Computation

Visualization

Programming

New Features Guide

Version 5

How to Contact The MathWorks:

(508) 647-7000 Phone

(508) 647-7001 Fax

(508) 647-7022 Technical Support Faxback Server

The MathWorks, Inc.
24 Prime Park Way
Natick, MA 01760-1500 Mail

`http://www.mathworks.com` Web
`ftp.mathworks.com` Anonymous FTP server

`support@mathworks.com` Technical support
`suggest@mathworks.com` Product enhancement suggestions
`bugs@mathworks.com` Bug reports
`doc@mathworks.com` Documentation error reports
`subscribe@mathworks.com` Subscribing user registration
`service@mathworks.com` Order status, license renewals, passcodes
`info@mathworks.com` Sales, pricing, and general information

New Features Guide (July 1996)

Printing History: January 1996 First printing New for Alpha-1
July 1996 Second printing Revised for Alpha-7

Contents

2 Upgrading to MATLAB 5

List of Tables

Introduction

MATLAB 5 is a significant new release of MATLAB. We've been listening to your requests for new features – via telephone, e-mail, and at conferences – for the past several years, and have carefully used them to design this new version. We have combined these requests with exciting innovations to bring you MATLAB 5.

This booklet

- Describes new features and enhancements in MATLAB 5.
- Provides guidelines for upgrading from MATLAB 4 to MATLAB 5.

If you're familiar with MATLAB 4, you should read this guide first. Go on to *Using MATLAB* and *Using MATLAB Graphics* for more details on any new feature. If you encounter a new MATLAB function in this book and want to learn more, consult the online *MATLAB Function Reference* . (See "MATLAB Documentation" below.)

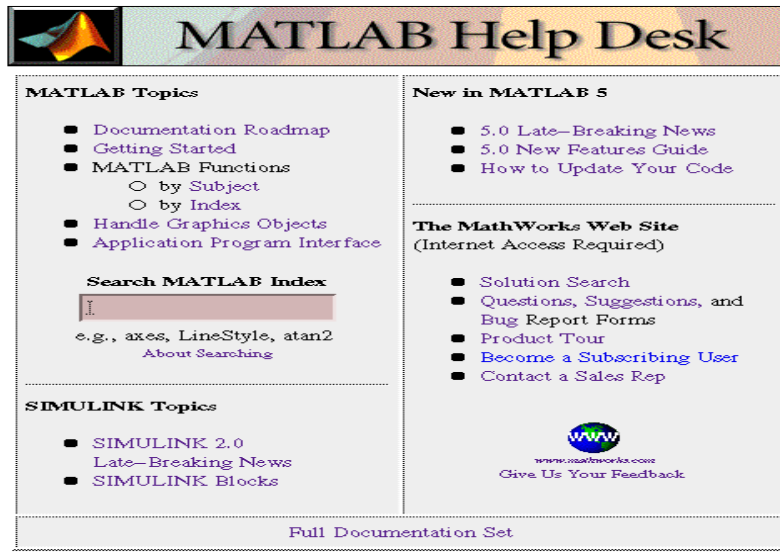
If you are a new MATLAB user, you should begin with *Getting Started with MATLAB*, which introduces you to MATLAB's capabilities as a programming and visualization language.

Help Desk

MATLAB 5 includes the MATLAB Help Desk, an enhanced Help facility that provides access to online help topics, online reference materials, electronic documentation, and World Wide Web pages through a Web browser. You do not need to be connected to the Internet to use this facility.

The Help Desk is optimized for the use of Netscape Navigator.

On all platforms you can access this facility via the `helpdesk` command. On the PC and Macintosh you can additionally access this facility via the **Help** menu or the **?** icon on the Command Window toolbar.



Related MATLAB Commands

- `help` displays MATLAB help text in the command window.
- `helpwin` opens a window and displays MATLAB help text.

MATLAB Documentation

The MATLAB documentation set has been rewritten, expanded, and divided into several volumes for ease of use. The set includes on-line help, as well as hypertext-based and printed manuals. The online *MATLAB Function Reference* is a compendium of all MATLAB language, mathematical, and graphics functions. You can access this documentation from the MATLAB Help Desk. Choose “MATLAB Functions ” to display the *Function Reference.*

The online documentation is augmented with a full set of printed documents, consisting of the following titles:

- *Getting Started with MATLAB*, an introductory document describing the fundamentals of MATLAB.
- *Using MATLAB*, which explains how to use MATLAB as both a programming language and a command-line application.
- *Using MATLAB Graphics*, which describes how to use MATLAB's graphics and visualization tools.
- *MATLAB Application Program Interface Guide*, which explains how to write C or Fortran programs that interact with MATLAB.
- *MATLAB 5 New Features Guide*, which summarizes new features and provides information useful in making the transition from MATLAB 4 to MATLAB 5.
- *MATLAB Installation Guide*, which describes how to install MATLAB on your platform.
- *Building GUIs with MATLAB*, which describes Guide, a Graphical User Interface (GUI) design tool.
- *MATLAB Notebook User's Guide*, which describes the use of Microsoft Word as an interface to MATLAB.
- *MATLAB Late-Breaking News*, which contains information that became available after the preparation of the rest of the documentation set.

If one or more of the printed documents is unavailable to you, you can locate an online version of the same document via the Help Desk.

Additionally, command line ASCII help and an extensive library of demonstration programs provide instant online reference information about MATLAB commands and demonstrate MATLAB features.

1

New Features and Enhancements

What's New in MATLAB?

MATLAB, the language of technical computing, has been designed to increase the scope and productivity of science and engineering, to accelerate the pace of discovery and development, to facilitate learning, and to amplify the creativity of research. MATLAB 5, the newest version of the MATLAB environment, vastly enhances programmer productivity, providing many ease-of-use/ease-of-learning features that enable the rapid development of larger and more complex applications. Befitting its name, MATLAB 5 features five major areas of programming innovation:

- Enhanced programming and application development tools
- New data types, structures, and language features
- Faster, better graphics and visualization
- More mathematics and data analysis firepower
- Major enhancements to the MATLAB application toolbox suite and to SIMULINK

Enhanced Programming and Application Development Tools

MATLAB 5 provides new M-file programming enhancements and application development tools that make it easier than ever to develop and maintain applications in MATLAB. Highlights include:

- Integrated M-file editor
- Visual M-file debugger
- M-file performance profiler
- Search path browser/editor
- Workspace browser
- Web-based online Help Desk/documentation viewer
- GUI builder
- Handle Graphics property editor
- Pre-parsed P-code files (P-files)
- Enhanced, self-diagnosing Application Program Interface (API)

New Data Types, Structures, and Language Features

MATLAB 5 introduces new data types and language improvements. These new features make it easy to build much larger and more complex MATLAB applications.

- Multi-dimensional arrays
- User-definable data structures
- Cell arrays: multi-type data arrays
- Character arrays: two bytes per character
- Single byte data type for images
- Object-oriented programming
- Variable-length argument lists
- Multifunction and private M-files
- Function and operator overloading
- `switch`/`case` statements

Faster, Better Graphics and Visualization

Graphics take another quantum leap with powerful new visualization techniques and significantly faster image display using the Z-buffer algorithm. Presentation graphics are also improved to give you more options and control over how you present your data.

- Visualization
  - Truecolor (RGB) support
  - Fast and accurate Z-buffer display algorithm
  - Flat, Gouraud, and Phong lighting
  - Vectorized patches for three dimensional modeling
  - Camera view model, perspective, fly throughs
  - Efficient 8-bit image display
  - Image file import/export

- Presentation Graphics
  - Greek symbols, sub/superscripts, multiline text
  - Dual axis plots
  - Three dimensional quiver, ribbon, and stem plots
  - Pie charts, three dimensional bar charts
  - Extended curve marker symbol family

More Mathematics and Data Analysis Firepower

- New ordinary differential equation solvers (ODEs)
- Delaunay triangulation
- Gridding for irregularly-spaced data
- Set theory functions
- Two-dimensional quadrature
- Time and date handling functions
- Multi-dimensional interpolation, convolution, FFT
- Bit-wise operators
- Iterative sparse methods
- Sparse matrix eigenvalues and singular values

Enhancements to Application Toolbox Suite and to SIMULINK

- SIMULINK 2.0
- Image Processing Toolbox 2.0
- Control System Toolbox 4.0
- Signal Processing Toolbox 4.0
- Symbolic Math Toolbox 2.0
- Many more to come ...

New Data Constructs

MATLAB 5 supports these new data constructs:

- Multidimensional arrays
- Cell arrays
- Structures

In addition, MATLAB 5 features an improved storage method for string data.

Multidimensional Arrays

Arrays (other than sparse matrices) are no longer restricted to two dimensions. You can create and access arrays with two or more dimensions by

- Using MATLAB functions like `zeros`, `ones`, or `rand`
- Using the new `cat` function
- Using the `repmat` function

MATLAB functions like `zeros`, `ones`, and `rand` have been extended to accept more than two dimensions as arguments. To create a 3-by-4-by-5 array of ones, for example, use

```
A = ones(3,4,5)
```

The new `cat` function enables you to concatenate arrays along a specified dimension. For example, create two rectangular arrays `A` and `B`:

```
A = [1 2 3; 4 5 6];
B = [6 2 0; 9 1 3];
```

To concatenate these along the third dimension:

```
C = cat(3,A,B)

C(:,:,1) =

     1     2     3
     4     5     6


C(:,:,2) =

     6     2     0
     9     1     3
```

You can also create an array with two or more dimensions in which every element has the same value using the `repmat` function. `repmat` accepts the value with which to fill the array, followed by a vector of dimensions for the array. For example, to create a 2-by-2-by-3-by-3 array `B` where every element has the value `pi`:

```
B = repmat(pi,[2 2 3 3]);
```

You can also use `repmat` to replicate or "tile" arrays in a specified configuration.

