

Syntax:

```
Start/Value <value> <element list>/ All
```

where `<element list>` *consists of matrix elements (e.g. A 1 2 3) and may include the* `TO` *keyword*

Often, it is not convenient to provide a value for all the elements of a matrix, if only a small number need to be modified. Under these circumstances, it is easier to explicitly change elements by *name.* Elements of matrices may be referred to by up to three subscripts, according to the syntax

```
A {<group>} <row> <col>
```

If the matrix you wish to refer to is in the current group, the group number may be omitted. The numbers `<group> <row> <col>` may be separated by any number of non-numeric or blank characters, so that, for example, to put .5 in row 2 column 3 of group 1's A matrix, you could enter:

```
Value.5 A 1 2 3
```

will work the same as

```
Value.5 A(1,2,3)
```

or even

```
Value.5 A group1 row2 column3
```

but *please note* the *foolish command*

```
Value.5 A column3 group1 row2
```

would do something *entirely different,* placing .5 in the 2nd column of row 1 of matrix A in group 3. *N.B. It is only possible to modify matrices in the current or previous groups.*

The difference between `Start` and `Value` lies in their treatment of elements when the keywords `ALL` or `TO` are used (`-` is a synonym for `TO`). With the keyword `ALL`, `Start` assigns a starting value to every free parameter specified at that point in the input file. `Value` does the opposite – it assigns its value to every *fixed* matrix element specified up to that point. Although `Start` does the same thing if the `TO` keyword is specified, i.e. only apply its value to free parameters, `Value` behaves differently. It will assign a value to all elements in the same specified range, free parameter or fixed.

The `TO` keyword should be used only to specify a range of matrix elements *within the same matrix.*

3.4 Putting Parameters in Matrices

Parallel to the placement of numbers in matrices described in Section 3.3, there are facilities for putting parameters in matrices. Four of these commands are equivalent to the LISREL commands of the same name, and there is a new command (`Specification`) that greatly simplifies specification for models with lots of equality constraints. Note also that all modifiable elements of a matrix can be specified as different free parameters using the keyword `Free` after the matrix is specified (see Section 3.1).

3.4.1 Pattern Command

Syntax:

```
Pattern <matrix name> {File=filename}
<numlist>
```

where `<numlist>` *is a list of* `1`*'s and* `0`*'s. Note that different syntax is required in multiple fit mode:*

Syntax:

```
Pattern <group number> <matrix name> {File=filename}
<numlist>
```

The `Pattern` command is a simple method that has the same syntax as the LISREL command on which it was based. Following the `Pattern <matrix>` command, the user must provide the correct number (see `Matrix` command page 36) of `1`'s and `0`'s for that matrix. A `1` (or any non-zero value) indicates that the element is a free parameter (which may be constrained to equal another free parameter - see the `Equate` command on page 39), and a `0` indicates that the element is to be fixed.

3.4.2 Fix and Free Commands

Syntax:

```
Fix/Free <value> <element list>
```

where `<element list>` *is a list of matrix elements*

The `Fix` and `Free` commands operate directly on specific matrix elements or sets of matrix elements. `Fix` makes a parameter fixed (if it was `Free` before) and `Free` makes an element a free parameter to be estimated. Matrix elements are referred to by group, row and column as described in Section 3.3, page 37. The keywords `TO` and `ALL` (see page 37) may be used to specify ranges of matrix elements to be fixed or freed.

For example, suppose in group 1, matrix A was defined as symmetric, with 4 rows and columns. Initially it would be patterned with zeroes throughout. The command

```
Free a 1 2 1 - a 1 4 3
```

would give the following pattern:

$$\begin{pmatrix} 0 & 1 & 3 & 6 \\ 1 & 2 & 4 & 7 \\ 3 & 4 & 5 & 8 \\ 6 & 7 & 8 & 0 \end{pmatrix}$$

if this command was followed by

```
Fix A 1 4 2
```

the parameter specification would become:

$$\begin{pmatrix} 0 & 1 & 3 & 6 \\ 1 & 2 & 4 & 0 \\ 3 & 4 & 5 & 8 \\ 6 & 0 & 8 & 0 \end{pmatrix}$$

In symmetric matrices, references to the upper triangle are legal also; anything done to an element one side of the diagonal (A_{ij}) is done to the corresponding element on the other side (A_{ji}).

3.4.3 Equate Command

Syntax:

```
Equate <matrix> {<gp>} <r> <c> <matrix> {<gp>} <r> <c> ...} }
```

In order to constrain matrix elements to equal one another, the `Equate` command may be used. Its primary purpose is to specify equality constraints among parameters, but it can be used to copy a numeric value from one matrix element to another. There is a big conceptual difference between the *first* element specified in a list and the others. The fixed or free status of the first element is given to the remaining elements in the list, be they fixed or free. If the first element is a free parameter, the same parameter is copied to the other elements. If the first element is fixed, then a warning message is printed, to the effect that all other elements will be fixed. The value in the first element is then passed to the other elements in the list. The `Equate` command may be used within matrices, or across matrices in the same group, or across matrices in different groups. Note that it is *not* possible to use `Equate` to make an immovable element (such as an element of a matrix specified as type `ID`, or an off-diagonal element of a diagonal matrix) into a free parameter.

