Mx

Statistical Modeling

Michael C. Neale

Mx: Statistical Modeling

by

Michael C. Neale
Department of Psychiatry
Medical College of Virginia
Virginia Commonwealth University
Richmond, Virginia, U.S.A.

Medical College of Virginia, Virginia Commonwealth University
Department of Psychiatry

First Published 1991
Second Edition 1994

Refer to Mx manual as:

Neale, M.C. (1994). *Mx: Statistical Modeling.* Box 710 MCV, Richmond, VA 23298: Department of Psychiatry. 2nd Edition.

Contents

List of Tables

List of Figures

Preface

The development of Mx owes much to LISREL and I acknowledge the pioneering effort put in by Karl Jöreskog and Dag Sörbom over the years. There are many who have supported and encouraged this effort in many different ways. I thank them all, and especially Lindon Eaves, Ken Kendler and John Hewitt since they also provided grant support[1] , and David Fulker for allowing modification of his notes on matrix algebra to be supplied as an appendix for this document. Jack McArdle and Steve Boker provided excellent path diagram drawing software, and Luther Atkinson suggested the binary file save option. Health Sciences Computing at the Medical College of Virginia, Virginia Commonwealth University provided excellent software, hardware and consultancy support. My family have had to put up with my working unreasonably long hours to put this package together, and I would like to thank them for their patience.

A number of users have been most helpful finding errors in the documentation or software or both, and for suggesting new features that would make Mx easier to use. I hope that all users will forward any comments, bug reports, or wish-lists to me. My current address is:

address	Department of Psychiatry
	Box 710 MCV
	Richmond VA 23298-0710, USA
phone	804 786 8590
fax	804 371 6577
email	neale@ruby.vcu.edu *(internet)*
	neale@vcuvax *(bitnet)*

and my order of preference for communication is email, fax, phone and snail mail. For problems encountered running Mx, email is especially useful when problem files are included. You can grab the code for a variety of operating systems via anonymous ftp to sapphire.gen.vcu.edu. Please have the courtesy (and self-interest)

[1]The author was supported by ADAMHA grants MH-40828, MH-45268, AG-04954 and MH-45127

to email me to let me know what you grab so that I can keep you informed of updates, bug reports etc.

The bad news is that this manual is quite long; the good news is that you don't need to read it all. There is a very brief background to structural equation modelling in Chapter 1. The "how to" part of the manual starts in Chapter 2.2, in which general syntax conventions and job structure are laid out, followed by description of the commands necessary to read data. Chapter 3 deals with the heart of Mx: how to define matrices and matrix algebra formulae for model-fitting, and ways of estimating and constraining parameters. Methods of changing the default fit-function, of decreasing and increasing the quantity (and quality) of the output, and for fitting sub-models efficiently, are described in Chapter 4.1. The last chapter supplies and briefly describes a number of example scripts. The Appendices describe the use of Mx under different operating systems, error codes, and some introductory matrix algebra.

Chapter 1

Introduction

1.1 Background to the Software

For many years, the analysis of human twin and family data has been hampered by the lack of good, general software that can be taught and used with ease. Mx has been written to fill some of this gap. In principle, any analysis can be done if the user is prepared to learn a programming language such as FORTRAN or C, and write programs as required to fit models, compute χ^2 and so on. From years of experience I know that this is not very efficient; programming is subject to subtle errors and is time-consuming. The philosophy of Mx is to protect the user from these errors, while at the same time providing the greatest ease and flexibility in model specification. The Mx command language has been written to maintain some consistency with LISREL (Jöreskog & Sörbom 1989) whose users should find Mx especially easy to learn.

The currently available Structural Equation Modelling (SEM) packages such as LISREL (Jöreskog & Sörbom, 1989), EQS (Bentler, 1989) and COSAN (Fraser, 1988) were written with one group in mind. Only after some time was the concept of multiple groups added to these programs, and they remain rudimentary in this respect. Multiple groups are the "stuff of life" of the analysis of twin and family data, so Mx was explicitly designed to make working with many groups as easy as possible. Other aims were to generalize the specification of multivariate models for data from relatives, and to allow boundary, linear and non-linear equality and inequality constraints.

Mx is not simply another SEM package. Of course it can fit any structural equation model by the currently popular methods, but it can do a number of other things too. It is possible to 'play' with matrix algebra, without fitting models, for the purposes of education or research. Models for genetic linkage can be fitted. Raw data with missing values can be analyzed by maximum likelihood. No doubt many other data analysis tasks can be performed; Mx is *flexible.*

Mx obtains its flexibility by allowing the user to specify matrix expressions. Effectively, a matrix algebra interpreter has been tied to state-of-the-art software for numerical optimization (NAG, 1990). The interpreter has been designed so that matrix formulae are entered much the same way as they would be written. This brings conceptual model development and data analysis much closer together.

1.2 Translating Path Diagrams into Matrices

Any model for covariance structure could in principle be translated into an infinite number of diagrams, or have an infinite number of matrix representations. For example, Figure 1.1 shows four possible ways in which the variance of a single variable could be modelled. The first estimates the variance directly, and the second

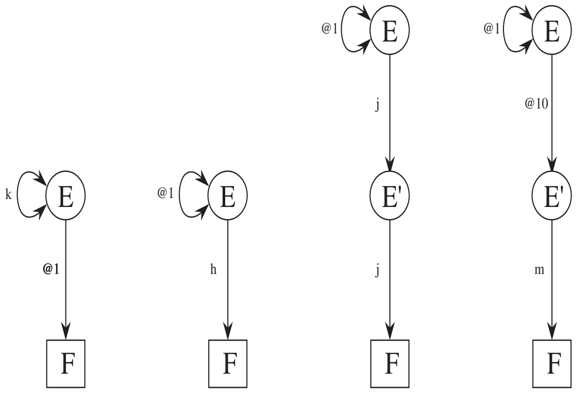


Figure 1.1: Four of the infinite variety of models for the variance of a single variable

shows a model for the square root of the variance (the standard deviation). The third and fourth estimate the fourth root of the variance and one-hundredth of the variance respectively. Using the latter two is definitely questionable, since much complexity has been added to the model with no increase in concepts. Choosing between the first two is more difficult. We note first that the second model implicitly constrains the variance to be positive. This constraint is advantageous for a model of a variance since an observed variance can never be negative. If we fit a model involving several different sources of variance, it can be important to keep them from becoming negative, even if when summed they result in a positive variance. Thus for most models, we select the second approach; in the analysis of genetically informative data it yields an intuitive model which generalizes easily to multivariate data analysis. The disadvantage of this approach is that the standard error of the path coefficient is less valid as a measure of significance (Neale *et al*, 1989).