Table 1-1: New Multidimensional Array Functions

Function	Description
`cat`	Concatenate arrays.
`flipdim`	Flip array along specified dimension.
`ind2sub`	Subscripts from linear index.
`ipermute`	Inverse permute the dimensions of a multidimensional array.
`ndgrid`	Generate arrays for multidimensional functions and interpolation.
`ndims`	Number of array dimensions.

Table 1-1: New Multidimensional Array Functions (Continued)

Function	Description
`permute`	Rearrange the dimensions of a multidimensional array.
`reshape`	Change size.
`shiftdim`	Shift dimensions.
`squeeze`	Remove singleton array dimensions.
`sub2ind`	Single index from subscripts.

Cell Arrays

Cell arrays have elements that are containers for any type of MATLAB data, including other cells. You can build cell arrays using assignment statements (for instance, `A(2,2) = {'string'}`), or by using the new `cell` function.

Table 1-2: New Cell Array Functions

Function	Description
`cell`	Create cell array.
`cell2struct`	Cell array to structure array conversion.
`celldisp`	Display top-level structure of cell array.
`cellplot`	Graphically display the structure of a cell array.
`num2cell`	Convert a matrix into a cell array.

Structures

Structures are constructs that have named fields containing any kind of data. For example, one field might contain a text string representing a name (`patient.name = 'Jane Doe'`), another might contain a scalar representing a billing amount (`patient.billing = 127.00`), and a third might hold a matrix of medical test results. You can organize these structures into arrays of data.

Create structure arrays by using individual assignment statements or the new `struct` function.

Table 1-3: New Structure Functions

Function	Description
`fields`	Field names of structure array.
`getfield`	Get field of structure array.
`rmfield`	Remove structure fields.
`setfield`	Set field of structure array.
`struct`	Create structure array.
`struct2cell`	Structure to cell array conversion.

Character Arrays

Strings now take up less memory than they did in previous releases. MATLAB 4 required 64 bits per character for string data. MATLAB 5 requires only 16 bits per character.

Table 1-4: New Character String Functions

Function	Description
`base2dec`	Base `B` to decimal number conversion.
`bin2dec`	Binary to decimal number conversion.
`char`	Convert numeric values to string.
`dec2base`	Decimal number to base conversion.
`dec2bin`	Decimal to binary number conversion.
`mat2str`	Convert a matrix into a string.
`strcat`	String concatenation.
`strmatch`	Find possible matches for a string.

Table 1-4: New Character String Functions (Continued)

Function	Description
`strncmp`	Compare the first n characters of two strings.
`strvcat`	Vertical concatenation of strings.

The MATLAB programming language does not require the use of data types. For many applications, however, it is helpful to associate specific attributes with certain categories of data. To facilitate this, MATLAB allows you to work with *objects*. Objects are typed structures. A single *class* name identifies both the type of the structure and the name of the function that creates objects belonging to that class.

Objects differ from ordinary structures in two important ways:

Data hiding. The structure fields of objects are not visible from the command line. Instead, you can access structure fields only from within a *method*, an M-file associated with the object class. Methods reside in class directories. Class directories have the same name as the class, but with a prepended `@` symbol. For example, a class directory named `@inline` might contain methods for a class called `inline` .

Function and expression overloading. You can create methods that override existing M-files. If an object calls a function, MATLAB first checks to see if there is a method of that name before calling a supplied M-file of that name. You can also provide methods that are called for MATLAB operators. For objects `a` and `b`, for instance, the expression `a + b` calls the method `plus(a,b)` if it exists.Programming Capabilities

MATLAB 5 includes flow-control improvements and new M-file programming tools.

Flow Control Improvements

MATLAB 5 features:

- A new flow control statement, the `switch` statement
- More efficient evaluation of logical operators

The switch statement is a convenient way to execute code conditionally when you have many possible cases to choose from. It is no longer necessary to use a series of elseif statements:

```
switch input_num
    case –1
        disp('negative one');
    case 0
        disp('zero');
    case 1
        disp('positive one');
    otherwise
         disp('other value');
end
```

Only the first matching case is executed.

switch can handle multiple conditions in a single case statement by enclosing the case expression in a cell array. For example, assume method exists as a string variable:

```
switch lower(method)
    case {'linear','bilinear'}, disp('Method is linear')
    case 'cubic', disp('Method is cubic')
    case 'nearest', disp('Method is nearest')
    otherwise, disp('Unknown method.')
end
```

Table 1-5: New Flow Control Commands

Command	Description
case	Case switch.
dbmex	Enable MEX-file debugging.
errortrap	Skip errors during testing.
otherwise	Default part of switch statement.
switch	Conditionally execute code, switching among several cases.

MATLAB now evaluates expressions involving logical operators more efficiently than before. For example, consider the expression `if a|b`. If `a` is true, then MATLAB will not evaluate `b`. Similarly, MATLAB won't execute statements following the expression `if a&b` in the event `a` is found to be false.

Table 1-6: New Logical Operators

Operator	Description
`iscell`	True for a cell array.
`isequal`	True if arrays are equal.
`isfinite`	True for finite elements.
`islogical`	True for logical arrays.
`isnumeric`	True if input is a numeric array.
`isprime`	True for prime numbers
`isspace`	True for space, newline, carriage return, tab, vertical tab, or formfeed.
`isstruct`	True for a structure.
`logical`	Convert numeric values to logical vectors.

M-File Programming Tools

MATLAB 5 adds three features to enhance MATLAB's M-file programming capabilities.

Variable Number of Input and Output Arguments

The `varargin` and `varargout` commands simplify the task of passing data into and out of M-file functions. For instance, the statement `function varargout = myfun(A,B)` allows M-file `myfun` to return an arbitrary number of output arguments, while the statement `function [C,D] = myfun(varargin)` allows it to accept an arbitrary number of input arguments.

Multiple Functions Within an M-File

It is now possible to have subfunctions within the body of an M-file. These are functions that the primary function in the file can access but that are otherwise invisible.

M-File Profiler

This utility lets you debug and optimize M-files by tracking cumulative execution time for each line of code. Whenever the specified M-file executes, the profiler counts how many time intervals each line uses.

Pseudocode M-Files

The pcode command saves a pseudocode version of a function or script to disk for later sessions. This *pseudocode* version is ready-to-use code that MATLAB can access whenever you invoke the function. In some cases, this reduces the time it takes to execute a function.

Table 1-7: New Programming Tools

Function	Description
`addpath`	Append directory to MATLAB's search path.
`applescript`	Load a compiled AppleScript from a file and execute it.
`assignin`	Assign variable in workspace.
`edit`	Edit an M-file.
`editpath`	Modify current search path.
`evalin`	Evaluate variable in workspace.
`fullfile`	Build full filename from parts.
`inmem`	Functions in memory.
`inputname`	Input argument name.

Table 1-7: New Programming Tools (Continued)

Function	Description
`mfilename`	Name of the currently running M-file.
`mexext`	Return the MEX filename extension.
`pcode`	Create pseudo-code file (P-file).
`profile`	Measure and display M-file execution profiles.
`rmpath`	Remove directories from MATLAB's search path.
`varargin`, `varargout`	Pass or return variable numbers of arguments.
`warning`	Display warning message.
`web`	Point web browser at file or web site.

New and Enhanced Language Functions

MATLAB 5 provides a large number of new language functions as well as enhancements to existing functions.

Table 1-8: New Elementary and Specialized Math Functions

Function	Description
`airy`	Airy functions.
`besselh`	Bessel functions of the third kind (Hankel).
`condeig`	Condition number with respect to eigenvalues.
`condest`	1-norm matrix condition estimate.
`dblquad`	Numerical double integration
`mod`	Modulus (signed remainder after division).
`normest`	2-norm estimate.

Table 1-9: New Time and Date Functions

Function	Description
`calendar`	Calendar.
`datenum`	Serial date number.
`datestr`	Create date string.
`datetick`	Date formatted tick labels.
`datevec`	Date components.
`eomday`	End of month.
`now`	Current date and time.
`weekday`	Day of the week.

Table 1-10: New Ordinary Differential Equation Functions

Function	Description
`ode45`, `ode23`, `ode113`, `ode23s`, `ode15s`	Solve differential equations, low and high order methods.
`odefile`	Define a differential equation problem for ODE solvers.
`odeget`	Extract options from an argument created with odeset.
`odeset`	Create and edit input arguments for ODE solvers.

Table 1-11: New Matrix Functions

Function	Description
`cholinc`	Incomplete Cholesky factorization.
`gallery`	More than 50 new test matrices.
`luinc`	Incomplete LU factorization.
`repmat`	Replicate and tile an array.
`sprand`	Random uniformly distributed sparse matrices.

Table 1-12: New Methods for Sparse Matrices

Method	Description
`bicg`	BiConjugate Gradients method.
`bicgstab`	BiConjugate Gradients Stabilized method.
`cgs`	Conjugate Gradients Squared method.
`eigs`	Find a few eigenvalues and eigenvectors.
`gmres`	Generalized Minimum Residual method.

Table 1-12: New Methods for Sparse Matrices (Continued)

Method	Description
pcg	Preconditioned Conjugate Gradients method.
qmr	Quasi-Minimal Residual method.
svds	A few singular values.