For example, given matrices specified in group 1 as follows:

```
Matrices
A Symm 3 3  FR
B Full 2 4
I Identity 6 6
```

the following `Equate` statements are legal:

```
Equate A 1 1 B(2,2) B 1 4
Equate B 2 1 A 1 2 3
Equate BANANA 2 2 APPLE 1 1 1
```

However, the following are *illegal:*

```
Equate A 1 1 I 2 2
Equate I 5 5 B 1 1
Equate A 1 2 C 1 1
Equate A 4 4 B 1 1
Equate A 2 2 G 4 1 1
```

They fail because: I is an identity matrix (1 & 2), `C` has not been specified (3), `A` does not have 4 rows and columns (4), and it is not possible to refer to an element of a matrix in a later group (5).

For large models with many constraints it is often more efficient to use the `Specification` command, or to seek repetitive structures in the model matrices and use partitioned matrices (see Section 5.2.4), or both. The kronecker product can be particularly useful when specifying repetitive partitioned matrix structures.

3.4.4 Specification Command

Syntax:

```
Specification <matrix name>
a { b {c}}
```

where `a, b` *and* `c` *are not necessarily distinct integers. Note that different syntax is required in multiple fit mode:*

Syntax:

```
Specification <group number> <matrix name>
a { b {c}}
```

Following the `Specification` command, the user supplies a list of numbers that varies in length according to the dimensions and type of the matrix (see Section 3.3). If a zero is supplied, it indicates that the element is to be fixed. Non-zero elements refer to free parameters, and the same number refers to the same parameter. For example, the command

```
Specification A
1 2 3 0 0 0
0 0 0 3 2 1
```

would be equivalent to the statements

```
Pattern A
1 1 1 0 0 0
0 0 0 1 1 1
Equate A 1 1 A 2 6
Equate A 1 2 A 2 5
Equate A 1 3 A 2 4
```

The second method becomes tedious and error-prone in large models. Users familiar with LISREL will recognize specification matrices in LISREL output, but not as input.

Note that the `Specification` and `Pattern` commands cannot be mixed in the same Mx job. This is for safety, because the opportunities for user error are too large.

3.4.5 Boundary Command

Syntax:

```
Bound low high <parlist> / ALL
```

where parlist is a list of matrix elements or a list of parameter `Specification` *numbers.*

Boundary Constraints

By default, parameters to be estimated are constrained to lie between -10000 and +10000. These limits can be increased or decreased with the `Bound` command. Boundaries may be supplied more than once for any parameter, but only the last `Bound` statement referring to a particular element is used. For example the statements

```
Boundary -1 1 all
Boundary .3 5 a 1 4 6 a 1 5 6
```

would change the limits for all parameters to -1 and +1, except those (if any) in elements `A 1 4 6` and `A 1 5 6`. The `TO` syntax may be used to specify ranges within matrices, so that

```
Boundary 0 1 X 1 2 to X 1 6
```

would make parameters in all elements between `X 1 2` and `X 1 6` lie take values between zero and one.

If the `Specification` command has been used to specify parameters in matrices, then it may be easier to refer to parameters with these numbers in a `Bound` command. Thus

```
Specification A
0 2 4 6
2 0 6 7
Boundary 0 10  2 4
```

would be permitted as a method of bounding parameters 2 and 4.

Inequality Constraints

The boundary command may be used to specify linear and non-linear inequality constraints[1]. Constraints specified in a constraint group are numbered consecutively by rows in the lower triangle of the `NInput_vars` × `NInput_vars` matrix. For example, if `NInput_vars=3` was specified in the first constraint group, it would have constraints numbered as follows:

1		
2	3	
4	5	6

Any following constraint group would be numbered in a similar fashion, but starting at 7. Normally, the upper and lower boundary for a constraint are both set at zero. These boundaries may be reset *using negative parameter numbers to refer to the constraint we wish to bound.* For example, to make the boundaries on constraint 2 zero and ten thousand, we would use the command:

```
Bound 0 10000 -2
```

While it is a little clumsy, this notation does permit say constraining parameter 2 to be greater than parameter 3. This could be achieved with the following constraint group:

```
Title constrain parameter 2 > parameter 3
Data Constraint NInput_vars=1
Matrices
a full 1 1
b full 1 1
Constraint a - b /
Specify a
2
```

[1] In order to understand this Section, familiarity with the Section 2.5.1 below is at least helpful.

```
Specify b
3
Boundary 0 10000 -1
End
```

3.5 Label Matrices and Select Variables

3.5.1 Labelling Matrices

Syntax:

```
Labels Row/Column <matrixname> <label-list>
```

Labels may be given for the row or column (or both) of any matrix that has free elements. Matrices without free elements (`ZE, ID, IZ,` and `ZI`) are never displayed so labels provided for these matrices will not appear on the output. The label-list contains labels separated by blanks or carriage returns. Labels *must not begin with a number* and be *up to 8 characters long*. More characters can be read, but Mx only regards the first eight as significant, and will only print the first eight on the output.

Labels may be given for the observed data by issuing a `Label` command before the matrices command, as described in Section 2.4.1.