For many people, path diagrams provide an intuitive summary of the model that they wish to test. Others prefer structural equations explicitly stating which variables are linear functions of which other variables. Still others prefer to use matrix algebra. All methods, if used properly, will yield precise mathematical expressions of a model. To make optimum use of Mx we must translate the models into matrix algebra in an accurate (and hopefully elegant) fashion. The step from matrix equation to a functional model fitting program is relatively small, and much insight can be gained from the matrix formulation.

To translate a model from a path diagram we must know a number of things about the variables. Path analysis has been justly criticized for being used insufficiently explicitly (Karlin et al, 1983). Many users implicitly assume that all variables, latent as well as observed, are standardized to unit variance. Others do not, but fail to define the parameters that give the latent variables their variance. Here we follow the method of McArdle (1980, 1984) and McArdle and Boker (1990). First we discriminate between latent and observed variables by drawing boxes around observed variables and circles around latent variables. This is not a crucial distinction, the same covariance structure may be derived with or without the circles and boxes, but it adds considerably to the ease with which a model can be 'seen' by the reader. Second, we separate variables into two sets: those that are caused by other variables, and those that are not. If a variable does not have a unidirectional arrow pointing at it, we call it an *external* variable (many other terms are common for this class of variable, such as independent, exogenous, and source, but we will use external). On the other hand, if a variable is caused by another variable we call it an *internal variable* (other terms include dependent, endogenous, and target). These two sets are mutually exclusive; a variable cannot be both external and internal. External variables may correlate (via a two-headed arrow) with other external variables. They may not correlate in this fashion with internal variables. The only relationship permitted between an external variable and an internal variable is that the external variable causes the internal one, shown in a diagram as a single headed arrow going from the external to the internal variable.

Assuming that we are not drawing a path diagram of every event in the universe since the big bang, we must allow for variables that are not shown in the diagram, but which contribute variance to one or more variables in the diagram. We have already dealt with the possibility that external variables covary due to these non-depicted variables, by allowing for double-headed arrows between them. It remains to specify the *variance* of external variables. One of the most concise methods of doing this is to draw a double-headed arrow from a variable to itself. The value of such a path is the residual variance of that variable. Both external and internal variables may have these residual paths; if they are absent, they are assumed to be zero. This formulation agrees with the McArdle and Boker (1990) RAMPATH path diagram software conventions. Output from Mx can be used with RAMPATH to visually display models, trace paths leading to variance and covariance of latent and observed variables, and to print publication quality path diagrams.

Path diagrams in this document follow two conventions:

1. External variables without explicit residuals have a variance of unity

2. Internal variables do not have any residual variance unless explicitly shown.

In certain circumstances, it is useful to partition external variables by placing them in different matrices. Although this can always be avoided, it can lead to increased conceptual clarity and numerical efficiency. Two external variables (either latent or observed) may be placed in different matrices if:

1. They do not correlate with each other

2. They do not both correlate with any third external variable.

A good example of this practice is the multivariate ACE model for twin data described in Section 5.2.4, in which genetic, shared environment and random environment components are estimated in separate matrices. Distinction between variables in this fashion may simplify requests for modification indices, or the fixing and freeing of parameters.

If there are no direct causal relationships between internal variables, the structural model may be specified with an $\mathbf{ABA}'$ formulation. The $\mathbf{B}$ matrix will contain the covariances among the external variables and the $\mathbf{A}$ matrix will have elements A_{ij} that represent paths from external variable j to internal variable j. If there are other variables 'lower' in the diagram, i.e. caused by the internal variables, they may be represented in another matrix and the $\mathbf{ABA}'$ formulation may be expanded to $\mathbf{CABA}'\mathbf{C}'$. This is only the case if a) the external variables are not causes of the second level of internal variables, and b) the second level of internal variables does not cause any variables at the second or first level. This approach may be continued through any number of levels of variables.

However, if an internal variable causes another internal variable at the same level or higher, then a different formulation is required. Both of these variables will need to be specified in the same matrix, and an $(\mathbf{I}-\mathbf{G})^{-1}$ formulation is needed to specify these effects. The rationale behind this approach is quite simple and general.

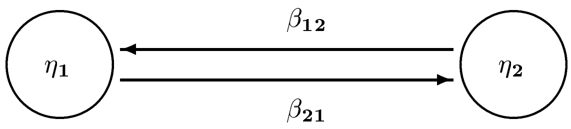
Consider the Path Diagram in Figure 1.2. This shows a feedback loop between two internal variables, which we call η variables. The total variance of η_1 and η_2 is the sum of the infinite geometric series:

$$\beta_{21}\beta_{12} + \beta_{21}^2\beta_{12}^2 + \beta_{21}^3\beta_{12}^3 \ldots$$

It is simple to show that if $|\beta_{21}\beta_{12}| < 1$ then the series converges. Let the sum of the series of n terms be called S_n, and let $\beta_{12}\beta_{21} = r$. Then

$$\begin{aligned} S_n &= r + r^2 + r^3 + \ldots r^n \\ rS_n &= \phantom{r + {}} r^2 + r^3 + \ldots r^n + r^{n+1} \end{aligned}$$

Figure 1.2: Feedback loop between two variables, η_1 and η_2.



Thus the difference between these two equations is:

$$(1-r)S_n = r - r^{n+1}$$

and so

$$S_n = \frac{r - r^{n+1}}{1-r}$$

which as n gets large tends to

$$\begin{aligned} S_n &= \frac{r}{1-r} \\ &= \frac{1}{1-r} - \frac{1-r}{1-r} \\ &= \frac{1}{1-r} - 1 \\ &= \frac{1}{1-\beta_{12}\beta_{21}} - 1 \end{aligned}$$

This formula generalizes to the case of a matrix of such effects between more than a pair of variables. In general (see LISREL 7 manual, Jöreskog & Sörbom, 1989, p. 34)

$$\mathbf{B} + \mathbf{B^2} + \mathbf{B^3} + \ldots = (\mathbf{I} - \mathbf{B})^{-\mathbf{1}} - \mathbf{I}.$$

Another way to see this formulation is directly from the structural equations. Figure 1.3 shows a multivariate path diagram of the LISREL structural equation model, where η variables are caused by two sets of independent variables, ξ and ζ.

