New Features in Mx 1.26

Michael C. Neale[1]

December 7, 1994

[1]Department of Psychiatry, Medical College of Virginia, Box 980710, Richmond, Virginia 23298, USA.

Contents

1 Introduction

During 1994 Mx has been revised and enhanced in many ways. Although a second edition of the manual was released in February 1994, it is already out of date in several areas. When the current rapid development rate slows to a comfortable pace a new manual will be produced. In the interim, this document is intended to supplement the 1994 Mx User's Guide. The following sections describe the new features according to their content. The file newfeatu.txt distributed with the PC version and available on the internet lists the version numbers at which the changes were made.

Backward compatibility is almost entirely preserved. The change to the order of evaluation of unary matrix operators `'` and `~` that follow matrix function arguments (e.g. `\eval(A)'`) is an exception. In general, moving to the newer syntactic forms will clarify the scripts you write and make it easier for others (including yourself six months later) to understand. We are behind in revising the example scripts. If you happened to think that revising them yourself would help you to understand the features of the package, and then happened to forward the results to us, both we and future new users would be very grateful!

2 Internet support

Mx is still public domain. You may download versions for most systems from opal.vcu.edu which has executables for most platforms along with documentation and examples. The latter have been extended to include all the appendices from the Neale & Cardon book (Neale and Cardon, 1992). There are also some simple scripts for Multitrait-multimethod models that are very easy to revise for different numbers of variables.

There is world-wide-web access to Mx. http://opal.vcu.edu/html/Mx.html is the Mx home page. Email bug reports, requests for further information, and *most important* your comments and suggestions for improvements to neale@ruby.vcu.edu - it is hard to overemphasize how important constructive criticism is at this time.

3 Job structure

One day there will be a graphical interface to Mx that will relieve the user of getting to grips with the details of scripts. Until then, some wrestling with the script language will be necessary. The good news is that a much deeper understanding of modeling can come from this activity. We are in the process of revising the script language to enhance its flexibility and readability.

Mx scripts used to be of the form:

1. TITLE - for reference
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 - signifies end of group

Positional information was used to mark the end of the MATRICES definition section; the script reader would detect a covariance/mean/threshold/compute statement and prepare to read the matrix formula. There is now a facility demarcate the matrix definition section, using the syntax:

```
Begin matrices;
A Full 5 5
End matrices;
```

After defining such matrices, the user is free to enter values and parameter specifications for the matrices prior to giving a matrix formula for covariance structure or means. This makes no difference to the final outcome, but it gives a more logical structure to the script. It is essential to use this prior definition approach when using the `\part` matrix function.

Lines in Mx scripts may be up to 1200 characters long on most systems; the IBM RS6000 compiler has an upper limit of 500 characters, which is the limit for the AIX version. The previous limit was 120 characters, and did not apply to data files read by CM/VL commands.

4 Multistatement Matrix Algebra

In many cases breaking up a complicated matrix algebra expression into smaller parts can improve readability or efficiency or both. For example, the matrix formula

```
(I-A)~*S*(I-A)~'
```

will find the inverse of `(I-A)` twice. For small A the loss of efficiency will be negligible - the extra time taken to re-program will be greater than any gained in execution time. For large A, users may already have found calculation groups to be a useful way of reducing calculations or simplifying script structure. Now there is a better way, defining new matrices as matrix functions:

```
begin matrices;
A full 10 10
S symm 10 10
I iden 10 10
end matrices;
begin algebra;
B = (I-A)~ ;
C = B*S*B' ;
end algebra;
```

so that the cpu-intensive matrix inversion is only carried out once. Each matrix that appears on the left hand side of the = sign is newly defined in this group (it must not have been previously defined). Note that matrix B, defined in the first line of algebra, may be used in subsequent lines.

4.1 Equating Matrices across groups

The usual equating of matrices across groups, with syntax of the form

```
A Full 3 3 = X(2)
```

in which matrix A is equated to matrix X in group 2 is supplemented by a global facility. All the matrices defined in an earlier group may be made available with the statement

```
Matrices = Group 1
```

It is possible to equate matrices that were defined in a `begin algebra ... end algebra` section across groups. However, the formula is copied between groups and is evaluated locally for each group, so no improvement in efficiency will be obtained this way. To obtain pre-computed matrices the compute statement must be used in an earlier group, and its result referenced via `A full 1 1 = %E1` syntax.

