Title

[M-0] intro — Introduction to the Mata manual

Contents

Description

Mata is a matrix programming language that can be used by those who want to perform matrix calculations interactively and by those who want to add new features to Stata.

This entry describes this manual and what has changed since Stata 10.

Remarks

This manual is divided into six sections. Each section is organized alphabetically, but there is an introduction in front that will help you get around.

If you are new to Mata, here is a helpful reading list. Start by reading

[M-1] **first**	Introduction and first session
[M-1] **interactive**	Using Mata interactively
[M-1] **how**	How Mata works

You may find other things in section [M-1] that interest you. For a table of contents, see

[M-1] **intro**	Introduction and advice

Whenever you see a term that you are unfamiliar with, see

[M-6] **Glossary**	Mata glossary of common terms

Now that you know the basics, if you are interested, you can look deeper into Mata's programming features:

[M-2] **syntax**	Mata language grammar and syntax

[M-2] **syntax** is pretty dense reading, but it summarizes nearly everything. The other entries in [M-2] repeat what is said there but with more explanation; see

[M-2] **intro**	Language definition

because other entries in [M-2] will interest you. If you are interested in object-oriented programming, be sure to see [M-2] **class**.

Along the way, you will eventually be guided to sections [M-4] and [M-5]. [M-5] documents Mata's functions; the alphabetical order makes it easy to find a function if you know its name but makes learning what functions there are hopeless. That is the purpose of [M-4]—to present the functions in logical order. See

[M-4] **intro**	Index and guide to functions
Mathematical	
[M-4] **matrix**	Matrix functions
[M-4] **solvers**	Matrix solvers and inverters
[M-4] **scalar**	Scalar functions
[M-4] **statistical**	Statistical functions
[M-4] **mathematical**	Other important functions
Utility and manipulation	
[M-4] **standard**	Functions to create standard matrices
[M-4] **utility**	Matrix utility functions
[M-4] **manipulation**	Matrix manipulation functions
Stata interface	
[M-4] **stata**	Stata interface functions
String, I/O, and programming	
[M-4] **string**	String manipulation functions
[M-4] **io**	I/O functions
[M-4] **programming**	Programming functions

What's new

This section is intended for previous Stata users. If you are new to Stata, you may as well skip it.

1. Mata now allows full object-oriented programming! A class is a set of variables, related functions, or both tied together under one name. One class can be derived from another by inheritance. Variables can be public, private, protected, or static. Functions can be public, private, protected, static, or virtual. Members, whether variables or functions, can be final. Classes, member functions, and access to member variables and calls to member functions are fully compiled—not interpreted—meaning there is no speed penalty for casting your program in terms of a class. See [M-2] **class**.

2. The new `moptimize()` suite of functions comprises Stata's new optimization engine used by `ml` and thereby, either directly or indirectly, by nearly all official Stata estimation commands. `moptimize()` provides full support for Stata's new factor variables. See [M-5] **moptimize( )**, [R] **ml**, and [R] **maximize**.

3. The full story is that Stata's `ml` is implemented in terms of Mata's `moptimize()`, which in turn is implemented in terms of Mata's `optimize()`. `optimize()` finds parameters $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ that maximize or minimize $f(p)$. `moptimize()` finds coefficients $\mathbf{b} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)$, where $p_1 = \mathbf{X}_1\mathbf{b}_1, p_2 = \mathbf{X}_2\mathbf{b}_2, \ldots, \mathbf{p}_n = \mathbf{X}_n\mathbf{b}_n$.