3.5.2 Identification Codes

Syntax:

```
ICodes <numlist>
```

where `<numlist>` *is a number list of length* `NInput_vars`.

The `ICodes` command may be used in conjunction with the `RM` fit function (Section 4.2.2) to specify a non-standard structure of the expected covariance matrix. By default, the identification codes for the covariance matrix are `1 2 3 4`... For example, if `NInput_vars=3` then by default the expected covariance matrix has a structure like this:

	1	2	3
1	V_1		
2	C_{12}	V_2	
3	C_{13}	C_{23}	V_3

From which structure it would be possible to read data in a `VLength` file that had forms:

```
1
1 .1
1
2 .2
1
3 .3
2
1 2 .1 .2
2
1 3 .1 .3
2
2 3 .2 .3
3
1 2 3 .1 .2 .3
```

and any of these could be reordered. For example, if the following `VLenghth` data were input:

```
2
3 1 .9 .4
```

Mx would generate a covariance matrix of the form

	3	1
3	V_3	
1	C_{13}	V_1

if means are being estimated, they will also be selected appropriately, in this case selecting μ_3, μ_1 from an initial vector $(\mu_1 \mu_2 \mu_3)$

The `ICodes` command allows the default order `1 2 3`... to change, making an infinite variety of input data vectors readable. The repetition of a number is most useful for pedigrees of variable structure, for example, if the model generates the covariance between two parents and two children, data that come from families with more than two children may be handled. In this case, the `ICodes` command would be:

```
ICodes 1 2 3 3
```

And thus the covariance matrix looks like this:

	F	M	C_1	C_2
F	V_F			
M	C_{FM}	V_M		
C_1	C_{FC}	C_{MC}	V_C	
C_2	C_{FC}	C_{MC}	C_{CC}	V_C

If the following `VLength` data were read:

```
3
3 3 3 .2 .4 .6
```

then Mx would create the following covariance matrix for this data structure:

	C_1	C_2	C_3
C_1	V_C		
C_2	C_{CC}	V_C	
C_3	C_{CC}	C_{CC}	V_C

The fact that two 3's have been given allows the generation of the expected covariance matrix between any number of individuals with code 3.

Chapter 4

Setting Options for Fit Functions and Output

4.1 Options and End Commands

Syntax:

```
Options <multiple> <fitfunc> <statout> <optimpar> <write>
End
```

where `<multiple>` *starts multiple fit mode,* `<fitfunc>` *specifies the fit function,* `<statout>` *requests statistical output,* `<optimpar>` *requests optimization parameters, and* `<write>` *specifies the filenames to write matrices to files*

The `Options` and `Output` lines of Mx allow the specification of a wide variety of keywords and parameters to control the type of fit function used, the amount of output requested, file names for result matrices, and many others. The `Options` command does not signify the end of a group, so several `Options` lines may be given within any group. `Options` commands should follow `Model` or `Covariance` statements, and should not be followed by `Bounds` commands. The `Output` line may be extended beyond a single line. If the last two characters of the line are ‘ `C`’ (with no blanks after the `C`) then parameters on the next line will be accepted. This facility is not implemented for any other command line. `Output` signifies the end of a group. To end a group without specifying any options, the `End` command may be used. So, for example,

```
Options RSiduals NAG=30
End
```

would be equivalent to

```
Output RSiduals NAG=30
```

Table 4.1: Default fit functions according to the type of data that have been read. `CMatrix` - covariance matrix; `KMatrix/PMatrix` - correlation matrix; `ACov` - asymptotic covariance matrix; `AVar` - asymptotic variance matrix; `Rawdata` - Raw data; `VLength` - variable length data; `CTable` - contingency table; ML - maximum likelihood; AWLS - asymptotic weighted least squares; DWLS - diagonal weighted least squares; RM raw maximum likelihood; ML_N - maximum likelihood assuming bivariate normal liability.

Input data	Default fit function
`CMatrix` `KMatrix` or `PMatrix`	ML
`CMatrix` `KMatrix` or `PMatrix` with `ACov`	AWLS
`CMatrix` `KMatrix` or `PMatrix` with `AVar`	DWLS
`Rawdata` `VLength`	RM
`CTable`	ML_N

4.2 Fit functions: Defaults and Alternatives

The default fit function for a group is set according to the type of data that are read. For example, if covariance matrices alone are read, the default is maximum likelihood. Table 4.1 shows the default fit functions selected by Mx for a given data input. Note that the method may change between groups. If a function that does not asymptote to χ^2 (e.g. RU or RM) is used in any group, then no χ^2 probability is given at the end of optimization. In general, the default fit function is appropriate for the data supplied. Mx does not provide for altering the input data from one type to another (e.g. converting a covariance matrix to a correlation matrix). However, it is a simple task to write a separate Mx input file to make this conversion.

4.2.1 Standard Fit Functions

There are several good introductions to the properties of different fit functions (e.g Jöreskog & Sörbom, 1989; Bentler, 1989). Controversy exists about the relative merits of the different methods in the face of assumption violations (see Kaplan, 1990), and it seems wise for the user to treat this information in the same way as a white wine from the Loire (drink youngest available). Currently, maximum likelihood is showing robustness in the face of violations of the assumptions of multivariate normality. Asymptotic weighted least squares generally performs better in the presence of kurtosis, but can be at least as badly affected by skewness as ML. See however, simulation work by Rigdon and Ferguson (1991) for problems with tetrachoric correlations.