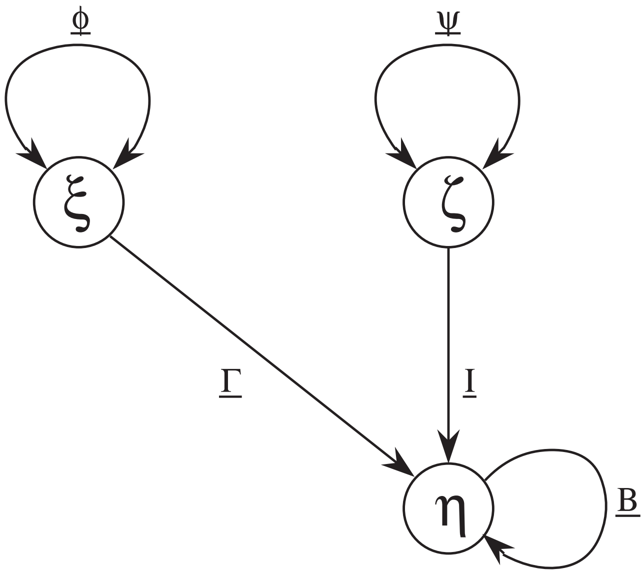


Figure 1.3: LISREL structural equation model for η variables

In addition, the η variables may cause each other, hence the unidirectional arrow from η to itself. Algebraically, the model for the η variables is:

$$\eta = \mathbf{B}\eta + \mathbf{\Gamma}\xi + \zeta$$

Here η appears on both sides of the equation, and we want it solely in terms of the other variables in the model. Hence

$$\begin{aligned}
\eta - \mathbf{B}\eta &= \mathbf{\Gamma}\xi + \zeta \\
(\mathbf{I} - \mathbf{B})\eta &= \mathbf{\Gamma}\xi + \zeta \\
(\mathbf{I} - \mathbf{B})^{-1}(\mathbf{I} - \mathbf{B})\eta &= (\mathbf{I} - \mathbf{B})^{-1}\mathbf{\Gamma}\xi + (\mathbf{I} - \mathbf{B})^{-1}\zeta \\
\eta &= (\mathbf{I} - \mathbf{B})^{-1}(\mathbf{\Gamma}\xi + \zeta)
\end{aligned}$$

This is the so-called Structural Equation Model in the LISREL manual (p.4) and together with the measurement models for $\mathbf{x}$ and $\mathbf{y}$, defines the entire LISREL model:

$$\begin{aligned}
\eta &= \mathbf{B}\eta + \mathbf{\Gamma}\xi + \zeta \\
\mathbf{y} &= \mathbf{\Lambda_y}\eta + \epsilon \\
\mathbf{x} &= \mathbf{\Lambda_x}\xi + \delta
\end{aligned}$$

In general, this model could be specified in Mx but to do so would reduce the efficiency and elegance with which models may be defined using general matrix formulae.

Chapter 2

Outline of Mx scripts and Data Input

2.1 Preparing Input Scripts

2.1.1 Commands, Numeric Input and Comments

Input files should be prepared with the text editor of your choice. If you use a wordprocessor (such as Wordperfect or Wordstar) the input file should be saved in DOS text (ASCII) format.

You may put comments anywhere in your input file using the character '!'. The Mx command processor ignores:

- All characters following `!` on any line
- Blank lines
- Anything after column 120

The processor is also entirely insensitive to case, except for filenames under UNIX. Basically, Mx reads two things: *keywords* and *numbers*. Unless explicitly stated otherwise, the first two letters of a keyword are sufficient to identify it. Keywords are separated by one or more blank spaces. Once the program has identified a keyword you can extend it to anything you like *as long as it doesn't have a blank character in it*, so `Data` and `Data_silly_words_` have the same effect.

Quite often, a keyword has the format `KEY=123` where `123` is a numeric value to be input. This is called a *parameter*. Mx ignores all (*including blanks*) non-numeric characters found between recognition of a parameter and reading a number, so that `NI=100` and `NInput_vars a lot of words 100` have the same effect.

2.1.2 Syntax Conventions

The syntax described for commands follows these conventions:

- alternatives are represented by `/`
- optional parameters or keywords are enclosed by { and }
- items to be substituted according to the specific application are enclosed by `<` and `>`

2.1.3 Job Structure

MX has been written for multiple groups, since genetically informative data generally comprise information on different types of relatives which form distinct groups. In the first group of an MX input file, you have to say how many groups there are (this is so that several MX programs can be run in one file). A group begins with a title line that contains from 1 to 120 characters for reference. The second line is the Data line, and the group ends with an Output line. What happens in between varies according to what *type* of group it is. Currently there are 3 types:

- DATA - containing data to be analysed (default).
- CALCULATION - allowing matrix operations for output or to simplify structure.
- CONSTRAINT - expressing linear and non-linear constraints between the parameters.

Any number of each type of group can be specified, in any order. Unless one of the keywords `Const` or `Calc` appears on the data line, MX expects to read a `Data` group. Effectively, there are 3 things to do:

- Supply the data
- Describe the model
- Request options

To do this, the input script will have the following structure:

1. User's TITLE
2. DATA: indicate group type: data/calculation/constraint
3. Read, select any observed data, labels
4. MATRICES: define at least one matrix
5. MODEL: define matrix formula: covariance/means/threshold/compute

6. Specify numbers and parameters, starting values, equality constraints
7. Request fit functions, statistical output and optimization options, multiple fit mode, save matrices and job specification
8. END command

Steps 1-3 supply data and are described in Section 2.2, 4-6 define the model (Section 3), and 7-8 requests output (Section 4.1). Constraint and calculation groups do not read any data, so they omit step 3.

2.1.4 Single Group Example

For example, an input file may look like this:

```
Simple MX example file
Data NGroups=1 NObservations=150 NInput_variables=2
CMatrix
1.2
.8 1.3
Matrices
A Full 2 1
D Diag 2 2
Covariance_model A*A' + D /
Specification A
1 2
Specification D
0 3
End
```

This would fit, by maximum likelihood (the default) a factor model to a covariance matrix calculated from 150 observations of two variables. The model is shown as a path diagram in Figure 2.1. Details of this example will be found in the following sections.

2.2 Data Groups

Every group has to begin with a Title line and a Data line. Optionally, these may be followed by reading of data. These commands are described in this section.

2.2.1 Title Line

The title line is purely for the user's reference, it is printed when MX prints the parameter specifications and the parameter estimates for a group. It is most useful when there are multiple groups. The title line is recognized by its location rather than a keyword at the start of a line.