   Improvements have been made to `optimize()`:

   a. `optimize()` with constraints is now faster for evaluator types `d0` and `v0` and for all gradient-based techniques. Also, it is faster for evaluator types `d1` and `v1` when used with constraints and with the `nr` (Newton–Raphson) technique.

   b. Gauss–Newton optimization, also known as quadratic optimization, is now available as technique `gn`. Evaluator functions must be of type 'q'.

   c. `optimize()` can now switch between techniques `bhhh`, `nr`, `bfgs`, and `dfp` (between Berndt–Hall–Hall–Hausman, Newton–Raphson, Broyden–Fletcher–Goldfarb–Shanno, and Davidon–Fletcher–Powell).

   d. `optimize()`, when output of the convergence values is requested in the trace log, now displays the identity and value of the convergence criterion that is closest to being met.

   e. `optimize()` has 15 new initialization functions:

```
optimize_init_cluster()              optimize_init_trace_dots()
optimize_init_colstripe()            optimize_init_trace_gradient()
optimize_init_conv_ignorenrtol()     optimize_init_trace_Hessian()
optimize_init_conv_warning()         optimize_init_trace_params()
optimize_init_evaluations()          optimize_init_trace_step()
optimize_init_gnweightmatrix()       optimize_init_trace_tol()
optimize_init_iterid()               optimize_init_trace_value()
optimize_init_negH()
```

   Also, new function `optimize_result_evaluations()` reports the number of times the evaluator is called.

4. Existing functions `st_data()` and `st_view()` now allow the variables to be specified as a string scalar with space-separated names, as well as a string row vector with elements being names. In addition, when a string scalar is used, you now specify either or both time-series–operated variables (e.g., `l.gnp`) and factor variables (e.g., `i.rep78`).

5. New function suite `deriv()` produces numerically calculated first and second derivatives of vector functions; see [M-5] **deriv( )**.

6. Thirty-four LAPACK (Linear Algebra PACKage) functions are now available in as-is form and more are coming. LAPACK is the premier software for solving systems of simultaneous equations, eigenvalue problems, and singular value decompositions. Many of Mata's matrix functions are and have been implemented using LAPACK. We are now in the process of making all the double-precision LAPACK real and complex functions available in raw form for those who want to program their own advanced numerical techniques. See [M-5] **lapack( )** and [R] **copyright lapack**.

7. New function suite `eigensystemselect()` computes the eigenvectors for selected eigenvalues; see [M-5] **eigensystemselect( )**.

8. New function suite `geigensystem()` computes generalized eigenvectors and eigenvalues; see [M-5] **geigensystem( )**.

9. New function suites `hessenbergd()` and `ghessenbergd()` compute the (generalized) Hessenberg decompositions; see [M-5] **hessenbergd( )** and [M-5] **ghessenbergd( )**.

10. New function suites `schurd()` and `gschurd()` compute the (generalized) Schur decompositions; see [M-5] **schurd( )** and [M-5] **gschurd( )**.

11. New function `_negate()` quickly negates a matrix in place; see [M-5] **_negate( )**.

12. New functions `Dmatrix()`, `Kmatrix()`, and `Lmatrix()` compute the duplication matrix, commutation matrix, and elimination matrix used in computing derivatives of functions of symmetric matrices; see [M-5] **Dmatrix( )**, [M-5] **Kmatrix( )**, and [M-5] **Lmatrix( )**.

13. New function `sublowertriangle()` extracts the lower triangle of a matrix, where lower triangle means below a specified diagonal; see [M-5] **sublowertriangle( )**.

14. New function `hasmissing()` returns whether a matrix contains any missing values; see [M-5] **missing( )**.

15. New function `strtoname()` performs the same actions as Stata's `strtoname()` function: it converts a general string to a string meeting the Stata naming conventions. See [M-5] **strtoname( )**.

16. New function `abbrev()` performs the same actions as Stata's `abbrev()` function: it returns abbreviated variable names. See [M-5] **abbrev( )**.

17. New function `_st_tsrevar()` is a handle-the-error-yourself variation of existing function `st_tsrevar()`; see [M-5] **st_tsrevar( )**.

18. Existing functions `ghk()` and `ghkfast()`, which evaluate multivariate normal integrals, have improved syntax; see [M-5] **ghk( )** and [M-5] **ghkfast( )**.