5 The #Define Command

In multivariate modeling it is quite common that the same matrix dimensions are used in many different parts of a script. For example, in an oblique factor analysis, with 10 observed variables and 2 factors, the dimensions of the matrices needed to define the model are dictated by these numbers. If matrix **L** contains the loadings, **P** the correlations between the loadings, and matrix **E** the residuals, we would require **L** to be of order 10×2 and **E** to be of order 10×10. Traditionally we would specify this in Mx with a job of the form

```
Title - factor analysis
Data NInput=10 NObservations=100
Matrices
A Full 10 2 Free
P Stan 2 2 Free
E Diag 10 10 Free
Covariance A*P*A' + E /
Start .5 all
End
```

This could be made more general with a couple of `#define` statements:

```
Title - factor analysis
#define factors 2
#define vars 10
Data NInput=10 NObservations=100
Matrices
A Full vars factors Free
P Stan factors factors Free
E Diag vars vars Free
Covariance A*P*A' + E /
Start .5 all
End
```

There is little to be gained in this simple model - we change two numbers to change the number of factors and number of observed variables, instead of six. As the models grow in complexity, the use of `#define` can bring greater simplicity and functionality to the script.

When Mx is looking for a number, if it encounters a letter it will read the word and check the dictionary for matching words to see if it has been defined. If it hasn't, a warning will be printed and the search for a number or a #define'd variable will continue. Some care is needed with spelling!

6 Matrix Functions and Operators

6.1 Unary functions

Several unary matrix functions, called with syntax of the form `\func(M)` have been added. These are shown in Table 1, along with the original list. We now discuss the new functions (from `\mean` onwards).

6.1.1 Column Means \mean

This function will compute the means of the columns of a matrix.

6.1.2 Column Covariances \aorder

This function will compute the covariance matrix of the columns of a matrix. Thus if data are presented as one line per subject, with r rows for each of the c variables, the output would be of order $c \times c$

6.1.3 Multivariate Normal Integration \mnor

The matrix function `\mnor` will compute multiple integrals of the multivariate normal, up to dimension 10. Its input is structured so that for n dimensional integration, the matrix has n columns and $n + 4$ rows. The first n rows define the covariance matrix, row $n + 1$ defines the mean vector, the last three are used to define the type of truncation experienced by each variable. This is best described with an example. The script:

```
Test multivariate normal integral function
data calc ng=1
matrices
a full 1 2 ! Upper limits
b full 1 2 ! Lower limits
t full 1 2 ! Type of integral
r stan 2 2 ! Covariance matrix
z full 1 2 ! Means
compute \mnor((r_z_a_b_t)) /
matrix r .3
matrix b 0 0
matrix a 1 1
matrix t 2 2
```

Table 1: Matrix functions available in Mx. Restrictions and result dimensions refer to 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	$r = c$	$r \times r$
`\mean()`	Mean of columns	None	$1 \times c$
`\cov()`	Covariance of columns	None	$c \times c$
`\mnor`	Multivariate normal integral	$r = c + 3$	1×1
`\momnor`	Moments of Multivariate normal	$r = c + 3$	1×1
`\aorder()`	Ascending sort order	$r \times 1$	$r \times 1$
`\dorder()`	Descending sort order	$r \times 1$	$r \times 1$
`\sortr()`	Row sort	None	$r \times \max(1, c - 1)$
`\sortc()`	Column sort	None	$\max(1, r - 1) \times c$
`\part()`	Extract part of matrix	None	Variable‡

†`vec` vectorizes by columns, in contrast to `m2v`, which vectorizes by rows.
‡ \part(A,B) takes two arguments. The elements of the 1×4 matrix B are used to define a rectangle within matrix $\mathbf{A}$ to be extracted.

```
option rs
end
```

computes the integral of the bivariate normal distribution with correlation .3 from 0 to 1 in both dimensions. The type parameters (matrix t) are flags that indicate the type of truncation required:

- 0 integral from a_j to infinity
- 1 integral from -infinity to a_j
- 2 integral from a_j to b_j
- 3 integral from $-\infty$ to $+\infty$ (this dimension is ignored)

where a_j and b_j are the elements of column j of matrices a and b.

Accuracy is set to six decimals by default. Lower precision may be set with

```
option eps=<value>
```

although it should be noted that this option will be treated globally, i.e., for all such integrals in a particular run.