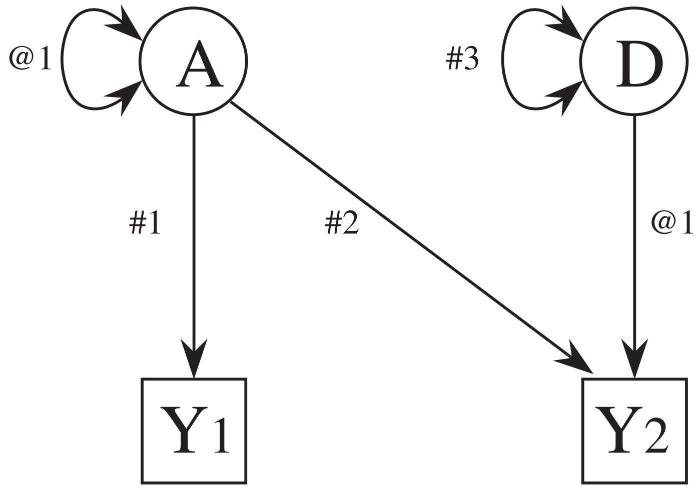


Figure 2.1: Free parameters are indicated by # and fixed parameters are indicated by @. Causal paths are shown as single headed arrows and correlational paths are shown as double-headed arrows.

2.2.2 Data Line

Syntax:

```
Data {Calc/Const} {NGroups=n NInput_vars=n NObservations=n}
```

where `Calc` *defines a calculation group and* `Const` *a constraint group, the default being a* `Data` *group*

Every group must have a data line. It has a number of parameters to indicate

i. if it is the first group, `NGroups`, the number of groups

ii. what kind of group is being input, and

iii. various characteristics (the number of input variables `NInput_vars` and the number of observations `NObservations`) of the data to be analyzed, if any .

The parameters may be specified in any order, and are summarized in Table 2.1. Note that `Data` groups must have `NInput_vars` and `NObservations` keywords. `Constraint` groups only require `NInput_vars`, and `Calculation` groups need no parameters except `NGroups` if it is the first group.

Table 2.1: Parameters of the Data line in Mx input files.

Parameter	Function	Required for group(s)
`Calc`	Specifies a calculation group	Calculation
`Const`	Specifies a constraint group	Constraint
`NGroups=` n	Number of groups	First group
`NInput_vars=` n	Number of input variables	Data, constraint
`NObservations=` n	Number of observations	Data

2.3 Commands for Reading Data

2.3.1 Covariance and Correlation Matrices

Syntax:

```
CMatrix/KMatrix/PMatrix {Full} {File=filename}
```

In a data group, a covariance matrix may be read using the keyword `CMatrix`. By default, `CMatrix` expects to read *the lower triangle* of an `NInput_vars` × `NInput_vars` matrix, from the input file. If the keyword `Full` appears, then a full matrix will be read. The matrix is read in free format, that is, the numbers are expected to be separated by one or more blank spaces or carriage returns. If the keyword `File` appears, then Mx will read the data from a file. This latter method is generally to be preferred, since it keeps the data in one place. If the data are changed, it is not necessary to change every file that uses these data. A FORTRAN format [in parentheses, e.g., `(6F10.5)`] for reading data can be read as the first line of a data file.

Correlation matrices (`KMatrix`) and matrices of polychoric and polyserial correlations (`PMatrix`) are read in the same way as covariance matrices (`CMatrix`). Although the diagonal elements of these matrices are all 1.0, and could in principle be omitted, Mx *insists on reading them.*

2.3.2 Asymptotic Variances and Covariances

Syntax:

```
ACov/AVar/AInv {File=filename}
```

In order to use asymptotic weighted least squares (AWLS see Section 4.2.1) or diagonally weighted least squares (DWLS, see Section 4.2.1) it is necessary to read a weight matrix. For compatability with LISREL and PRELIS (Jöreskog & Sörbom, 1986; 1993), Mx expects to receive a weight matrix multiplied by the number of observations. If the `File=` option is used, a PRELIS output file (created with the `SA=filename` or the `SV=filename` PRELIS commands) may be read. By default, Mx expects to receive an asymptotic weight matrix (`ACov`) whose size depends on (i) `NInput_vars` and (ii) whether a correlation matrix or covariance matrix has been input. If `NInput_vars`$=k$, then if `CMatrix` has been input, the number of rows in `ACov` is

$$p = k(k+1)/2$$

or if `PMatrix` or `KMatrix` have been input, the number of rows in `ACov` is

$$q = k(k-1)/2.$$

If you use PRELIS, please be sure to use PRELIS 2 instead of PRELIS 1, as the off-diagonal elements of the asymptotic weight matrices produced by PRELIS 1 appear quite inaccurate at the time of writing (PRELIS 1.20 and earlier). The weight matrices can thus be very large,

$$p(p+1)/2$$

or

$$q(q+1)/2.$$

An `ACov` line makes AWLS the default method of estimation for that group. If AWLS is requested on the `Options` line in a group without an `ACov`, and error will result. Similarly, DWLS is default if `AVar` is read.

Note that inverting the asymptotic covariance matrix can take an appreciable amount of time for large problems. Two facilities are available to combat this problem. First, the inverse of the matrix can be read instead. A simple Mx job could be used to invert and save the inverse, for example:

```
Commands to invert a 325x325 asymptotic weight matrix
Data Calculate NGroups=1
Matrices
P Symm 325 325
Compute  P~ /
Matrix P File=weight.asy
Output MX%E=weight.inv
```

The inverse of the asymptotic matrix (`AInv`), saved in the file weight.inv could be used in place of the matrix itself, with a command of the form:

```
AInv Full File=weight.inv
```

The `Full` keyword is essential here because Mx is agnostic about the symmetry of square matrices created in calculation groups. It is safer to assume that it is not symmetric to maintain consistency across applications. Alternatively, the inverse would implicitly be saved if the binary save feature described in Section 4.4.3 were used, which saves the whole job specifications.

2.3.3 Raw Data and Variable Length Files

Syntax:

`Rawdata/VLength {File=filename}`

Mx will read two types of raw data. `Rawdata` reads *rectangular* data, i.e. where every observation has the same number of input variables (`NInput_vars` on the `Data` line). This is appropriate if there are no missing data, or if missing data have been imputed. If the `Fix` keyword is used, the data are read from a file which should have a FORTRAN format as the first line. If the first line just has `*` or `(*)` on it, the data are read in *free format,* i.e. numbers are separated by one or more spaces or new line characters.

`VLength` is a variable length record reader, which allows reading of raw data where there may be missing values. The default (and mandatory) format for these data is *free.* A line with comments or `*` can be placed at the start of the file, but it will be ignored by Mx except for printing a warning and the line itself in the output file. The structure of a `VLength` file is as follows:

- number of input variables (k)
- identification codes for the k variables
- observed data for the k variables.

For every case, the number of input variables has to be on a line by itself. The identification codes must be integers that correspond to codes read by the `ICodes` command (see Section 3.5.2). For example, a file might contain the following:

```
3
1 2 3 .33 .62 .95
2
2 3 1.4 -2.2
1
2 .37
```

This example reads 3 variables for the first observation, with identification codes 1 2 3, and data values .33 .62 and .95. The second observation has no data for variable 1, but supplies data for 2 and 3, while the third supplies data for variable 2 alone. By default, data of this type are fitted using the raw maximum likelihood fit function (see Section 4.2.2).