Subscripting and Assignment Enhancements

In MATLAB 5, you can now:

- Access the last element of an array using the end keyword.
- Obtain consistent results for indexing expressions consisting of all ones.
- Use scalar expansion in subarray assignments.

A statement like A(ones([m,n])) now always returns an m-by-n array in which each element is A(1). In previous versions, the statement returned different results depending on whether A was or was not an m-by-n matrix.

In previous releases, expressions like A(2:3,4:5) = 5 resulted in an error. MATLAB 5 automatically "expands" the 5 to be the right size (that is, 5*ones(2,2)).

Integer Bit Manipulation Functions

The bitfun directory contains commands that permit bit-level operations on integers. Operations include setting and unsetting, complementing, shifting, and logical AND, OR, and XOR.

Table 1-13: New Bitwise Functions

Function	Description
bitand	Bitwise AND.
bitcmp	Complement bits.
bitmax	Maximum floating-point integer.
bitor	Bitwise OR.

Table 1-13: New Bitwise Functions (Continued)

Function	Description
bitset	Set bit.
bitshift	Bitwise shift.
bittest	Test bit.
bitxor	Bitwise XOR.

Dimension Specification for Data Analysis Functions

MATLAB's basic data analysis functions now enable you to supply a second input argument. This argument specifies the dimension along which the function operates. For example, create an array A:

```
A = [3 2 4; 1 0 5; 8 2 6];
```

To sum along the first dimension of A, incrementing the row index, specify 1 for the dimension of operation:

```
sum(A,1)

ans =

    12     4    15
```

To sum along the second dimension, incrementing the column index, specify 2 for the dimension:

```
sum(A,2)

ans =

     9
     6
    16
```

Other functions that accept the dimension specifier include prod, cumprod, and cumsum.

Wildcards in Utility Commands

The asterisk (*) can be used as a wildcard in the `clear` and `whos` commands. This allows you, for example, to clear only variables beginning with a given character or characters, as in

```
clear A*
```

Empty Arrays

Earlier versions of MATLAB allowed for only one empty matrix, the 0-by-0 matrix denoted by `[]`. MATLAB 5 provides for matrices and arrays in which one, but not all, of the dimensions is zero. For example, `1-by-0`, `10-by-0-by-20`, and `[3 4 0 5 2]` are all possible array sizes.

The two-character sequence `[]` continues to denote the 0-by-0 matrix. Empty arrays of other sizes can be created with the functions `zeros`, `ones`, `rand`, or `eye`. To create a 0-by-5 matrix, for example, use

```
E = zeros(0,5)
```

The basic model for empty matrices is that any operation that is defined for `m`-by-`n` matrices, and that produces a result whose dimension is some function of `m` and `n`, should still be allowed when `m` or `n` is zero. The size of the result should be that same function, evaluated at zero.

For example, horizontal concatenation

```
C = [A B]
```

requires that `A` and `B` have the same number of rows. So if `A` is `m`-by-`n` and `B` is `m`-by-`p`, then `C` is `m`-by-`(n+p)`. This is still true if `m` or `n` or `p` is zero.

Many operations in MATLAB produce row vectors or column vectors. It is now possible for the result to be the empty row vector

```
r = zeros(1,0)
```

or the empty column vector

```
c = zeros(0,1)
```

MATLAB 5 retains MATLAB 4 behavior for `if` and `while` statements. For example

```
if A, S1, else, S0, end
```

will execute statement `S0` when `A` is an empty array.

Some MATLAB functions, like `sum` and `max`, are *reductions*. For matrix arguments, these functions produce vector results; for vector arguments they produce scalar results. Backwards compatibility issues arise for the argument `[]`, which in MATLAB 4 played the role of both the empty matrix and the empty vector. In MATLAB 5, empty inputs with these functions produce these results:

- `sum([])` is `0`
- `prod([])` is `1`
- `max([])` is `[]`
- `min([])` is `[]`

New Data Analysis Features

MATLAB 5 provides an expanded set of basic data analysis functions.

Table 1-14: New Statistical Data Analysis Functions

Function	Description
`convhull`	Convex hull.
`cumtrapz`	Cumulative trapezoidal numerical integration.
`delaunay`	Delaunay triangularization.
`dsearch`	Search for nearest point.
`factor`	Prime factors.
`inpolygon`	Detect points inside a polygonal region.
`nchoosek`	All possible combinations of `n` elements taken `k` at a time.
`perms`	All possible permutations.
`polyarea`	Area of polygon.
`primes`	Generate a list of prime numbers.
`sortrows`	Sort rows in ascending order.
`tsearch`	Search for enclosing Delaunay triangle.
`voronoi`	Voronoi diagram.

MATLAB 5 also offers expanded data analysis in the areas of:

- Higher-dimension interpolation
- Extended `griddata` functionality based on Delaunay triangulation
- New set theoretic functions

Higher-Dimension Interpolation

The new functions `interp3` and `interpn` let you perform three-dimensional and multidimensional interpolation. `ndgrid` provides arrays that can be used in multidimensional interpolation.

Table 1-15: New Interpolation Functions

Function	Description
`interp3`	Three-dimensional data interpolation (table lookup).
`interpn`	Multidimensional data interpolation (table lookup).
`ndgrid`	Generate arrays for multidimensional functions and interpolation.

griddata Based on Delaunay Triangulation

`griddata` supports triangle-based interpolation using nearest neighbor, linear, and cubic techniques. It creates smoother contours on scattered data using the `cubic` interpolation method.

Set Theoretic Functions

The functions `union`, `intersect`, `ismember`, `setdiff`, and `unique` treat vectors as sets, allowing you to perform operations like union (A∪B), intersection (A∩B), and difference (A–B) of such sets. Other set-theoretical operations include location of common set elements (`ismember`) and elimination of duplicate elements (`unique`).

Table 1-16: New Set Functions

Function	Description
`intersect`	Set intersection of two vectors.
`ismember`	Detect members of a set.
`setdiff`	Return the set difference of two vectors.
`setxor`	Set `XOR` of two vectors.

Table 1-16: New Set Functions (Continued)

Function	Description
`union`	Set union of two vectors.
`unique`	Unique elements of a vector.

New and Enhanced Handle Graphics Features

MATLAB 5 features significant improvements to Handle Graphics. For details on all graphics functions, see *Using MATLAB Graphics.*

Plotting Capabilities

MATLAB's basic plotting capabilities have been improved and expanded in MATLAB 5.

Table 1-17: New and Enhanced Plotting Capabilities

Function	Description
`area`	Filled area plot.
`bar3`	Vertical 3-D bar chart.
`bar3h`	Horizontal 3-D bar chart.
`barh`	Horizontal bar chart.
`gplot`	"Graph theoretic" graph.
`pie`	Pie chart.
`pie3`	Three-dimensional pie chart.
`plotyy`	Plot graphs with Y tick labels on left and right.
`stem3`	Three-dimensional stem plot.

area Function

The `area` function plots a set of curves and fills the area beneath the curves.

Bar Chart Enhancements

`bar3`, `bar3h`, and `barh` draw vertical and horizontal bar charts. These functions, together with `bar`, support multiple filled bars in grouped and stacked formats.

legend Enhancement

`legend` can label any solid-color patch and surface. You can now place legends on line, bar, ribbon, and pie plots, for example.

Table 1-18: New Graph Annotation Functions

Function	Description
`box`	Axes box.
`datetick`	Display dates for Axes tick labels.

Marker Style Enhancement

A number of new line markers are available, including, among others, a square, a diamond, and a five-pointed star. These can be specified independently from line style.

Stem Plot Enhancements

`stem` and `stem3` plot discrete sequence data as filled or unfilled stem plots.

Three-Dimensional Plotting Support

`quiver3` displays three-dimensional velocity vectors with (u,v,w) components. The `ribbon` function displays data as three-dimensional strips.

Table 1-19: New Three-Dimensional Plotting Functions

Function	Description
`quiver3`	Three-dimensional quiver plot.
`ribbon`	Draw lines as 3-D strips.
`rotate3d`	Three-dimensional rotation using the mouse.

Data Visualization

MATLAB 5 features many new and enhanced capabilities for data visualization.

New Viewing Model

Axes camera properties control the orthographic and perspective view of the scene created by an Axes and its child objects. You can view the Axes from any

location around or in the scene, as well as adjust the rotation, view angle, and target point.

New Method for Defining Patches

You can define a Patch using a matrix of faces and a matrix of vertices. Each row of the face matrix contains indices into the vertex matrix to define the connectivity of the face. Defining Patches in this way reduces memory consumption because you no longer need to specify redundant vertices.

Triangular Meshes and Surfaces

The new functions `trimesh` and `trisurf` create triangular meshes and surfaces from `x`, `y`, and `z` vector data and a list of indices into the vector data.

Table 1-20: New Triangular Mesh and Surface Functions

Function	Description
`trisurf`	Triangular surface plot.
`trimesh`	Triangular mesh plot.

Improved Slicing

`slice` now supports an arbitrary slicing surface.

Contouring Enhancements

The contouring algorithm now supports parametric surfaces and contouring on triangular meshes. In addition, `clabel` rotates and inserts labels in contour plots.

Table 1-21: New Contour Plot

Function	Description
`contourf`	Filled contour plot.

New zoom Options

The `zoom` function supports two new options:

- *scale_factor* – zooms by the specified scale factor relative to the current zoom state (e.g., `zoom(2)` zooms in by a factor of two).
- `fill` – zooms to the point where the objects contained in the Axes are as large as they can be without extending beyond the Axes plot box from any view. Use this option when you want to rotate the Axes without seeing an apparent size change.

Graphics Presentation

MATLAB 5 provides improved control over the display of graphics objects.