In the following sections, the calculation of the fit functions is described, where

$\mathbf{S}$ is the observed covariance matrix, $\mathbf{\Sigma}$ is the expected covariance matrix, $\mathrm{tr}(\mathbf{A})$ indicates the trace of and $|\mathbf{A}|$ indicates the determinant of matrix $\mathbf{A}$. $\mathbf{S}$ and $\mathbf{\Sigma}$ are of order p and df is one less than the sample size used to calculate $\mathbf{S}$.

LS

The unweighted least squares fit function is calculated by the formula:

$$\mathrm{LS} = \mathrm{df}\left(\frac{\mathrm{tr}(\mathbf{S}-\mathbf{\Sigma})^2}{2}\right)$$

ML

When model fitting to covariance matrices, the maximum likelihood fit function is

$$\mathrm{ML} = \mathrm{df}\left\{\ln|\mathbf{\Sigma}| - \ln|\mathbf{S}| + \mathrm{tr}\mathbf{S}\mathbf{\Sigma}^{-1}) - p\right\}$$

and this is modified when both a mean vector x and a model for the means μ (see Section 3.2.3) are supplied, this function is augmented to become

$$\mathrm{ML}_{\mathrm{C+M}} = \mathrm{df}\left\{\ln|\mathbf{\Sigma}| - \ln|\mathbf{S}| + \mathrm{tr}\mathbf{S}\mathbf{\Sigma}^{-1}) - p + (x-\mu)'\Sigma^{-1}(x-\mu) + 1\right\}$$

In order for the `ML` fit function to be calculated, $\mathbf{\Sigma}$ must be positive definite. If, during optimization, the determinant of $\mathbf{S}$ is less than 10^{-30} then Mx uses a penalty function to try to steer optimization back towards a positive definite solution. The penalty function is

$$100\left\{p^2 + 10^{33}\left(\frac{10^{-30} - |\mathbf{S}|}{|\mathbf{\Sigma}|}\right)\right\}$$

If starting values supplied begin optimization in this region, it is difficult for the optimizer to escape this high plateau, so optimization may fail. To avoid this, starting values may be revised or the LSML fit function may be used to obtain sensible starting values for ML estimation.

GLS

Generalized least squares is based on the principles of Aitken (1934-35); see also Browne (1974)

$$\mathrm{GLS} = \frac{\mathrm{tr}\left(\mathbf{I} - \mathbf{S}^{-1}\mathbf{\Sigma}\right)^2}{2}$$

AWLS

Asymptotic weighted least squares follows from work by Browne (1982, 1984) and others. Effectively, the variance covariance matrix of the observed summary statistics is used as a weight matrix $\mathbf{W}$. Formally the fit function is

$$\text{AWLS} = \sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{n}\sum_{l=1}^{k}(S_{ij}-\Sigma_{ij})\,\mathbf{W}_{ij,kl}^{-1}\,(S_{kl}-\Sigma_{kl})$$

In LISREL, and by default in Mx, if a correlation matrix (`KMatrix` or `PMatrix`) is supplied, the above formula is modified to

$$\text{AWLS}_{\text{KM}} = \sum_{i=2}^{n}\sum_{j=1}^{i-1}\sum_{k=2}^{n}\sum_{l=1}^{k-1}(S_{ij}-\Sigma_{ij})\,\mathbf{W}_{ij,kl}^{-1}\,(S_{kl}-\Sigma_{kl}) + \frac{df}{2}(1-\Sigma_{ii})^2$$

The term $\frac{df}{2}(1-\Sigma_{ii})^2$ exists to constrain the elements of the diagonal to equal one. If the model fails to meet this constraint, the fit function is inflated. Mx prints the amount of the fit function that is due to this component of model misspecification. The second term is not calculated (and does not contribute to the fit function) if the keyword `DI` appears on the `Options` line.

An alternative approach to maintaining a diagonal of ones would be to standardize the expected covariance matrix before calculating the AWLS fit function. Mx does this if the keyword `Start` appears on the `Options` line. Some care with starting values is needed here, since the solution for a model standardized in this way is necessarily *not* unique. A third approach would be to use a nonlinear constraint group (see Section 2.5.2) that specifies that the diagonal elements should equal unity.

DWLS

Diagonally weighted least squares is a simplified form of AWLS for use with large problems where the AWLS matrix becomes unmanageably large. It is a compromise with less statistical validity than AWLS and should be used with caution. `Select` does not work with DWLS to discourage its use... The fit function for DWLS is simply:

$$\text{DWLS} = \sum_{i=1}^{n}\sum_{j=1}^{i}(S_{ij}-\Sigma_{ij})\,\mathbf{W}_{ij,ij}^{-1}\,(S_{ij}-\Sigma_{ij})$$

where $\mathbf{W}$ is a diagonal matrix of variances of the observed covariances. $\sum_{i=1}^{n}\sum_{j=1}^{i}$ is replaced by $\sum_{i=2}^{n}\sum_{j=1}^{i-1}$ if a correlation matrix is supplied as data.