It is quite simple to prepare `VLength` files with SAS or SPSS. However, caution should be exercised with SAS which uses a `.` for a missing value. Depending on the operating system under which you are running Mx, this dot may produce a file read error or be read as a zero. Here are a few lines of SAS code to output a `VLength` file from an array of two variables `V{2}`, either or both of which may be missing. The third and fourth lines need to be modified to declare the length of the array and to copy the required variables to the array into it. Certain applications may also need to change the format of the PUT statement that writes the data values.

```
DATA ONE; SET ZERO;
COUNT=0;
NVAR=2;                /*  Number of variables in total !!Change!! */
ARRAY V{2} AT1 AT2;    /*  Set up array for variables   !!Change!! */
DO I=1 TO NVAR;        /*  Count the non-missing observations */
 IF V{I} NE . THEN DO;
  COUNT+1;
 END;
END;
FILE MXVLFILE;          /* Filename for future Mx input !!Change!! */
IF COUNT NE 0 THEN DO; /* Write observations if there are any */
 PUT COUNT;
 DO I=1 TO NVAR;
  IF V{I} NE . THEN PUT I @@;             /* Write the identifiers */
 END;
 PUT;
 DO I=1 TO NVAR;
  IF V{I} NE . THEN PUT V{I} 13.6 +1 @@; /* Write the data values */
 END;
 PUT;
END;
```

2.3.4 Contingency Tables

Syntax:

```
CTable <r> <c> {File=filename}
```

Mx will read contingency tables of order r by c. NInput_vars must be 2 for a group reading a contingency table. Both r and c must be greater than 1 but they do not have to be equal. A contingency table contains frequency data (or counts) such that each cell C_{ij} indicates the number of observations falling in row category i and column category j. The frequencies supplied should be greater than or equal to zero. Mx will automatically handle incomplete ascertainment which the user can flag by supplying a negative number for cells that have not been ascertained. See Section 4.2.3 for an example that cross-classifies schizophrenic proband disease status with disease status in their relative. See Section 3.2.4 for details on specifying a matrix to contain estimates of thresholds.

The ordering of the categories should follow the natural numbering of the rows and columns, so that a table with a strong positive correlation between the variables would have large frequencies on the leading diagonal. Supplying a CTable changes the default fit function to the likelihood of observing the frequencies assuming a bivariate normal distribution of liability underlies the observed presence in a cell. See Section 4.2.3 for details on fitting structural models to contingency table data.

2.3.5 Means

Syntax:

```
Means {File=filename}
```

A vector of means, length `NInput_vars` may be read. When fitting models by maximum likelihood, a model for the mean structure may be provided. The joint likelihood of the means and the covariances is maximized, enabling tests of hypotheses about equality of means across variables or across groups. Similar tests are possible when models are fitted by maximum likelihood to raw data. However, for balanced data (in which all variables have been obtained from all subjects) considerable computer time may be saved by fitting to the means and covariances.

2.3.6 Higher Moment Matrices

Syntax:

```
Skewness/Kurtosis {File=filename}
```

Matrices of skewness and kurtosis may be read with these commands. These are provided for future developments in Mx that will allow model fitting to these types of data in addition to means and covariances. Currently there is no facility to use matrices read in this way. However, model fitting with higher moments could be done with user-specified functions (see Section 4.2.4).

2.4 Label and Select Variables

2.4.1 Labelling Input Variables

Syntax:

```
Labels <list of labels> /
```

Labels may be given for the observed data by issuing a `Label` command, before the `Matrices` command. These labels may be used to select variables, for example:

```
Data NGroups=1 NInput_vars=3 NObservations=171
CMatrix File=Cov.mat
Labels ALC1 ALC2 AGE
Select ALC2 ALC1 /
```

would read the lower triangle of a 3×3 covariance matrix from the file Cov.mat, and label the variables ALC1 ALC2 and AGE. The variables ALC2 and ALC1 are then selected for analysis, changing their original order. Note that a space is required before the '/' character when using labels. See also Section 3.5.1 for details on labelling specified matrices.

2.4.2 Select Variables

Syntax:

```
Select <numlist or varlist> /
```

Variables may be selected for analysis using the `Select` command. It may be used to reorder data or to pick a selected number of variables for analysis. In either case, *a / must end the command.* `Select` accepts *integers* which correspond to the order of the input variable. More conveniently, `Select` will operate on variable labels (see Section 2.4.1). The command will work with raw data as supplied by the `Rawdata` or `VLength` commands (see Sections 4.2.2 and 2.3.3).

Select with Variable Length Data

In combination with the `VLength` command, it will change the identification codes to consecutive integers starting at 1. For example, if the following `Select` line was read:

```
Select 3 4 2 /
```

a `VLength` record of the form:

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

would effectively be changed to:

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

thus the observation originally numbered 3 has become observation 1, observation numbered 4 has become observation 2, and observation numbered 2 has become observation 3. `Select` will automatically reduce the number of data vectors if there are no matches for a particular data vector and the codes in the `Select` line. The final number of vectors and observations used in the analysis is given in the output file.

`Select` cannot contain more numbers than the `NInput_vars` specified on the `Data` line. To do so would necessarily result in a singular correlation or covariance matrix. Likewise, the same variable cannot be selected twice.

2.5 Calculation and Constraint Groups

The use of calculation and constraint groups is very similar the use of groups that read data. The three types of group are fully command compatible with the exception of commands for reading data, which can be used by data groups alone.

2.5.1 Calculation Groups

The keyword `Calc` on the `Data` line indicates that the group is used for calculation. The calculated matrix formula from such a group is printed if the `RSiduals` command appears on the `Options` line. There are no restrictions on the type and dimensions of a matrix than can be produced with this command (other than memory limits). The result of the calculation may be used in later groups by using the `=%En` syntax when specifying a matrix, where `n` is the number of the calculation group. Note that there is a strict ordering within the input file; results cannot be taken from a calculation that has not yet occurred.

The `Calc` group provides a facility for printing out results of matrix operations. Any calculation group that is not followed by a constraint or data group is not calculated until the end of optimization, thus avoiding unnecessary waste of computer time.

2.5.2 Constraint Groups

Constraint groups, indicated by `Const` on the `Data` line, set up a matrix of nonlinear constraints. The matrix is symmetric, and is of order k, where k is the number of input variables (`NInput_vars`) specified on the input line. Thus the number of constraints in each such group is $k(k+1)/2$. Any nonlinear equality constraint can be expressed in the form

$$C\left\{\Theta\right\} = 0$$

where $C\left\{\Theta\right\}$ is a function of the free parameters. For example, if `NInput_vars=3`, six nonlinear constraints would be specified. The expected covariance matrix for the group would be structured:

	1	2	3
1	C_1		
2	C_2	C_3	
3	C_4	C_5	C_6

The optimizer would find values of the parameters that keep the C_i very close to zero. One way to think of these constraints is to imagine that the group has a very large sample size and has an observed covariance matrix full of zeroes.