6.1.4 Moments of the Truncated Multinormal \momnor

The matrix function `\momnor` will compute moments of the truncated multinormal distribution. Currently, it will work only with 'tails' of the distribution, though selection may be absent for some variables. Here is a bivariate example:

```
Test moments of truncated normal function
data calc ng=1
matrices
r symm 2 2  !covariance matrix
m full 1 2  !means
t full 1 2  !thresholds
s full 1 2  !selection vector
n full 1 2  !# of abscissae
compute \momnor((r_m_t_s_n)) /
matrix r 1 .5 1
matrix m 0 0
matrix t 1.282 1.282
matrix s 1 1
matrix n 16 16
option rs
end
```

This script requests the covariances and means of individuals selected above the threshold 1.282 in a N(0,1) bivariate normal distribution. It returns the covariance matrix in the first n rows, and the means in row $n + 1$.

Note: this function has not been tested very thoroughly yet. CPU time will go up with the number of abscissae, but 64 is the maximum (and it goes in jumps 16 20 24 32 48 64, along with some smaller jumps below that). Mx will take care of it by assigning i) 16 if you enter 0 or less, ii) 64 if you enter 64 or more, and iii) the next lowest value if you happen to chose an intermediate value (eg it will pick 24 if you enter 30).

6.1.5 Ascending Order \aorder

This function gets the ascending order of a column vector. For example, `\aorder(A)` with

$$\mathbf{A} = \begin{pmatrix} .6 \\ .1 \\ .3 \end{pmatrix}$$

would yield

$$\begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}$$

6.1.6 Descending order \dorder

This function gets the descending order of a column vector. For example, `\dorder(A)` with

$$\mathbf{A} = \begin{pmatrix} .6 \\ .1 \\ .3 \end{pmatrix}$$

would yield

$$\begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}$$

6.1.7 Sort Rows \sortr

Used to sort a column vector or matrix by rows. If a vector, the vector elements themselves are used to sort. If a matrix, the first column is taken to be the sort order - and must contain a permutation of the integers 1 to the number of rows, as might be extracted using, e.g., `\aorder` above.

6.1.8 Sort Columns \sortc

This function works the same way as `\sortr` but by columns.

6.1.9 Extract Part \part(A,B)

would take a rectangular patch out of matrix A (formerly this was possible only by pre- & post-multiplying by elementary matrices). One has to be very careful to initialize matrix B before this statement is given, because the result dimensions are needed to check syntax. To pre-initialize B you would use the following job structure

```
Title
Data Calc Ngroups=1
Begin Matrices;
A Symm 3 3
B Full 4 1
End Matrices;  !    <- End matrix definitions with this statement
Matrix A
1
2 3
4 5 6
Matrix B
2 1 3 3
Compute \part(A,B) /   !  <- Compute statement *after* matrix statement
Option rs
End
```

The format for matrix **B** is row,column,row,column so in this example the rectangle from 2,1 (row 2, column 1) to 3,3 will be extracted, giving

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

Note that the elements of B may define any two opposite corners of a submatrix of A.

6.1.10 Order of evaluation

Versions of Mx up to and including 1.25 displayed somewhat counterintuitive evaluation of expressions such as `\eval(A)'`. This used to give the eigenvalues of the transpose of **A**, and now gives the more intuitive transpose of the matrix of eigenvalues.

6.2 Binary Operators

Table 2 shows all the binary matrix operators available in Mx. There is only one addition since version 1.10, the quadratic product denoted by the `&` character. The matrix power function (`^`) was changed so that the second matrix is no longer restricted to a scalar. It now works like a Kronecker product but with elements of the first matrix raised to the power of those in the second matrix instead of multiplied. To some extent, the `\part()` function is binary, but we prefer to list this with the other matrix functions.

6.2.1 Quadratic &

Many structural equation and other models rely on this form and if **A** is a complex function of matrices, the quadratic operator is both simple and efficient to implement. Note that **A** can be any shape, but to be conformable for quadratic product the matrix **B** must be square and have the same number of columns as the matrix **A**.