LSML

This fit function starts with unweighted least squares, and takes parameter estimates from the solution as starting values for maximum likelihood estimation. This method is useful to avoid having to specify starting values that generate a positive definite covariance matrix. Its disadvantage is that it consumes more computer time than would supplying appropriate start values and using ML alone.

4.2.2 Analyzing Raw Data

When we have a sample of complete multinormal data, the summary statistics of means and covariance matrices are sufficient statistics to obtain maximum likelihood estimates of parameters (see the keyword `ML` above). It is common practise to remove from analysis any subject that has missing data. However, there are occasions when missing data result in the omission of a significant amount of data from analysis. If the number of types of missing data is small, for example, if there are really two sub-populations, one that has data on 6 tests, and one that lacks data on the fourth test, then covariance matrices could be computed separately for the two populations and models fitted separately to the different groups. Mx is flexible, allowing different groups to have different numbers of input variables.

Unfortunately, this multi-group approach breaks down if the number of sub-populations is large and the sample size for each group is too small to estimate a positive definite observed covariance matrix (this happens if the number of variables exceeds the number of subjects). For this reason, a number of methods of handling incomplete data are provided in Mx. `RM` (raw maximum likelihood) is really an extension of the multigroup method described above, but calculates twice the negative log-likelihood of the data for each observation. The procedure follows the theory described by Lange *et al.,* (1976). If there are k observed variables in the set, the normal probability density function of an observed vector x_i is

$$|2\pi\mathbf{\Sigma}|^{-n/2}\exp\left\{-\frac{1}{2}(\mathbf{x}_i-\mu_i)'\mathbf{\Sigma}^{-1}(\mathbf{x}_i-\mu_i)\right\}$$

where $\mathbf{\Sigma}$ is the population covariance matrix and μ_i is the (column) vector of population means of the variables, and $|\mathbf{\Sigma}|$ and $\mathbf{\Sigma}^{-1}$ denote the determinant and inverse of the matrix $\mathbf{\Sigma}$, respectively. The fit function is thus:

$$\mathrm{RM}=-\left\{k\log(2\pi)+\log|\mathbf{\Sigma}|+(\mathbf{x}_i-\mu_i)'\mathbf{\Sigma}^{-1}(\mathbf{x}_i-\mu_i)\right\},$$

If there are incomplete data, a separate group could be constructed for each different type of data vector. This could be rather tedious for anything beyond a very few types of vector, so Mx provides a second, more general approach. The approach is to create a variable length record file (Section 2.3.3). This allows the use of the above fit function, but with a variable length observed vector x. The appropriate mean vector μ and covariance matrix $\mathbf{\Sigma}$ is automatically created by

Mx for each observation. To save on computer time, the creation of μ and $\mathbf{\Sigma}$ (and importantly $\mathbf{\Sigma^{-1}}$) is done only if a vector is different in structure from the previous vector. Therefore, considerable CPU-time saving can be obtained if sorted data are supplied to Mx.

4.2.3 Contingency Table Analysis

Mx will allow only 2-way contingency tables to be analyzed, so the covariance structure is necessarily limited to two variables. Figure 4.1 shows a contour plot of the frequency distribution of two variables, X and Y. For an r by c contingency

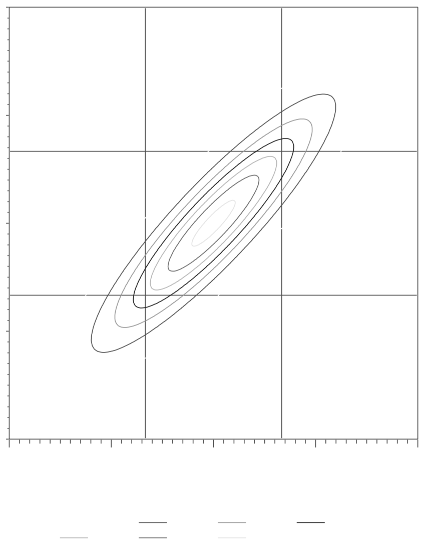

Figure 4.1: Contour plot showing a bivariate normal distribution with correlation r=.9 and two thresholds in the X and Y dimensions.

table, there are assumed to be $r-1$ row thresholds and $c-1$ column thresholds that separate the observed categories of individuals.

Twice the log-likelihood of the observed frequency data is calculated as:

$$\ln \mathrm{L_{CT}} = \sum_{i=1}^{r}\sum_{j=1}^{c} 2n_{ij} \ln\left\{\frac{n_{ij}}{n_{..}p_{ij}}\right\}$$

where n_{ij} is the observed frequency in cell ij, p_{ij} is the expected proportion in cell ij, and $n_{..}$ is the total number of observations in the contingency table. The expected proportion in each cell is calculated by numerical integration of the bivariate normal distribution, performed by subroutine BIVNOR (Schervish, 1984). For example, the expected proportion with individual 1 lying in the category between threshold a and threshold b and with individual 2 lying in the category between threshold c and threshold d would be given by:

$$L_{1,2} \quad = \quad \int_a^b \int_c^d \phi(v_1, v_2)\, dv_2 dv_1 \tag{4.1}$$

where ϕ denotes the multinormal probability density function, and v_i is the liability of individual i.