Mx increases the number of degrees of freedom by the number of nonlinear constraints, which assumes that each constraint identifies a parameter. This is obviously not always the case, and the `DF` parameter on the `Options` line may be used to correct for failures of this assumption.

The optimization routine treats constraints in an intelligent fashion; if it finds the derivatives of the constraint functions with respect to certain parameters to be zero, it does not calculate them during optimization. This means that if some of the specified constraint functions are always zero, little additional computational cost is incurred. If there is no feasible point for the constraints, for example one of them always takes the value .5, an IFAIL=3 error message is returned.

Chapter 3

Building Models with Matrices

All groups, be they constraint, calculation, or data, require at least one matrix in order to operate. The next few sections describe the types of matrix that may be used, the operators that act on and between them, and ways of putting parameters and numbers into them.

3.1 Matrices Command

Syntax:

```
Matrices
<matrix name> <type> <rows> <columns> {= <name> <group> / Free}
<matrix name> <type> <rows> <columns> {= <name> <group> / Free}
.
.

<matrix name> <type> <rows> <columns> {= <name> <group> / Free}
```

A group must have the *3-letter* `MAT` command, followed by at least one matrix definition. The end of `Matrices` reading is signalled by an `Covariance`, `Means`, or `Threshold` command. Matrix names are restricted to one letter, from A to Z. The type of a matrix may be one of the 11 forms described in Table 3.1, and its row and column dimensions are specified with integers. Once the type and size of a matrix has been defined, it cannot be changed. Note that matrix definitions are *group specific;* for example, matrix A in group 1 does not have to be the same type or size as matrix A in group 2.

Optionally, a matrix may be *constrained* to equal a matrix previously specified. For example, we could use the command

Table 3.1: Types of matrices that may be specified in Mx. # free elements indicates the number of elements that can be altered by the user, where r is the number of rows and c the number of columns of the matrix.

Type	Structure	Shape	# of free elements
`Zero`	Every element is zero (null matrix)	Any	0
`Unit`	Every element is one (unit matrix)	Any	0
`Iden`	Identity matrix	Square	0
`IZero`	Identity | Zero partitioned matrix	Any	0
`ZIden`	Zero | Identity partitioned matrix	Any	0
`Diag`	Diagonal matrix	Square	r
`Sdiag`	Subdiagonal (zeros on & above diagonal)	Square	$r(r-1)/2$
`Stand`	Standardized (symmetric, ones on diagonal)	Square	$r(r-1)/2$
`Symm`	Symmetric	Square	$r(r+1)/2$
`Lower`	Lower triangular	Square	$r(r+1)/2$
`Full`	Full	Any	$r \times c$

```
A Symm 3 3 = Y2
```

to equate matrix A in this group to matrix Y in group 2. Note that the current group must be number three or greater. Several additional options allow constraints to other quantities found in previous groups, such as the observed or expected covariance matrix. The special codes for constraining a matrix to equal those defined or computed in previous groups are shown in Table 3.2. These add to the flexibility of Mx.

It is especially important to note that *none of the* `%E, %O, %M, %F` *and* `%P` *equalities may refer to groups that appear after the current group.* When matrices are constrained to be equal in this fashion, the type and row $\times$ column dimensions of the earlier matrix are retained. If the two specifications do not agree, a warning is printed. Both the number of rows and the number of columns must be supplied for square matrices, but only the first is used to define the size of the matrix.

All changeable elements of matrices are initialized at zero and are fixed parameters, unless the `Free` keyword is used, in which case each changeable element is specified as a different free parameter. Examples of the results of using the keyword `Free` are shown in Table 3.3.

More detail on specifying parameters in matrices is given in Sections 3.3 to 3.4.4.

Table 3.2: Special codes for constraining a matrix to equal one computed or defined in group n. Such equalities may be set only in groups $n+i$ where $i = 0, 1 \dots Ng$ where Ng is the number of groups in total. Note: NI_n is the number of input variables in group n following any selection; NR and NC are respectively the number of rows and columns in a contingency table, and may be requested only if group n has such a table.

Symbol	Matrix quantity	Dimensions
`%O`n	Observed covariance (data) matrix	$\mathrm{NI}_n \times \mathrm{NI}_n$
`%E`n	Expected covariance matrix	$\mathrm{NI}_n \times \mathrm{NI}_n$
`%M`n	Expected mean vector	$1 \times \mathrm{NI}_n$
`%P`n	Expected proportions under bivariate normal	$\mathrm{NR}_n \times \mathrm{NC}_n$
`%F`n	Function value	1×1

3.2 Covariances, Model or M2 Command

Syntax:

```
Covariances/M2/Model A {operator B {operator C ...}} /
```

The covariance command uses the matrices specified following the `Matrices` line and special symbols to perform operations on or between them. A covariances statement may contain a single matrix and no operations, or it could be very complex. The command may extend over several lines and *must end in a /*

3.2.1 Matrix Operations

In ordinary algebra, operators such as $+ - \times$ and $\div$ have an order of evaluation established by convention. Multiply and divide are done before addition and subtraction. Multiply and divide are done in left-to-right order if they appear consecutively, as are addition and subtraction. We could say then, that $\times$ and $\div$ have priority 1, and $+$ and $-$ have priority 2. These default priorities can be changed with the use of brackets () which specify that operations inside the brackets are done first. For example,

$$a + b \times c = a + bc$$

whereas

$$(a + b) \times c = ac + bc$$

A similar hierarchy has been established for the matrix operators in Mx, and again it may be revised by the use of brackets. Table 3.4 shows the matrix operators and their (default) order of evaluation.

Table 3.3: Examples of use of the matrices command to specify the dimensions of different matrix types. The keyword `Free` following each command makes each modifiable element in the matrix a separate free parameter, numbered in order as shown in the second column. In the third column, values of elements are shown, with ? representing a free parameter.