Enhancements to Axes Objects

MATLAB 5 provides more advanced control for three-dimensional Axes objects. You can control the three-dimensional aspect ratio for the Axes' plot box, as well as for the data displayed in the plot box. You can also zoom in and out from a three-dimensional Axes using viewport scaling and Axes camera properties.

The `axis` command supports a new option designed for viewing graphics objects in 3-D:

```
axis vis3d
```

This option prevents MATLAB from stretching the Axes to fit the size of the Figure window and otherwise altering the proportions of the objects as you change the view.

In a two-dimensional view, you can display the *x*-axis at the top of an Axes and the *y*-axis at the right side of an Axes.

Color Enhancements

`colordef white` or `colordef black` changes the color defaults on the root so that subsequent figures produce plots with a white or black axes background color. The figure background color is changed to be a shade of gray, and many other defaults are changed so that there will be adequate contrast for most

plots. `colordef none` sets the defaults to their MATLAB 4 values. In addition, a number of new colormaps are available.

Table 1-22: New Figure and Axis Color Control

Function	Description
`colordef`	Select Figure color scheme.

Table 1-23: New Colormaps

Function	Description
`autumn`	Shades of red and yellow colormap.
`colorcube`	Regularly spaced colors in RGB colorspace that provide more steps of gray, pure red, pure green, and pure blue.
`lines`	Colormap of colors specified by the Axes' `ColorOrder` property.
`spring`	Shades of magenta and yellow colormap.
`summer`	Shades of green and yellow colormap.
`winter`	Shades of blue and green colormap.

Text Object Enhancements

MATLAB 5 supports a subset of LaTex commands. A single Text graphics object can support multiple fonts, subscripts, superscripts, and Greek symbols. See the `text` function in the online *MATLAB Function Reference* for information about the supported LaTex subset.

You can also specify multiline character strings and use normalized font units so that Text size is a fraction of an Axes' or Uicontrol's height. MATLAB supports multiline text strings using cell arrays. Simply define a string variable as a cell array with one line per cell.

Improved General Graphics Features

The MATLAB startup file sets default properties for various graphics objects so that new Figures are aesthetically pleasing and graphs are easier to understand.

Table 1-24: New Figure Window Creation and Control Commands

Command	Description
`dialog`	Create a dialog box.
`hgmenu`	Display default **File** and **Edit** menus for Figures.

Z-buffering is now available for fast and accurate three-dimensional rendering.

Lighting

MATLAB supports a new graphics object called a Light. You create a Light object using the `light` function. Three important Light object properties are:

- `Color` – the color of the light cast by the Light object
- `Mode` – either infinitely far away (the default) or local
- `Position` – the direction (for infinite light sources) or the location (for local light sources)

You cannot see Light objects themselves, but you can see their effect on any Patch and Surface objects present in the same Axes. You can control these effects by setting various Patch and Surface object properties – `AmbientStrength`, `DiffuseStrength`, and `SpecularStrength` control the intensity of the respective light-reflection characteristics; `SpecularColorReflectance` and `SpecularExponent` provide additional control over the reflection characteristics of specular light.

The Axes `AmbientLightColor` property determines the color of the ambient light, which has no direction and affects all objects uniformly. Ambient light effects occur only when there is a visible Light object in the Axes.

The Light object's `Color` property determines the color of the directional light, and its `Mode` property determines whether the light source is a point source (`Mode` set to `local`), which radiates from the specified position in all directions, or a light source placed at infinity (`Mode` set to `infinite`), which shines from the direction of the specified position with parallel rays.

You can also select the algorithm used to calculate the coloring of the lit objects. The Patch and Surface `EdgeLighting` and `FaceLighting` properties select between no lighting, and flat, Gouraud, or Phong lighting algorithms.

print Command Revisions

The `print` command has been extensively revised for MATLAB 5. Consult *Using MATLAB Grapics* for a complete description of `print` command capabilities. Among the new options available for MATLAB 5:

- The `-loose` option makes the PostScript bounding box equal to the Figure's `PaperPosition` property. PICT (Macintosh) and EPSI (X) previews are the same size as the generated PostScript drawing.
- Z-buffer images may be printed at user-selectable resolution.
- The `print` function can generate an M-file that recreates a Figure.
- Uicontrol objects print by default unless suppressed with the `-noui` option. In earlier versions of MATLAB, Uicontrols did not appear when you printed Figures. If you specify the `-noui` option with the `print` command, MATLAB ignores Uicontrols and prints only Axes and Axes children.

Additional print Device Options

The `print` command has several new device options:

Table 1-25: print Command Device Options

Device	Description
`-dljet4`	HP LaserJet 4 (defaults to 600 dpi)
`-ddeskjet`	HP DeskJet and DeskJet Plus
`-ddjet500`	HP Deskjet 500
`-dcdeskjet`	HP DeskJet 500C with 1 bit/pixel color
`-dcdj500`	HP DeskJet 500C
`-dcdj550`	HP Deskjet 550C
`-dpjxl`	HP PaintJet XL color printer
`-dpjxl300`	HP PaintJet XL300 color printer

Table 1-25: print Command Device Options (Continued)

Device	Description
-ddnj650c	HP DesignJet 650C
-dbj200	Canon BubbleJet BJ200
-dbjc600	Canon Color BubbleJet BJC-600 and BJC-4000
-depsonc	Epson LQ-2550 and Fujitsu 3400/2400/1200
-dibmpro	IBM 9-pin Proprinter
-dtiffpack	TIFF PackBits (tag = 32773) (monochrome)
-dbmp256	8-bit (256-color) BMP file format
-dbmp16m	24-bit BMP file format
-dpcxmono	Monochrome PCX file format
-dpcx24b	24-bit color PCX file format, three 8-bit planes
-dpbm	Portable Bitmap (plain format)
-dpbmraw	Portable Bitmap (raw format)
-dpgm	Portable Graymap (plain format)
-dpgmraw	Portable Graymap (raw format)
-dppm	Portable Pixmap (plain format)
-dppmraw	Portable Pixmap (raw format)
-dbit	A plain “bit bucket” device
-dbitrgb	Plain bits, RGB
-dbitcmyk	Plain bits, CMYK

Image Support

MATLAB 5 provides a number of enhancements to image support. These enhancements include:

- Truecolor support
- New functions for reading images from and writing images to graphics files
- 8-bit image support

Truecolor

In addition to indexed images, in which colors are stored as an array of indices into a colormap, MATLAB 5 now supports truecolor images. A truecolor image does not use a colormap; instead, the color values for each pixel are stored directly as RGB triplets. In MATLAB, the `CData` property of a truecolor Image object is a three-dimensional (`m`-by-`n`-by-3) array. This array consists of three `m`-by-`n` matrices (representing the red, green, and blue color planes) concatenated along the third dimension.

Reading and Writing Images

The `imread` function reads image data into MATLAB arrays from graphics files in various standard formats, such as TIFF. You can then display these arrays using the `image` function, which creates a Handle Graphics® Image object. You can also write MATLAB image data to graphics files using the `imwrite` function. `imread` and `imwrite` both support a variety of graphics file formats and compression schemes.

8-Bit Images

When you read an image into MATLAB using `imread`, the data is stored as an array of 8-bit integers. This is a much more efficient storage method than the double-precision (64-bit) floating-point numbers that MATLAB typically uses.

The Handle Graphics Image object has been enhanced to support 8-bit `CData`. This means you can display 8-bit images without having to convert the data to double precision. MATLAB 5 also supports a limited set of operations on these 8-bit arrays. You can view the data, reference values, and reshape the array in various ways. To perform any mathematical computations, however, you must first convert the data to double precision, using the `double` function.

Note that, in order to support 8-bit images, certain changes have been made in the way MATLAB interprets image data. This table summarizes the conventions MATLAB uses:

Image Type	Double-Precision Data (Double Array)	8-Bit Data (uint8 Array)
Indexed (colormap)	Image is stored as a 2-D (m-by-n) array of integers in the range [1,length(colormap)]; colormap is an m-by-3 array of floating-point values in the range [0, 1]	Image is stored as a 2-D (m-by-n) array of integers in the range [0, 255]; colormap is an m-by-3 array of floating-point values in the range [0, 1]
Truecolor (RGB)	Image is stored as a 3-D (m-by-n-by-3) array of floating-point values in the range [0, 1]	Image is stored as a 3-D (m-by-n-by-3) array of integers in the range [0, 255]

Note that MATLAB interprets image data very differently depending on whether it is double precision or 8-bit. The rest of this section discusses things you should keep in mind when working with image data to avoid potential pitfalls. This information is especially important if you want to convert image data from one format to another.

Indexed images

In an indexed image of class `double`, the value 1 points to the first row in the colormap, the value 2 points to the second row, and so on. In a `uint8` indexed image, there is an offset; the value 0 points to the first row in the colormap, the value 1 points to the second row, and so on. The `uint8` convention is also used in graphics file formats, and enables 8-bit indexed images to support up to 256 colors. Note that when you read in an indexed image with `imread`, the resulting image array is always of class `uint8`. (The colormap, however, is of class `double`; see below.)

If you want to convert a `uint8` indexed image to `double`, you need to add 1 to the result. For example:

```
X64 = double(X8) + 1;
```

To convert from double to uint8, you need to first subtract 1, and then use round to ensure all the values are integers:

```
X8 = uint8(round(X64 – 1));
```

The order of the operations must be as shown in these examples, because you cannot perform mathematical operations on uint8 arrays.

When you write an indexed image using imwrite, MATLAB automatically converts the values if necessary.