Since n_{ij} is not estimated, the number of degrees of freedom associated with an $r \times c$ table is $rc - 1$. If z cells have not been ascertained, the number of degrees of freedom is reduced by z. In order to compute an approximate χ^2 statistic, twice the likelihood of the data under the model is subtracted from the likelihood of the observed data themselves, calculated as

$$\ln \mathrm{L}_{\mathrm{CTO}} = \sum_{i=1}^{r} \sum_{j=1}^{c} 2n_{ij} \ln \left\{ \frac{n_{ij}}{n_{..}} \right\}$$

See Section 3.2.4 for details on specifying thresholds for models fitted to contingency table data.

Non-random Ascertainment

Mx will automatically calculate an ascertainment correction while calculating the likelihood of the incompletely ascertained data. For example, if we ascertain a sample of 60 probands from hospital records and examine their spouses, of whom 10 are observed to have the same disorder, then a 2×2 contingency table would be supplied as follows:

```
CTable 2 2
-1 -1
50 10
```

The `-1` in the cells of the first row indicate that subjects were not ascertained in these areas. The likelihood of the observed data must be corrected for the incomplete ascertainment of subjects for study. Effectively, as we omit certain classes of person from observation, so the likelihood of observing the remaining individuals increases. Mathematically this is expressed by dividing the likelihood by the proportion of the population remaining after ascertainment. We obtain this by subtracting the proportions in all omitted classes from the total population proportion (i.e. 1.0). In our example, assuming that individual 1 has to be above threshold, the proportion omitted is

$$\tilde{A} \quad = \quad \int_{-\infty}^{t} \int_{-\infty}^{t} \phi(v_1, v_2)\, dv_2 dv_1 + \int_{-\infty}^{t} \int_{t}^{\infty} \phi(v_1, v_2)\, dv_2 dv_1 \tag{4.2}$$

where t is the ascertainment threshold, v_1 and v_2 are the liability values of individuals 1 and 2, and ϕ is the multinormal probability density function. The likelihood corrected for ascertainment would simply be the likelihood as obtained before, but divided by $1 - \tilde{A}$

4.2.4 User-defined Fit Functions

If the `US` keyword appears on the `Options` line, the fit function for the group is to be user specified. In order for this to be the case, the matrix expression given as the model (`Constraint` or `Covariance` command) *must evaluate to a scalar.* There are no other rules. Any of the automatically defined fit functions `LS, ML, AWLS` etc. could be specified as user-defined functions, but it is generally less efficient to do so. User-defined functions are recommended only when the provided functions are not suitable. An example is given in Section 5.5

4.3 Statistical Output and Optimization Options

In this Section we discuss some of the statistics that Mx will compute automatically. While the range of these statistics is limited at present, the user should note that it is quite straightforward to compute his or her own functions of the parameters or goodness of fit statistics using calculation groups (see Section 2.5.1 for syntax and Section 5.2.4 for an example script that computes standardized estimates).

4.3.1 Appearance

Syntax:

```
Option NDecimals=n or Width=m
```

where `n` *is the number of decimal places, and* `m` *is the number of columns*

By default, Mx will print most numbers with three decimal places, or use exponential format if there are very small or very large numbers in a matrix. You may override this default with the `NDecimals` keyword, where `NDecimals=n` will print `n` decimal places of precision.

Mx prints up to 80-columns of output, which is suitable for viewing on an 80-column display or legal/letter/A4 paper (in portrait orientation) with a 10cpi font. This default may be changed with the option `Width=m` where `m` is the number of columns desired. At the present time, the `NDecimals` and `Width` parameters cannot be used together (sorry).

4.3.2 Suppressing Output

Syntax:

`Option NO_output`

Before describing ways in which MX output can be increased, we note the valuable keyword `NO_output` which prevents printing of all output for a group. This option should be used by the ecologically-minded as often as possible. Even the environmentally unconscious may find it useful to reduce their disk-space usage, but some caution should be taken not to use it too frequently since valuable information that could reveal misspecification of the model might be missed.

4.3.3 Residuals

Syntax:

`Option RSiduals`

The `RSiduals` keyword requests that the observed matrix, the expected matrix, and the residuals (observed - expected) be printed. In calculation and constraint groups, only the expected matrix is printed, since neither has any data. Note that `RSiduals` is the only way to have the observed matrix printed, and may be useful if the `Select` command has been used. When means have been supplied, the observed and expected mean vectors will be printed. Expected means also appear when using maximum likelihood with raw data. With contingency tables, Mx prints the observed and expected frequencies and their difference.

Under asymptotic weighted least squares MX prints two types of residual matrix. First, it prints the unweighted difference between the observed and expected correlations or covariances. Second, it prints a *weighted residual matrix,* which is calculated from the formula:

$$\mathrm{WRes}_{ij} = \sum_{k=2}^{n}\sum_{l=1}^{k-1}\left(S_{ij} - \Sigma_{ij}\right)\mathbf{W}^{-1}_{ij,kl}\left(S_{kl} - \Sigma_{kl}\right) + \frac{df}{2}\left(1 - \Sigma_{ii}\right)^2$$

(see Section 4.2.1 on page 4.2.1). The sum of these elements gives the fit function for the group (neglecting any penalty function for the diagonal of a correlation matrix). Note that not all elements will be positive, but that their sum is necessarily non-negative. Quite often, inspection of the weighted residuals will give a clearer idea of the cause of model failure than consideration of the unweighted residuals alone.