Example command	Parameter Matrix	Values
`A Zero 2 3 Free`	0 0 0 0 0 0	0 0 0 0 0 0
`B Unit 2 3 Free`	0 0 0 0 0 0	1 1 1 1 1 1
`C Iden 3 3 Free`	0 0 0 0 0 0 0 0 0	1 0 0 0 1 0 0 0 1
`D IZero 2 5 Free`	0 0 0 0 0 0 0 0 0 0	1 0 0 0 0 0 1 0 0 0
`E ZIden 2 5 Free`	0 0 0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0 0 1
`F Diag 3 3 Free`	1 0 0 0 2 0 0 0 3	? 0 0 0 ? 0 0 0 ?
`G SDiag 3 3 Free`	0 0 0 1 0 0 2 3 0	0 0 0 ? 0 0 ? ? 0
`H Stand 3 3 Free`	0 1 2 1 0 3 2 3 0	1 ? ? ? 1 ? ? ? 1
`I Symm 3 3 Free`	1 2 4 2 3 5 4 5 6	? ? ? ? ? ? ? ? ?
`J Lower 3 3 Free`	1 0 0 2 3 0 4 5 6	? 0 0 ? ? 0 ? ? ?
`K Full 2 4 Free`	1 2 3 4 5 6 7 8	? ? ? ? ? ? ? ?

Table 3.4: Matrix operators available in Mx, together with their priority for evaluation. See also Table 3.5 for matrix functions.

Symbol	Name	Function	Example	Priority
~	Inverse	Inversion	`A~`	1
'	Transpose	Transposition	`A'`	1
^	Power	Element powering	`A ^ B`	2
*	Star	Multiplication	`A * B`	3
.	Dot	Dot product	`A . B`	3
@	Kron	Kronecker product	`A @ B`	3
%	Eldiv	Element division	`A % B`	3
+	Plus	Addition	`A + B`	4
-	Minus	Subtraction	`A - B`	4
|	Bar	Horizontal adhesion	`A | B`	5
_	Under	Vertical adhesion	`A _ B`	6

A line has been drawn between the first two operators (Inverse & Transpose) and the rest because inverse and transpose are *unary* operators, that is, they operate on *one* matrix. The rest form a single new matrix from two matrices, and are thus *binary* operators. These operators are now described in detail.

Readers unfamiliar with matrix algebra may benefit from reading Appendix C, where examples and exercises are given. Readers familiar with matrix algebra may wish to examine Tables 3.4 and 3.5 for the variety of available operators and functions, and use Sections 3.2.1 and 3.2.2 for reference.

Inverse $\sim$

Only square matrices may be inverted, but they may be either symmetric or non-symmetric. The inverse of matrix $\mathbf{A}$ is usually written $\mathbf{A}^{-1}$ and implies that

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$$

where $\mathbf{I}$ is the identity matrix. To request an inverse with Mx, we use the symbol `~`. If the inverse does not exist (possibly due to rounding errors), Mx will terminate with an error message. Some precautions can be taken to avoid this, such as supplying starting values that allow inversion, or putting boundary constraints on parameters to prevent their taking values that would lead to a singular matrix.

Transpose $'$

Any matrix may be transposed. The transpose of $\mathbf{A}$ is written $\mathbf{A}'$. The order of the matrix changes from $r \times c$ to $c \times r$, as the rows become the columns and vice-versa.

Power ^

All the elements of a matrix may be raised to a power using the ^ symbol. The matrix to be powered may be of any dimension, but the power to which each element must be raised has to be specified in a 1×1 matrix. It is possible to use negative powers and non-integer exponents to indicate reciprocal functions and roots of elements, but note that it is not possible to raise a negative number to a non-integer power. For example, the cube of every element of a matrix would be obtained by `A^B` if `B` was a 1×1 matrix with 3 as its only element.

Star $*$

Star is the ordinary form of matrix multiplication. The elements of $\mathbf{A}$ $(m \times n)$ and $\mathbf{B}$ $(n \times p)$ are combined to form the elements of matrix $\mathbf{C}$ $(m \times p)$ using the formula

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \times B_{kj}$$

Matrices multiplied in this way have to be *conformable for multiplication*. This means that *the number of columns in the first matrix must equal the number of rows in the second matrix.*

For example, the matrix product $\mathbf{A} * \mathbf{B}$

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} * \begin{pmatrix} g & h \\ i & j \end{pmatrix} = \begin{pmatrix} a \times g + b \times i & a \times h + b \times j \\ c \times g + d \times i & c \times h + d \times j \\ e \times g + f \times i & e \times h + f \times j \end{pmatrix}$$

$$= \begin{pmatrix} ag + bi & ah + bj \\ cg + di & ch + dj \\ ge + fi & eh + fj \end{pmatrix}$$

Dot .

Dot is another type of matrix multiplication, which is done *element by element.* For two matrices to be multiplied in this way, they have to have the same dimensions, i.e. the number of rows in $\mathbf{A}$ has to be the same as the number of rows in $\mathbf{D}$, and the number of columns in $\mathbf{A}$ has to be the same as the number of columns in $\mathbf{D}$. Elements of the dot product are described by the formula

$$C_{ij} = A_{ij} \times D_{ij}$$

For example, the dot product $\mathbf{A}.\mathbf{D}$ is

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} . \begin{pmatrix} g & h \\ i & j \\ k & l \end{pmatrix} = \begin{pmatrix} a \times g & b \times h \\ c \times i & d \times j \\ e \times k & f \times l \end{pmatrix}$$

Kron $\otimes$

The *right* Kronecker product of two matrices $\mathbf{A} \otimes \mathbf{B}$ is formed by multiplying each element of $\mathbf{A}$ by the matrix $\mathbf{B}$. If $\mathbf{A}$ is of order $(m \times n)$ and $\mathbf{B}$ is of order $(p \times q)$, then the result will be of order $mp \times nq$. There are no conformability criteria for this type of product. The symbol $\otimes$ is imitated in Mx input files with the symbol `@`.

For example, the Kronecker product $\mathbf{A} \otimes \mathbf{B}$ is

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \otimes \begin{pmatrix} g & h \\ i & j \end{pmatrix} = \begin{pmatrix} a \times g & a \times h & b \times g & b \times h \\ a \times i & a \times j & b \times i & b \times j \\ c \times g & c \times h & d \times g & d \times h \\ c \times i & c \times j & d \times i & d \times j \\ e \times g & e \times h & f \times g & f \times h \\ e \times i & e \times j & f \times i & f \times j \end{pmatrix}$$

Eldiv %

Eldiv does *element by element* division. For two matrices to be divided in this way, they have to have the same dimensions, i.e. the number of rows in $\mathbf{A}$ has to be the same as the number of rows in $\mathbf{D}$, and the number of columns in $\mathbf{A}$ has to be the same as the number of columns in $\mathbf{D}$. Elements of the result are described by the formula

$$C_{ij} = A_{ij} \div D_{ij}$$

If any element of $\mathbf{D}$ is zero, the corresponding cell in the result matrix is set to 10^{35}.