Colormaps

Colormaps in MATLAB are always m-by-3 arrays of double-precision floating-point numbers in the range [0, 1]. In most graphics file formats, colormaps are stored as integers, but MATLAB does not support colormaps with integer values. imread and imwrite automatically convert colormap values when reading and writing files.

Truecolor Images

In a truecolor image of class double, the data values are floating-point numbers in the range [0, 1]. In a truecolor image of class uint8, the data values are integers in the range [0, 255].

If you want to convert a truecolor image from one data type to the other, you must rescale the data. For example, this call converts a uint8 truecolor image to double:

```
RGB64 = double(RGB8)/255;
```

This call converts a double truecolor image to uint8:

```
RGB8 = uint8(round(RGB*255));
```

The order of the operations must be as shown in these examples, because you cannot perform mathematical operations on uint8 arrays.

When you write a truecolor image using imwrite, MATLAB automatically converts the values if necessary.

New and Enhanced Handle Graphics Object Properties

This section lists new graphics object properties supported in MATLAB 5. It also lists graphics properties whose behavior has changed significantly. *Using MATLAB Graphics* provides a more detailed description of each property.

Table 1-26: Properties of All Graphics Objects

Property	Description
`BusyAction`	Controls events that potentially interrupt executing callback routines.
`Children`	Enhanced behavior allows reordering of child objects
`CreateFcn`	A callback routine that executes when MATLAB creates a new instance of the specific type of graphics object
`DeleteFcn`	A callback routine that executes when MATLAB deletes the graphics object
`HandleVisibility`	Controls scope of handle visibility
`Interruptible`	Now on by default
`Parent`	Enhanced behavior allows reparenting of graphics objects
`Selected`	Indicates whether graphics object is in selected state
`SelectionHighlight`	Determines if graphics objects provide visual indication of selected state
`Tag`	User-specified object label

Table 1-27: Axes Properties

Property	Description
`AmbientLightColor`	Color of the surrounding light illuminating all Axes child objects when a Light object is present.
`CameraPosition`	Location of the point from which the Axes is viewed.
`CameraPositionMode`	Automatic or manual camera positioning.
`CameraTarget`	Point in Axes viewed from camera position.
`CameraTargetMode`	Automatic or manual camera target selection.
`CameraUpVector`	Determines camera rotation around the viewing axis.
`CameraUpVectorMode`	Default or user-specified camera orientation.
`CameraViewAngle`	Angle determining the camera field of view.
`CameraViewAngleMode`	Automatic or manual camera field of view selection.
`DataAspectRatio`	Relative scaling of *x*-, *y*-, and *z*-axis data units.
`DataAspectRatioMode`	Automatic or manual axis data scaling.
`FontUnits`	Units used to interpret the `FontSize` property (allowing normalized text size).
`Layer`	Draw axis lines below or above child objects.
`NextPlot`	Enhanced behavior supports `add`, `replace`, and `replacechildren` options.
`PlotBoxAspectRatio`	Relative scaling of Axes plot box.
`PlotBoxAspectRatioMode`	Automatic or manual selection of plot box scaling.

Table 1-27: Axes Properties (Continued)

Property	Description
`Projection`	Select orthographic or perspective projection type.
`TickDirMode`	Automatic or manual selection of tick mark direction (allowing you to change view and preserve the specified `TickDir`).
`XAxisLocation`	Locate *x*-axis at bottom or top of plot.
`YAxisLocation`	Locate *y*-axis at left or right side of plot.

Table 1-28: Figure Properties

Property	Description
`CloseRequestFcn`	Callback routine executed when you issue a `close` command on a Figure.
`Dithermap`	Colormap used for true-color data on pseudo-color displays.
`DithermapMode`	Automatic dithermap generation.
`IntegerHandle`	Integer or floating-point Figure handle.
`PaperPositionMode`	WYSIWYG printing of Figure.
`NextPlot`	Enhanced behavior supports `add`, `replace`, and `replacechildren` options.
`PointerShapeCData`	User-defined pointer data.
`PointerShapeHotSpot`	Active point in custom pointer.
`PrintPostProcess`	Commands to execute at the end of the printing process.
`Renderer`	Select painters or Z-buffer rendering or enable MATLAB to select automatically.
`Resize`	Determines if Figure window is resizeable.

Table 1-28: Figure Properties (Continued)

Property	Description
`ResizeFcn`	Callback routine executed when you resize the Figure window.

Table 1-29: Image Properties

Property	Description
`CData`	Enhanced behavior allows true color (RGB values) specification.
`CDataMapping`	Select direct or scaled interpretation of indexed colors.

Table 1-30: Light Properties

Property	Description
`Color`	Color of the light source.
`Position`	Place the light source within Axes space.
`Style`	Select infinite or local light source.

Table 1-31: Line Properties

Property	Description
`Marker`	The marker symbol to use at data points (markers are now separate from line style).
`MarkerEdgeColor`	The color of the edge of the marker symbol.
`MarkerFaceColor`	The color of the face of filled markers.

Table 1-32: Patch Properties

Property	Description
AmbientStrength	The strength of the Axes ambient light on the particular Patch object.
CData	Enhanced behavior allows true color (RGB values) specification.
CDataMapping	Select direct or scaled interpretation of indexed colors.
DiffuseStrength	Strength of the reflection of diffuse light from Light objects.
FaceLightingAlgorithm	Lighting algorithm used for Patch faces.
Faces	The vertices connected to define each face.
FaceVertexCData	Color specification when using the Faces and Vertices properties to define a Patch.
LineStyle	Type of line used for edges.
Marker	Symbol used at vertices.
MarkerEdgeColor	The color of the edge of the marker symbol.
MarkerFaceColor	The color of the face of filled markers.
MarkerSize	Size of the marker.
NormalMode	MATLAB-generated or user-specified normal vectors.
SpecularColorReflectance	Control the color of the specularly reflected light from Light objects.
SpecularExponent	Control the shininess of the Patch object.
SpecularStrength	Strength of the reflection of specular light from Light objects.

Table 1-32: Patch Properties (Continued)

Property	Description
VertexNormals	Definition of the Patch's normal vectors.
Vertices	The coordinates of the vertices defining the Patch.

Table 1-33: Root Properties

Property	Description
CallbackObject	Handle of object whose callback is currently executing.
ErrorMessage	Text of the last error message issued by MATLAB.
ErrorType	The type of the error that last occurred.
ShowHiddenHandles	Show or hide graphics object handles that are marked as hidden.
TerminalHideGraphCommand	Command to hide graphics window when switching to command mode.
TerminalDimensions	Size of graphics terminal.
TerminalShowGraphCommand	Command to expose graphics window when switching from command mode to graphics mode.

Table 1-34: Surface Properties

Property	Description
AmbientStrength	The strength of the Axes ambient light on the particular Surface object.
CData	Enhanced behavior allows true color (RGB values) specification.

Table 1-34: Surface Properties (Continued)

Property	Description
CDataMapping	Selects direct or scaled interpretation of indexed colors.
DiffuseStrength	Strength of the reflection of diffuse light from Light objects.
FaceLightingAlgorithm	Lighting algorithm used for Surface faces.
Marker	Symbol used at vertices.
MarkerEdgeColor	The color of the edge of the marker symbol.
MarkerFaceColor	The color of the face of filled markers.
MarkerSize	Size of the marker.
NormalMode	MATLAB generated or user-specified normal vectors.
SpecularColorReflectance	Control the color of the specularly reflected light from Light objects.
SpecularExponent	Control the shininess of the Surface object.
SpecularStrength	Strength of the reflection of specular light from Light objects.
VertexNormals	Definition of the Surface's normal vectors.
Vertices	The coordinates of the vertices defining the Surface.

Table 1-35: Text Properties

Property	Description
FontUnits	Select the units used to interpret the FontSize property (allowing normalized text size).
Interpreter	Allows MATLAB to interpret certain characters as LaTex commands.

Table 1-36: Uicontrol Properties

Property	Description
Enable	Enable or disable (gray out) uicontrols.
FontAngle	Select character slant.
FontName	Select font family.
FontSize	Select font size.
FontUnits	Select the units used to interpret the FontSize property (allowing normalized text size).
FontWeight	Select the weight of text characters.
ListboxTop	Select the listbox item to display at the top of the listbox.
SliderStep	Select the size of the slider step.
Style	Enhanced to include listbox device.

Table 1-37: Uimenu Properties

Property	Description
Enable	Enable or disable (gray out) uicontrols.

Improvements to Graphical User Interfaces (GUIs)

General GUI Enhancements

MATLAB 5 provides general enhancements that are useful in the GUI area:

- Starting MATLAB with the –nosplash argument suppresses the splash screen on UNIX.
- Using the `CloseRequestFcn` callback can abort a Figure `close` command.
- Stacking of Figure and Axes graphics objects may be varied to affect the order in which MATLAB displays these objects.
- The mouse pointer can be set to a number of different symbols or you can create a custom Figure pointer.
- On the Windows platforms edit controls now have a three-dimensional appearance.

MATLAB 5 provides features that make it easier to create MATLAB GUIs. Major enhancements include List Box objects to display and select one or more list items. You can also create modal or non-modal error, help, and warning message boxes. In addition, uicontrol edit boxes now support multiline text.

Table 1-38: New GUI Controls

Function	Description
`msgbox`	Display message box.
`dragrect`	Drag pre-defined rectangles.
`inputdlg`	Display a dialog box to input data.
`questdlg`	Question dialog.
`rbbox`	Rubberband box.
`selectmoveresize`	Interactively select, move, or resize objects.

MATLAB 5 also provides more flexibility in callback routines. You can specify callbacks that execute after creating, changing, and deleting an object.