For example, the quadratic product **A&B**

$$\begin{pmatrix} a & b \end{pmatrix} * \begin{pmatrix} g & h \\ i & j \end{pmatrix} * \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} a^2g + abi + abh + b^2i \end{pmatrix}$$

Table 2: Matrix operators available in Mx, together with their priority for evaluation. See also Table 1 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
&	Quad	Quadratic `A*B*A'`	`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

7 Methods for Model Fitting

7.1 Reading Data

Although variable-length 'vl' files are a compact way of supplying datasets that include missing values, they are not always convenient to produce. Some expertise in the use of the data-handling preprocessor (e.g., SAS, SPSS) is needed to generate vl files and mistakes can happen. Instead, users may now input missing data in rectangular format, using . (dots) as missing value indicators. The command to read such files is

```
Rectangular file=filename
```

A common error in reading data with `CM` or `AC` commands is to read them as full matrices when they are stored as symmetric, or vice versa. Mx now attempts to be a bit smarter about this process. If a user forgets to put the `FU` keyword on the `CM` line, but Mx detects an Mx-style data file that was saved in full format, it will read it as full instead.

7.2 Definition variables

This feature allows new and interesting things with vl or rectangular data files. Essentially, some of the variables may be assigned as definition variables which may subsequently be used in defining parameters of the model. We shall illustrate this with an example: suppose that the covariance between two variables varies as a function of age. A traditional approach to this problem might involve splitting the sample into two groups, young and old, and fitting a two-group model. Comparison of the fit statistic obtained when the covariance is constrained to be equal in the two groups to that obtained when each group has its own covariance structure provides a test of heterogeneity. But what if want to make use of all the information on age, which is a continuous variable, rather than just pick an arbitrary cutoff for young vs. old? We can use the observed age variable to define the covariance structure for that particular observation.

Suppose variables 1, 2 and 3 correspond to verbal IQ, quantitative IQ, and age, which we read from file iq.vl. Mx uses the `definition` keyword to identify variables that are to be used in the model; they are extracted from the data set so modeling is restricted to the other variables. Comments begin with a `!` character.

```
Title - verbal and performance IQ covariance as a function of age
Data NInput=3 Ngroups=1   ! single group example with 3 variables
Labels Verbal Quant Age
VL file=IQ.VL
Definition_variables Age /  ! This variable is referenced as -1 in
!                              specification statements
Matrices
L Lower 2 2 Free ! Triangular matrix of paths from factor Fi to variable Oj
X Lower 2 2 Free ! Triangular matrix of paths from factor Ai to variable Oj
R Diag  2 2      ! Matrix for variances of Ai latent variables
M Full 1 2 Free  ! Matrix for estimating means
Means M /                   ! Formula to compute mean vector
Covariance L*L' + X*R*X' /  ! Formula to compute covariances
Specify R        ! Place our definition variable on the diagonal elements of R
-1 -1
Matrix M 100 100 ! Starting values for means
Matrix L 15 0 15 ! Starting values for constant covariance component
Matrix X 3 0 3   ! Starting values for age-dependent covariance component
Option Rs        ! Request some output
End
```

Internally, Mx will recalculate the predicted covariance matrix for every observation. In fact, it only does this if the definition variables have different values from the preceding case, so sorting may improve performance if the definition variables are quasi-continuous or ordinal. The usual raw data log-likelihood function is computed for every vector of observations, but using the appropriate covariance matrix for this group. If we were to assign one and zero to the age variable in accordance to our cutoff approach, we would get the same results as the two-group heterogeneity method.

Apart from this handling of continuous heterogeneity, we should be aware that considerably more complex models may be attacked. All the tools of Mx matrix functions and operators may be used to define linear and non-linear functions of the definition variables and model parameters.

7.3 Optimization changes

The backend optimizer has been revised by Walter Murray and colleagues to allow some backing-out from infeasible regions. For example, if the fit function involves a multivariate normal integral, but the current parameter estimates give a non-positive definite matrix for this operation, the fit function cannot be computed. It is possible for optimization to be abandoned — or even started — at such an infeasible point, in which case the new IFAIL=-1 will be reported.