4.3.4 Adjusting Degrees of Freedom

Syntax:

`Option DFreedom=n`

where `n` *is the adjusted number of degrees of freedom*

If a correlation matrix is read instead of a covariance matrix, the number of statistics provided is usually less than when variances are also given. The amount of the reduction in information depends on the structure of the data. For example, if MZ and DZ twins have been measured on one variable, there are four statistics that are necessarily equal (the variances of twin 1 and twin 2 in the MZ and DZ groups). Only one of these statistics confers any information (it scales the size of the MZ and DZ covariances), so three degrees of freedom are lost, and `DF=-3` should be placed on the `Options` line. For multivariate twin data, `DF=-3k` should be used, where 3k is three times the number of variables on which each twin is measured. We can extend this idea to m groups of pedigrees of size n, each measured on k variables, in which case `DF=` $-k(mn-1)$ should be used.

4.3.5 Power Calculations

Syntax:

```
Option Power=alpha,df
```

where `alpha` *is the probability level of the test, and* `df` *are the degrees of freedom associated with it*

Power calculations are useful in a wide variety of contexts, especially experimental design and getting grants. For theoretical work, once one has established that it is possible in principle to detect an effect, a natural question is 'what are the chances of finding it with a sample of x many subjects?' See Neale & Cardon (1992) for a blow-by-blow account of how power calculations are performed using LISREL. They are much easier to perform using MX; however, we would urge that users do not use the methods blindly — a little knowledge is a dangerous thing — but instead buy a copy of Neale & Cardon! If the function values computed by MX may be considered as a χ^2, then the `Power` command will compute the power of the study to reject the hypothesis at significance level α (`alpha`) for the given number of degrees of freedom (`df`). In addition, the program very kindly works out the total sample size that would be required, given the current proportion of subjects in each group, to reject the hypothesis at various power levels from .25 to .99. See Section 5.2.2 for an example application of this method in the context of the classical twin study.

4.3.6 Confidence Intervals

Syntax:

```
Option CInterval=n
```

where `n` *is the desired percentage of the confidence interval; e.g.* `CI=90` *will give 90% confidence intervals*

In order to have some idea of magnitude of the difference in fit between two competing models, it is useful to have confidence limits on their χ^2 statistics. These are computed using the inverse non-central χ^2 distribution, in the following way. MX searches for the χ^2 (with degrees of freedom equal to those of the model) whose non-central χ^2 equals the observed χ^2 goodness of fit of the model. This procedure is done twice, to find the lower and upper limits. An example will make this clearer. If we requested the 90% confidence limits of a χ^2 of 10 with 5df, then the lower limit would be calculated as the χ^2 whose 95% limit was 10; the upper limit would be computed as the χ^2 whose 5% limit was 10. Note that these two statistics do *not* always bound the χ^2 statistic itself, i.e. an observed χ^2 may lie outside its own confidence intervals. See (Browne and Cudeck, 1990).

4.3.7 Standard Errors

Syntax:

```
Option SErrors
```

In order to retain flexibility in model specification, there are certain costs to be incurred. One of these costs is that explicit differentiation of the fit function with respect to the parameters is not feasible. In MX standard errors are computed using numerical methods (NAG routine E04XAF). Normally this routine fails to get central difference and forward difference approximations to the Hessian to agree, because the gradient is very close to zero. The technical output of routine E04XAF is automatically printed in the file NAGDUMP.OUT if NAG=n where n is greater than zero, is used on the `Options` line. MX warns the user that the errors are approximate. In the output file they are printed after each matrix that contains free parameters. Definitely $\beta-$test at present. Note that it is possible to obtain more strictly correct standard errors when the second derivatives are known. For example, using variants of the LISREL model, the second derivatives of component matrices are known (see e.g., Jöreskog, 1973) and could be calculated after optimization by adding a few calculation groups at the end.

4.3.8 Randomizing Starting Values

Syntax:

```
Option THard=n
```

where `n` *is a positive or a negative integer*

If the parameter `THard` is set using `TH=n` where `n` is a positive integer, MX will generate random starting values for all parameters and attempt to fit the model again. This attempt is likely to fail if no bounds are specified, because the default boundaries are -10000 and +10000, and the random values will be random from within these bounds. Most optimizers fail if starting values are too far away from the final solution; MX has shown greater tolerance than LISREL in this respect.

`THard` can be very useful when exploring the identification of structural equation models. If data are generated with particular fixed values for the variable parameters (Θ_1), then optimization from a different set of starting values (Θ_2) should give a solution of the original values (Θ_1). This can be tested a number of times using `THard`. If sensible bounds are not given for the parameters, this test will likely fail because Θ_1 will not be recovered. The key to underidentification is finding a solution that fits perfectly, but with a parameter vector other than Θ_1. If this is the case, the hypothesis that the model is identified has been *falsified*. Finding a number of solutions at Θ_1 does not prove identification of the model, it merely increases the support for the hypothesis that it is identified. Of course, this method does not show that the model has been specified correctly; a completely misspecified model may be identified.