For example, the eldiv $\mathbf{A}\%\mathbf{D}$ is

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \begin{pmatrix} g & h \\ i & j \\ k & l \end{pmatrix} = \begin{pmatrix} a \div g & b \div h \\ c \div i & d \div j \\ e \div k & f \div l \end{pmatrix}$$

Plus +

Addition of matrices is performed *element by element.* For two matrices to be added, they have to have the same dimensions, i.e. the number of rows in $\mathbf{A}$ has to be the same as the number of rows in $\mathbf{D}$, and the number of columns in $\mathbf{A}$ has to be the same as the number of columns in $\mathbf{D}$. Elements of the sum, $\mathbf{C}$ are described by the formula

$$C_{ij} = A_{ij} + D_{ij}$$

For example, the sum $\mathbf{A} + \mathbf{D}$ is

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} + \begin{pmatrix} g & h \\ i & j \\ k & l \end{pmatrix} = \begin{pmatrix} a + g & b + h \\ c + i & d + j \\ e + k & f + l \end{pmatrix}$$

Minus $-$

Subtraction of matrices is performed *element by element.* For one matrix to be subtracted from another, they have to have the same dimensions, i.e. the number of rows in $\mathbf{A}$ has to be the same as the number of rows in $\mathbf{D}$, and the number of columns in $\mathbf{A}$ has to be the same as the number of columns in $\mathbf{D}$. Elements of the difference, $\mathbf{C}$ are described by the formula

$$C_{ij} = A_{ij} - D_{ij}$$

For example, the difference $\mathbf{A} - \mathbf{D}$ is

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} - \begin{pmatrix} g & h \\ i & j \\ k & l \end{pmatrix} = \begin{pmatrix} a-g & b-h \\ c-i & d-j \\ e-k & f-l \end{pmatrix}$$

Note that in MX there is also a *unary* minus operator, so that an expression such as $-\mathbf{A}$ is legal. This operation changes the sign of each element of $\mathbf{A}$

Bar $|$

Bar allows partitioning of matrices. Its operation is called horizontal adhesion because $\mathbf{A}|\mathbf{D}$ is formed by sticking $\mathbf{D}$ onto the right hand side of $\mathbf{A}$. For two matrices to be adhered in this way, they have to have the same number of rows. If $\mathbf{A}$ $(m \times n)$ and $\mathbf{D}$ $(m \times p)$ are adhered, the result $\mathbf{C}$ is of order $(m \times (n+p))$.

For example, the operation $\mathbf{A}|\mathbf{D}$ is

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} | \begin{pmatrix} g & h \\ i & j \\ k & l \end{pmatrix} = \begin{pmatrix} a & b & g & h \\ c & d & i & j \\ e & f & k & l \end{pmatrix}$$

Under _

Under allows partitioning of matrices. Its operation is called vertical adhesion because $\mathbf{A}\_\mathbf{D}$ is formed by sticking $\mathbf{D}$ underneath $\mathbf{A}$. For two matrices to be adhered in this way, they have to have the same number of columns. If $\mathbf{A}$ $(m \times n)$ and $\mathbf{D}$ $(p \times n)$ are adhered, the result $\mathbf{C}$ is of order $((m+p) \times n)$.

For example, if

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

and

$$\mathbf{D} = \begin{pmatrix} g & h \\ i & j \\ k & l \end{pmatrix}$$

then

$$\mathbf{A\_D} = \begin{pmatrix} a & b \\ c & d \\ e & f \\ g & h \\ i & j \\ k & l \end{pmatrix}$$

3.2.2 Matrix Functions

A number of matrix functions, shown in Table 3.5, have been provided in Mx, which are especially useful for in specialized applications involving user-defined fitting-functions (`US`; see Section 4.2.4, page 54). Functions differ from operators because they take an *argument* enclosed by brackets (). This argument may be a single matrix name, or a complex matrix formula. The argument is evaluated *before* the function is applied, consistent with the rules for using brackets. Functions form a second set of unary operators (see Section 3.2). Descriptions of these functions follow.

Trace \tr()

The trace of a matrix is the sum of the elements on the leading diagonal, i.e.

$$\sum_{i=1}^{n} A_{ii}$$

It is only allowed for *square* matrices.

Determinant \det()

Properties of determinants, and ways of calculating them are discussed in Appendix C. `\det()` is only calculated for *square* matrices.

Sum \sum()

The sum of a matrix is the sum of all its elements, i.e.,

$$\sum_{i=1}^{r} \sum_{j=1}^{c} A_{ij}$$

Product \prod()

The product function of a matrix yields the product of all its elements, i.e.,

$$\prod_{i=1}^{r} \prod_{j=1}^{c} A_{ij}$$

Table 3.5: Matrix functions available in Mx. Restrictions are on rows (r) and columns (c) of input argument.

Keyword	Function	Restrictions	Result Dimensions
`\tr()`	Trace	$r = c$	(1×1)
`\det()`	Determinant	$r = c$	(1×1)
`\sum()`	Sum	$None$	(1×1)
`\prod()`	Product	$None$	(1×1)
`\max()`	Maximum	$None$	(1×1)
`\min()`	Minimum	$None$	(1×1)
`\abs()`	Absolute value	$None$	$(r \times c)$
`\cos()`	Cosine	$None$	$(r \times c)$
`\cosh()`	Hyperbolic Cosine	$None$	$(r \times c)$
`\sin()`	Sin	$None$	$(r \times c)$
`\sinh()`	Hyperbolic Sin	$None$	$(r \times c)$
`\tan()`	Tan	$None$	$(r \times c)$
`\tanh()`	Hyperbolic Tan	$None$	$(r \times c)$
`\exp()`	Exponent ($e^{\mathbf{A}}$)	None	$(r \times c)$
`\ln()`	Natural logarithm	None	$(r \times c)$
`\sqrt()`	Square root	None	$(r \times c)$
`\d2v()`	Diagonal to Vector	None	$(\min(r, c) \times 1)$
`\v2d()`	Vector to Diagonal	$r = 1$ or $c = 1$	$(\max(r, c) \times \max(r, c))$
`\m2v()`	Matrix to Vector	None	$(rc \times 1)$
`\vec()`	Matrix to Vector†	None	$(rc \times 1)$
`\vech()`	Lower triangle to Vector	None	$(rc \times 1)$
`\stnd()`	Standardize matrix	$r = c$	$(r \times c)$
`\eval()`	Real eigenvalues	$r = c$	$r \times 1$
`\evec()`	Real eigenvectors	$r = c$	$r \times r$
`\ival()`	Imaginary eigenvalues	$r = c$	$r \times 1$
`\ivec()`	Imaginary eigenvectors	$r = c$	$r \times r$

†`vec` vectorizes by columns, in contrast to `m2v`, which vectorizes by rows.