7.4 New Options

7.4.1 Fit indices

Users may supply the results of fitting a null model (usually a simple diagonal model of variances, but others are possible) with the command

```
Option null=10.5,3
```

which will extend the output in the following way:

```
 Fit statistic                      Estimate
Tested Model fit >>>>>>>>>>>>>       4.73426
Tested Model df >>>>>>>>>>>>>>       4.00000
Null Model fit* >>>>>>>>>>>>>>    1563.94400
Null Model df* >>>>>>>>>>>>>>>       6.00000
Normed fit Index >>>>>>>>>>>>>       0.99697
Normed fit Index 2 >>>>>>>>>>>       0.99953
Tucker Lewis Index >>>>>>>>>>>       0.99929
Parsimonious Fit Index>>>>>>>>       0.66465
Parsimonious Fit Index 2 >>>>>       0.02940
Relative Non-centrality Index>       0.99953
Centrality Index >>>>>>>>>>>>>       0.99961

*User-supplied null-model statistic
```

They are calculated as follows:

$$
\begin{aligned}
NFI &= (F_N - F_T)/F_N && (1)\\
NFI_2 &= (F_N - F_T)/(F_N - df_T) && (2)\\
TLI &= ((F_N/df_N) - (F_T/df_T))/((F_N/df_N) - 1.0) && (3)\\
PFI &= (df_T/df_N)NFI && (4)\\
PFI_2 &= 2NFI_2(df_T)/(p*(p-1)) && (5)\\
RNI &= ((F_N - df_N) - (F_T - df_T))/(F_N - df_N) && (6)\\
CI &= \exp(-.5((F_T - df_T)/N)) && (7)
\end{aligned}
$$

where F_N and F_T are the goodness-of-fit (chi-squared) statistics respectively obtained under the Null and Tested model, and which have df_N and df_T degrees of freedom. N is the total sample size (over all groups) and p is the number of free parameters in the model. Tanaka (1993) is a useful source for these statistics; later versions of this document will include further citations.

7.4.2 Option check

By default, Mx does not test identification of models via examination of the rank of the hessian matrix of parameter estimates. Option check does this, but it should be noted that the results can give either false positives or false negatives. While this is true to some extent of programs that use exact derivatives, it is more true of Mx which uses numerically estimated derivatives. When option check is invoked, Mx computes the eigenvalues and eigenvectors of the hessian matrix, and uses this information to assess potential areas of underidentification. As stated elsewhere — especially in Jöreskog's early papers — a better procedure is to attempt to find alternative sets of parameter estimates that give an equally good fit to the data (which is proof of underidentification). Mx provides `option th=` to facilitate this proof.

7.4.3 Standard errors SE

This option is not strictly new, but has been updated to work more reliably. This option uses the numerical estimates of the hessian matrix of the parameters to provide approximate standard errors on the parameters. It gives estimates consistent with those of other programs under a variety of conditions, but may fail if penalty functions or estimates out of feasible bounds are used to obtain the hessian approximation. Under these circumstances, standard errors may appear much too small. The same is true if the function precision is low (which may happen if, e.g., the fit function involves numerical estimation of integrals). The problem is sometimes overcome by altering the error function precision parameter, with `option err=value` By default it is set close to machine accuracy; setting it to 1.e-10 (or larger) may correct problems with totally unrealistic estimates of standard errors.

Again, assessing significance and standard errors directly through changes in the model can provide more robust estimates. It is possible to trick Mx into doing this for you, but we hope to implement the method in a more user-friendly way soon.

7.5 Changing the model

Multiple fit mode has always made revising the model and refitting from earlier solutions easy to do by changing the parameter contents and values of matrix elements. It is now possible to change matrix formulae and other characteristics of a group, using the

```
#group 3
```

syntax. Options or matrix formulae supplied after this command would apply to group 3. For example, suppose that after fitting a model — perhaps one that took days to run — we might discover that we had accidentally forgotten to request residuals from group 5. If, in anticipation of this or similar errors, we had issued a save command:

```
< usual model commands >
Option multiple
End       !<- end statement of last group

save incase.mxs
End
```

it would be possible to retrieve the solution, add the appropriate option, and re-run:

```
Title - revise options to see residuals
get incase.mxs
#group 5
option rs
End
```

This type of strategy may be useful to obtain the additional fit-statistics for null-model comparisons. Additional features in this area are under construction.

8 Acknowledgements

Mx development is supported by NIH grant RR08123. We are very grateful to all those who made comments and suggestions for improving the program. Please keep the comments coming!

References

Neale, M. C. & Cardon, L. R. (1992). *Methodology for Genetic Studies of Twins and Families.* Kluwer Academic Publishers.

Tanaka, J. S. (1993). Multifaceted conceptions of fit in structural equation models. In K. A. Bollen & J. S. Long (Eds.), *Testing Structural Equation Models* (pp. 10–39). Sage Publications.