Table 1-39: New Program Execution Controls

Function	Description
`uiresume`	Resume suspended M-file execution.
`uiwait`	Blocks program execution.
`waitfor`	Blocks execution until a condition is satisfied.

Guide

Guide is a Graphical User Interface (GUI) design tool. In other words, it makes it easy to create and modify GUIs in MATLAB. The individual pieces of the Guide environment are designed to work together, but they can also be used individually. For example, there is a Property Editor (invoked by the command propedit) that allows you to modify any property of any Handle Graphics object, from a figure to a line. Point the Property Editor at a line and you can change its color, position, thickness, or any other line property.

The Control Panel is the centerpiece of the Guide suite of tools. It lets you "control" a figure so that it can be easily modifed by clicking and dragging. As an example, you might want to move a button from one part of a figure to another. From the Control panel you put the button's figure into an editable state, and then it's simply a matter of dragging the button into the new position. Once a figure is editable, you can also add new uicontrols, uimenus, and plotting axes.

Table 1-40: Guide Tools

Tool	Command	Description
Control Panel	`guide`	Control figure editing.
Property Editor	`propedit`	Modify object properties.
Callback Editor	`cbedit`	Modify object callbacks.
Alignment Tool	`align`	Align objects.
Menu Editor	`menuedit`	Modify figure menus.

Enhancements to the Application Program Interface (API)

The MATLAB 5 API introduces data types and functions not present in MATLAB 4. This section summarizes the important changes in the API. For details on any of these topics, see the *MATLAB Application Program Interface Guide*.

New Fundamental Data Type

The MATLAB 4 `Matrix` data type is obsolete. MATLAB 5 programs use the `mxArray` data type in place of `Matrix`. The `mxArray` data type has extra fields to handle the richer data constructs of MATLAB 5.

Functions that expected `Matrix` arguments in MATLAB 4 expect `mxArray` arguments in MATLAB 5.

New Functions

The API introduces many new functions that work with the C language to support MATLAB 5 features.

Support for Structures and Cells

MATLAB 5 introduces structure arrays and cell arrays. Therefore, the MATLAB 5 API introduces a broad range of functions to create structures and cells, as well as functions to populate and analyze them. See "How to Convert Each MATLAB 4 Function" on page 2-21 for a complete listing of these functions.

Support for Multidimensional Arrays

The MATLAB 4 `Matrix` data type assumed that all matrices were two-dimensional. The MATLAB 5 `mxArray` data type supports arrays of two or more dimensions. The MATLAB 5 API provides two different `mxCreate` functions that create either a two-dimensional or a multidimensional `mxArray`.

In addition, MATLAB 5 introduces several functions to get and set the number and length of each dimension in a multidimensional `mxArray`.

Support for Nondouble Precision Data

The MATLAB 4 `Matrix` data type represented all numerical data as double-precision floating-point numbers. The MATLAB 5 `mxArray` data type

can store numerical data in six different integer formats and two different floating-point formats.

Note Although the MATLAB API supports these different data representations, MATLAB itself does not currently provide any operations or functions that work with nondouble precision data. Nondouble precision data may be viewed, however.

Access toSpecial Numbers

Several `mex`-prefix functions that access special numbers such as Infinity, NaN, and `eps` have been renamed. The new names use the `mx` prefix instead of the `mex` prefix. For example, `mexGetEps` is obsolete; call `mxGetEps` instead. These functions are now available from the stand-alone interfaces.

OLE Support

The MATLAB API now provides OLE support on the Windows platforms.

MATLAB 4 Features Unsupported in MATLAB 5

Non-ANSI C Compilers

MATLAB 4 let you compile MATLAB applications with non-ANSI C compilers. MATLAB 5 requires an ANSI C compiler.

printf and scanf

MATLAB 5 MEX-files no longer support calls to the ANSI C `printf` and `scanf`. Instead of calling `printf`, your MEX-file should always call `mexPrintf`. Instead of calling `scanf`, your MEX-file should call `mexCallMATLAB` with the fifth argument set to the input function.

New Platform Specific Features

Two features are available on both the Macintosh and MS Windows platforms:

- Japanese characters

  It is now possible to generate annotation and string constants that use Japanese characters.

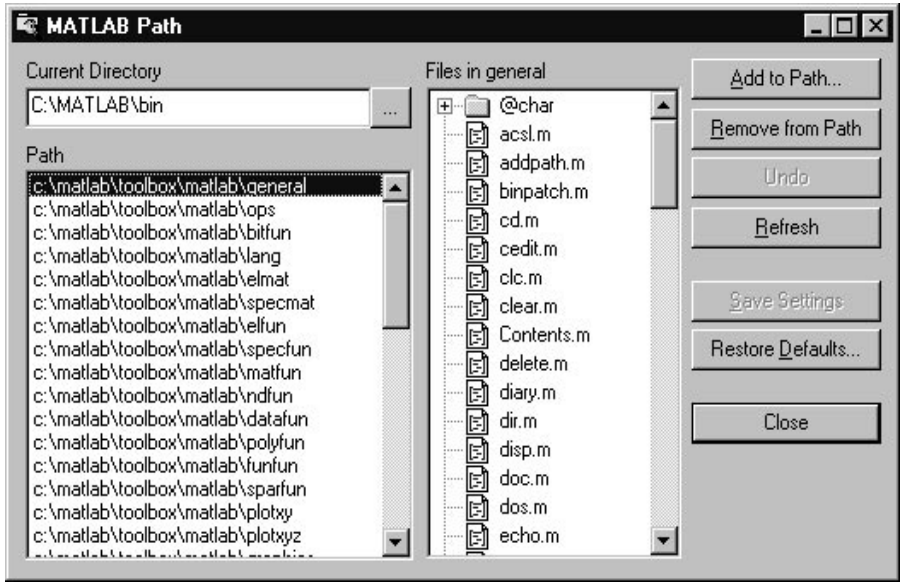
- 16-bit stereo sound

  MATLAB 5 now supports 16-bit stereo sound.

MS Windows

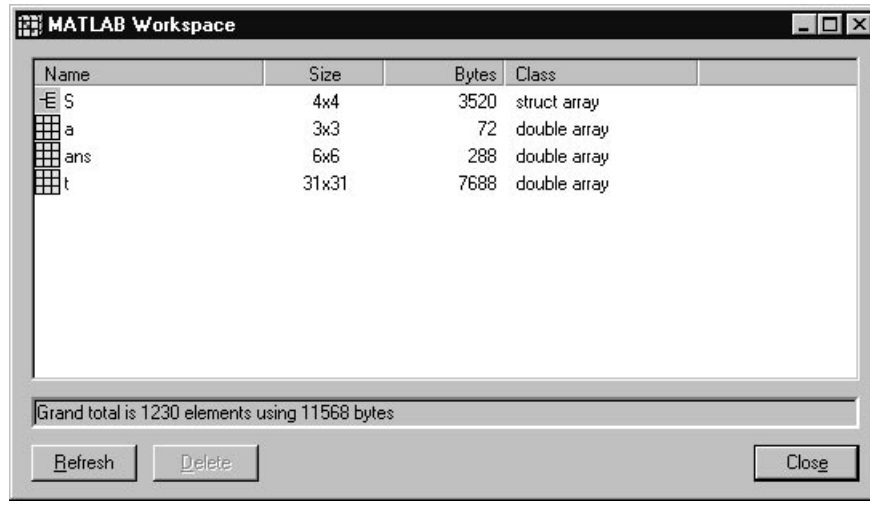
Path Browser

The Path Browser lets you view and modify the MATLAB search path. All changes take effect in MATLAB immediately.



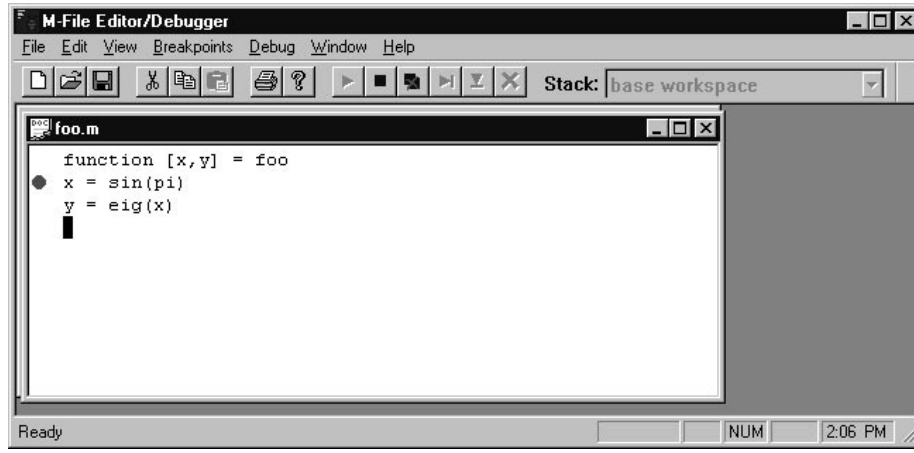
Workspace Browser

The Workspace Browser lets you view the contents of the current MATLAB workspace. It provides a graphical representation of the traditional whos output. In addition, you can clear workspace variables and rename them.



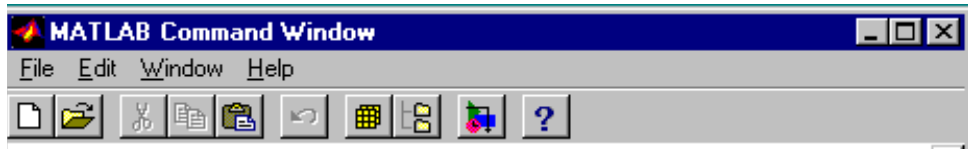
M-File Debugger

The graphical M-file debugger allows you to set breakpoints and single-step through M-code. The M-file debugger starts automatically when a breakpoint is hit.