19. Existing functions `vec()` and `vech()` are now faster for both real and complex matrices; see [M-5] **vec( )**.

20. Mata has 13 new distribution-related functions: `hypergeometric()` and `hypergeometricp()`; `nbinomial()`, `nbinomialp()`, and `nbinomialtail()`; `invnbinomial()` and `invnbinomialtail()`; `poisson()`, `poissonp()`, and `poissontail()`; `invpoisson()` and `invpoissontail()`; and `binomialp()`; see [M-5] **normal( )**.

21. Mata has nine new random-variate functions for beta, binomial, chi-squared, gamma, hypergeometric, negative binomial, normal, Poisson, and Student's *t*: `rbeta()`, `rbinomial()`, `rchi2()`, `rgamma()`, `rhypergeometric()`, `rnbinomial()`, `rnormal()`, `rpoisson()`, and `rt()`.

    Also, `rdiscrete()` is provided for drawing from a general discrete distribution.

    Old functions `uniform()` and `uniformseed()` are replaced with `runiform()` and `rseed()`. All random-variate functions start with `r`. See [M-5] **runiform( )**.

22. Existing functions `sinh()`, `cosh()`, `asinh()`, and `acosh()` now have improved accuracy; see [M-5] **sin( )**.

23. New function `soundex()` returns the soundex code for a name and consists of a letter followed by three numbers. New function `soundex_nara()` returns the U.S. Census soundex for a name and also consists of a letter followed by three numbers, but is produced by a different algorithm. See [M-5] **soundex( )**.

24. Existing function `J(`*r*`,` *c*`,` *val*`)` now allows *val* to be specified as a matrix and creates an *r*`*rows(`*val*`)` $\times$ *c*`*cols(`*val*`)` result. The third argument, *val*, was previously required to be 1×1. Behavior in the 1×1 case is unchanged. See [M-5] **J( )**.

25. Existing functions `sort()`, `_sort()`, and `order()` sorted the rows of a matrix based on up to 500 of its columns. This limit has been removed. See [M-5] **sort( )**.

26. New function `asarray()` provides associative arrays; see [M-5] **asarray( )**.

27. New function `hash1()` provides Jenkins' one-at-a-time hash function; see [M-5] **hash1( )**.

28. Mata object-code libraries (`.mlib`s) may now contain up to 2,048 functions and may contain up to 1,024 by default. Use `mlib create`'s new `size()` option to change the default. The previous fixed maximum was 500. See [M-3] **mata mlib**.

29. Mata on 64-bit computers now supports matrices larger than 2 gigabytes when the computer has sufficient memory.

(*Continued on next page*)

30. One hundred and nine existing functions now take advantage of multiple cores when using Stata/MP. They are

acos()	factorial()	mm()
arg()	Fden()	mmC()
asin()	floatround()	mod()
atan2()	floor()	mofd()
atan()	Ftail()	month()
betaden()	gammaden()	msofhours()
binomial()	gammap()	msofminutes()
binomialtail()	gammaptail()	msofseconds()
binormal()	halfyear()	nbetaden()
ceil()	hh()	nchi2()
chi2()	hhC()	nFden()
chi2tail()	hofd()	nFtail()
Cofc()	hours()	nibeta()
cofC()	ibeta()	normal()
Cofd()	ibetatail()	normalden()
cofd()	invbinomial()	npnchi2()
comb()	invbinomialtail()	qofd()
cos()	invchi2()	quarter()
day()	invchi2tail()	round()
dgammapda()	invF()	seconds()
dgammapdada()	invFtail()	sin()
dgammapdadx()	invgammap()	sqrt()
dgammapdx()	invgammaptail()	ss()
dgammapdxdx()	invibeta()	tan()
digamma()	invibetatail()	tden()
dofC()	invnchi2()	trigamma()
dofc()	invnFtail()	trunc()
dofd()	invnibeta()	ttail()
dofh()	invnormal()	week()
dofm()	invttail()	wofd()
dofq()	ln()	year()
dofw()	lnfactorial()	yh()
dofy()	lngamma()	ym()
dow()	lnnormal()	yq()
doy()	lnnormalden()	yw()
exp()	mdy()	
F()	minutes()	

Also see

[M-1] **first** — Introduction and first session

[M-6] **Glossary**