The `THard` parameter may be used with a negative integer to request repeat fits from the solution, resetting the covariance matrix to an identity matrix (see Section 4.3.10).

4.3.9 Jiggling Parameter Starting Values

Syntax:

```
Option Jiggle
```

Similar to the randomizing option described above, parameter start values can be jiggled. Jiggling replaces each parameter start value x_i with $x_i + .1(x_i + .5)$. This option can be useful to nudge Mx away from a saddle point which can be troublesome when using numerically estimated derivatives (Mx does this by default). An example of a common saddlepoint is when parameters are started at or very near to zero, and the estimates x and $-x$ have the same effect on the function value. Such situations are very common in structural equation models which feature quadratic forms in their expectations; the ACE genetic model is one such example.

When used in conjunction with a negative `THard` parameter, jiggling will occur each time the refit is attempted. This may cause estimates to drift from their initial values, especially if the parameter concerned has no effect on the fit function.

4.3.10 Changing Optimization Parameters

Mx uses NAG's routine E04UCF to perform numerical optimization in the presence of general linear, non-linear and boundary constraints. Certain versions of the code use the essentially equivalent routine NPSOL, obtained from Walter Murray and Philip Gill at Stanford University in place of E04UCF. Mx does its best to choose suitable values for the parameters that control optimization, taking into account the number of parameters to be estimated, the numerical precision of the function value, and so on. However, the enormous variety of types of optimization tasks that can in principle be requested with Mx means that the automatic selection of these parameters is not always ideal. In addition, difficulties in optimization may require examination of the optional output that E04UCF can generate. Mx allows the user to print these data and to alter the parameters as needed. There are also facilities for automatically performing some of the solutions of optimization difficulties suggested in the NAG manual.

In general, parameters have been set for NAG on the cautious side, so that many of the warning messages about NAG's IFAIL parameter being non-zero are spurious. This seems better than being misled by the program ending without a noise, but giving the wrong answer. Of least concern are IFAIL's of 1 (code GREEN). Next easiest to deal with are IFAIL=4 (code BLUE - it ran out of breath), for which the `IT=n` may be used. IFAIL=3 (code RED - bad news) occurs in connection with constraints, normally they have been misspecified. Printing the residuals in the constraint groups often helps. IFAIL=6 (also code RED - take note) is the most tricky. Sometimes it occurs because of ill-conditioning of the Hessian, which can be verified by examination of the NAG output file (see Section 4.3.10). A solution here may be to use the `TH=-n` which requests optimization from the current 'solution' with the Hessian reset to the identity matrix, `n` times. On other occasions, it may appear because of insufficient numerical precision, yet still be at the right place. If a parameter in your model is not identified, IFAIL=6 is quite likely.

Appropriate choice of starting values is always recommended. Many users start parameters at zero because this is the default value of free matrix elements. In practice, Mx attempts to avoid so doing in two ways. First any parameter in the range -.01 to .01 is started at .01, and second, the JI option may be used to further nudge the parameter value away from a possible saddle point at zero (see Section 4.3.9).

Setting NAG Parameters

The defaults given in this section are the defaults as set by Mx, not the defaults suggested by NAG.

NAG=n

If this statement appears on the `Options` line, the technical output from NAG is printed in a file called `NAGDUMP.OUT`. The value `n` controls the Major Print Level, the higher the number the more verbose the output file. Minimum output is written with `NAG=1` and maximum is written with `NAG=30`. The only printing that Mx does to this file is an initial value of the function. Mx rescales all functions to 1.0 to assist general optimization, so the function value printed by NAG is a proportion of this initial value.

DE

Sets all optimization parameters at the default values given in the NAG manual.

FEAS=n

Will control the Nonlinear Feasibility Tolerance, i.e. `FEAS=r` defines "the maximum acceptable absolute violations in linear and nonlinear constraints at a 'feasible' point; i.e., a linear constraint is considered satisfied if its violation does not exceed `r`" (NAG, 1990). It has no effect if there are no nonlinear constraints. Default = .00001

IT=n

Controls the number of major iterations. Should be increased if IFAIL=4 error messages occur. Default = 100.

LI=r

Linesearch tolerance. Default = .9 if no non-linear constraints are present, otherwise, Default = .3

OP=r

Sets the optimality tolerance, a parameter controlling the accuracy of the solution. Default = $1^{-(n+6)}$ where n is approximately $\log(F_1)$ where F_1 is the function value at the starting values.

STE=r

Infinite step size. Default = 10000.

FU=r

Specifies function precision. In general this should be set at a lower value than the required precision of the solution. Default = $1^{-(n+8)}$ where n is approximately $\log(F_1)$ where F_1 is the function value at the starting values.

Obtaining Extensive Debug Output: DB=1

If the parameter `DB=1` is set on the output line, together with `NAG=n` where `n` is greater than zero, additional information will be written to the NAGDUMP.OUT file. For each evaluation of the function to obtain the gradient of the parameter vector, the fit function value for each group, the total fit function, and the values of all the parameters are printed. On each evaluation of the function to obtain the constraint functions, the values of all the parameters and the constraint functions are