Command Window Toolbar

A toolbar is now optionally present for the Command Window. The toolbar provides single-click access to several commonly used operations:

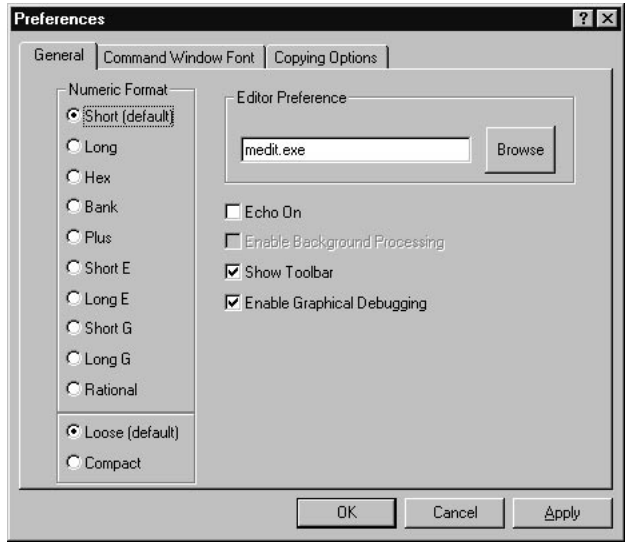


- Open a new editor window
- Open a file for editing
- Cut, copy, paste, and undo
- Open the Workspace Browser
- Open the Path Browser
- Create new SIMULINK model
- Access the Help facility

New Dialog Boxes

New **Preferences** dialog boxes are accessible through the **File** menu. Some of these were previously available through the **Options** menu in MATLAB 4. There are three categories of preferences:

- General
- Command Window Font
- Copying Options



Macintosh

User Interface Enhancements

- Optional toolbars in the Command Window, Editor windows, and M-file debugger allow rapid access to commonly used features.

- Color syntax highlighting in the Command Window, Editor windows, and M-file debugger provides visual cues for identifying blocks of code, comments, and strings.
- Almost all lists and text items in the Command Window, Editor, Path Browser, Workspace Browser, M-file debugger, and Command History Window have optional dynamic or “live” scrolling; the display is scrolled as the scroll box of a scrollbar is moved.
- Macintosh Drag and Drop is supported throughout MATLAB for rapid and easy exchange of text between windows.

Command Window Features

- Typing on the current command line can now be undone and redone. This includes cutting, clearing, overtyping, dragging, and dropping.
- Placing the caret on an error message and pressing **Enter** opens the M-file in the Editor, positioned to the offending line.

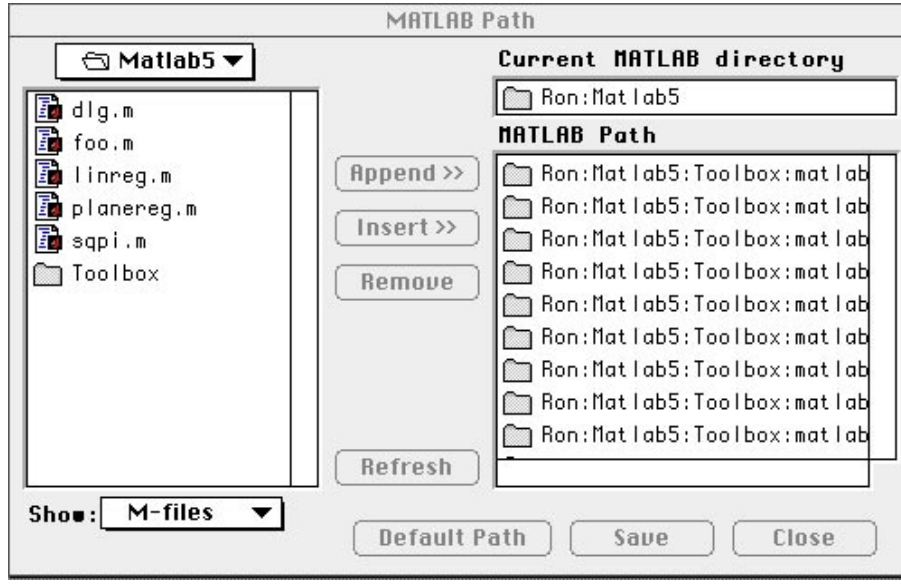
Command History Window

The Command History window contains a list of all commands executed from the Command Window. Commands are saved between MATLAB sessions, so

you can select and execute a group of commands from a previous day's work to continue quickly from where you left off.

Path Browser

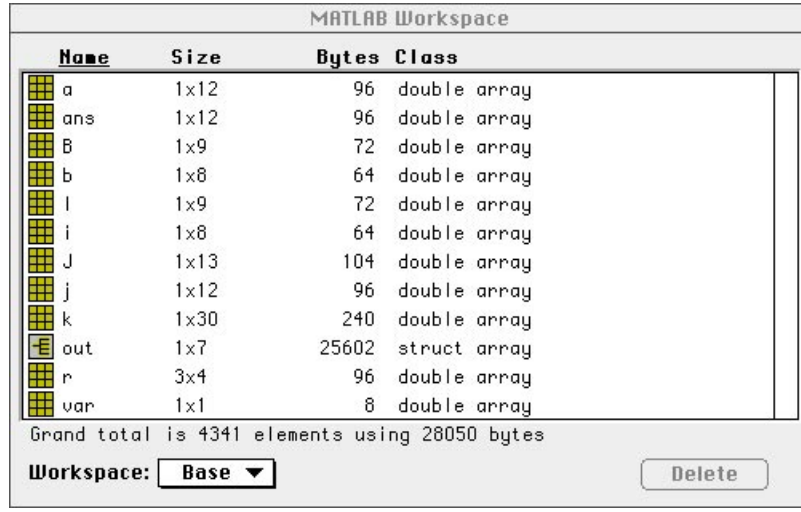
The Path Browser provides an intuitive, easy-to-use graphical interface for viewing and modifying the MATLAB search path. The search path may be reordered or modified simply by dragging items in the path list.



Workspace Browser

The Workspace Browser allows you to view the contents of the current MATLAB workspace. It provides a graphic representation of the traditional `whos` output. You can delete variables from the workspace and sort the work-

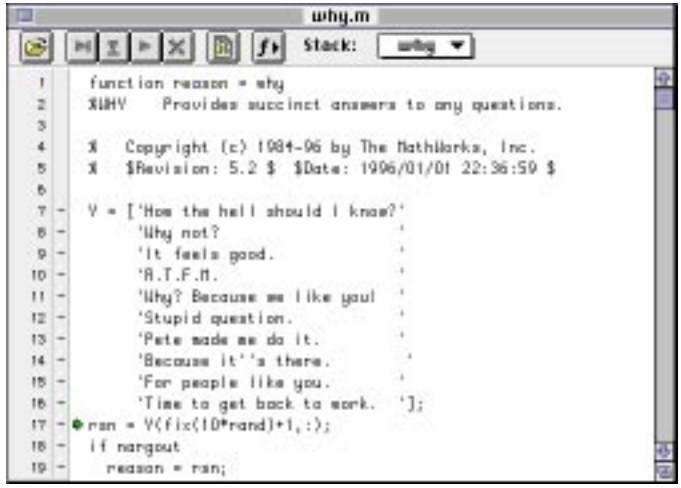
space by various criteria. Double-clicking workspace variables displays that variable's contents in the Command Window.



M-File Debugger

MATLAB 5 includes a graphical M-file debugger, which allows you to set breakpoints and single-step through M-code. Selecting text in the debugger

window and pressing the **Enter** (not the **Return**) key evaluates that text in the Command Window.



Editor Features

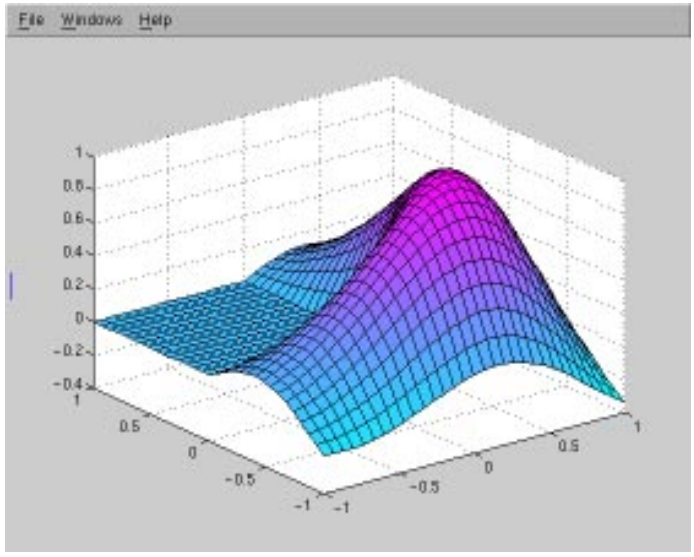
- Command-clicking in the title of an Editor window displays a pop-up menu containing the full path to the M-file. Selecting a folder from the pop-up menu opens that folder in the Finder.
- Selecting text in an Editor window and pressing **Enter** evaluates that text in the Command Window.
- Typing a close parenthesis, bracket, or brace briefly highlights the matching open parenthesis, bracket, or brace.
- Double-clicking a parenthesis, bracket, or brace selects all text within the matching parenthesis, bracket, or brace.

• Line numbers may be optionally displayed.

UNIX Workstations

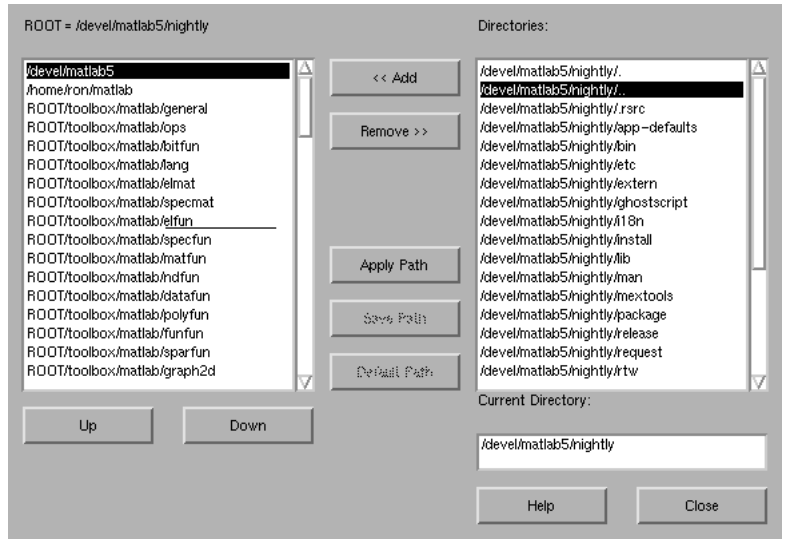
Figure Window Toolbar

The Figure window now provides a toolbar with a **File** pulldown menu. Selecting the **Print** option on the **File** menu activates a set of pushbuttons that allow easy setting of the most frequently used print options.



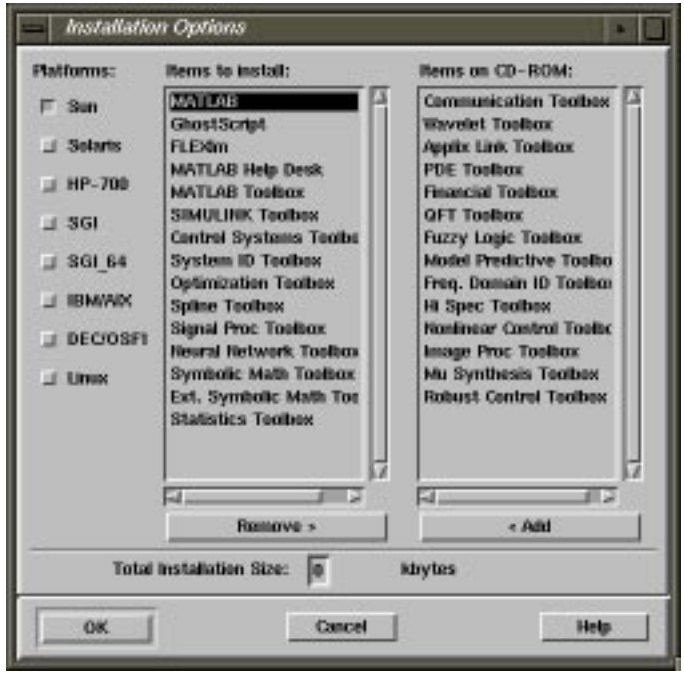
Path Editor

The `pathedit` command displays a GUI that allows you to view and modify your MATLAB search path.



Simplified Installation Procedure

The installation procedure now uses a GUI to select or deselect products and platforms.



2

Upgrading to MATLAB 5

Upgrading from MATLAB 4 to MATLAB 5

MATLAB 5 is a major upgrade to MATLAB. Although The MathWorks endeavors to maintain full upwards compatibility between subsequent releases of MATLAB, inevitably there are situations where this is not possible. In the case of MATLAB 5, there are a number of changes that you need to know about in order to migrate your code from MATLAB 4 to MATLAB 5.

It is useful to introduce two terms in discussing this migration. The first step in converting your code to MATLAB 5 is to make it MATLAB 5 *compatible*. This involves a rather short list of possible changes that let your M-files run under MATLAB 5. The second step is to make it MATLAB 5 *compliant*. This means making further changes so that your M-file is not using obsolete, but temporarily supported, features of MATLAB. It also can mean taking advantage of MATLAB 5 features like the new data constructs, graphics, and so on.

There are a relatively small number of things that are likely to be in your code that you will have to change to make your M-files MATLAB 5 compatible. Most of these are in the graphics area.

There are a somewhat larger number of things you can do (but don't have to) to make your M-files fully MATLAB 5 compliant. To help you gradually make your code compliant, MATLAB 5 displays warning messages when you use functions that are obsolete, even though they still work correctly.

Converting M-Files from MATLAB 4 to MATLAB 5

This section describes some changes you can make to your MATLAB 4 code to eliminate error messages and warnings due to incompatible and noncompliant statements.

Table 2-1: Language Changes

Function	Change	Action
`bessel`	The `bessel` functions no longer produce a table for vector arguments of the same orientation.	In `besselj(nu,x)`, specify `nu` as a row and `x` as a column to produce a table.
`case`, `otherwise`, `switch`, `default`	`case`, `otherwise`, `switch`, and `default` cannot be used as variable names.	Rename your variables.
`dialog`	`dialog.m` now creates a modal dialog.	Use the `msgbox` function instead.
`end`	extra `end` statements	Remove redundant `end` statements.
`eps`	`eps` is a function	`eps = 0` no longer redefines `eps` for other functions (it makes a local variable called `eps` in the current workspace). Functions that base their tolerance on externally defined `eps` won't work. Change code accordingly.
`global`	Undefined globals	Define globals before they are used. Always put the `global` statement at the top of the M-file (just below the help comments).
`gradient`	`gradient` no longer produces complex output.	Use two outputs in the two-dimensional case.

Table 2-1: Language Changes (Continued)

Function	Change	Action
`input`	`input('prompt','s')` no longer outputs an initial line feed. Prompts now show up on the same line.	Update code accordingly if this causes a display problem. Add `\n` in the prompt string to force a line feed.
`interp1`	The old `interp1` syntax (`interp1(x,n)`) no longer calls `interpft`. A warning was in place in MATLAB 4.	Update code accordingly.
	`interp1` now returns a row vector when given a row vector. It used to return a column vector.	Transpose the output of `interp1` to produce the MATLAB 4 result when `xi` is a row vector.
	`interp1('spline')` returns `NaN's for out of range values.`	Use `spline` directly.
`interp2`	The old `interp2` syntax (`interp2(x,y,xi)`) no longer calls `interp1`. A warning was in place in MATLAB 4.	Update code accordingly.
`interp3`	The old `interp3` syntax (`interp3(z,m,n)` or `interp3(x,y,z,xi,yi)`) no longer calls `griddata` (users were warned in 4.0). `interp3` is now three-dimensional interpolation.	Update code accordingly.
`isempty`	`A == []` and A ~= [] as a check for an empty matrix produce warning messages.	Use `isempty(A)` or `~isempty(A)`. In a future version `A == []` will produce an empty result.
`isspace`	`isspace` only returns `true` (`1`) on strings. `isspace(32)` is `0` (it was `1` in MATLAB 4).	Wrap your calls to `isspace` with `char`.

Table 2-1: Language Changes (Continued)

Function	Change	Action
`logical`	Some masking operations where the mask isn't defined using a logical expression now produce an out of range index error.	Wrap the subscript with a call to `logical` or use the logical expression `A~=0` to produce MATLAB 4 behavior.
	Boolean indexing is no longer directly supported.	Use `logical` to create the index array.
`matlabrc`	On the PC, MATLAB no longer stores the path in `matlabrc`.	All platforms except the Macintosh now use `pathdef.m`. The Macintosh still stores its path in the MATLAB `Settings` file (usually in the `Preferences` folder).
`max`	`max(size(v))`, as a means to determine the number of elements in a vector `v`, fails when `v` is empty.	Use `length(v)` in place of `max(size(v))` as the upper limit on loops over the elements of any vector.
`nargin,nargout`	`nargin` and `nargout` are functions.	`nargout = nargout-1` (and any similar construction) is an error. To work around this change, assign `nargin` to a local variable and increment that variable. Rename all occurrences of `nargin` to the new variable. The same holds true for all functions.
`ones`	`A(ones(size(A)))` no longer produces `A`.	This statement produces copies of the first element of `A`. Use `A(ones(size(A))~=0)` or just `A` to produce the MATLAB 4 behavior.
	Functions such as `ones`, `eye`, `rand`, and `zeros` give an error if supplied with a matrix argument (such as `zeros(A)`).	Use the syntax `ones(size(A))` instead.

Table 2-1: Language Changes (Continued)

Function	Change	Action
`rand`	`rand('normal')` and `rand('uniform')` no longer supported.	Use `randn` for normally distributed and `rand` for uniformly distributed random numbers.
`round`	Subscripts must be integers.	To reproduce MATLAB 4 behavior, wrap noninteger subscripts with `round()`. Strings are no longer valid subscripts (since they are not integers in the strict sense).
`slice`	`slice` no longer requires the number of columns (`ncols`) argument.	Update code accordingly.
`strcmp` `strncmp`	`strcmp` and `strncmp` now return `false` (`0`) when any argument is numeric. They used to perform an `isequal`.	Call `isequal` for all nonstrings you want to compare.
	`a(:) = b` where `a` doesn't exist creates an error. This used to do the same thing as `a = b(:)` when `a` didn't exist.	Either initialize `a` or use `a = b(:)` instead.
	Must use an explicitly empty matrix to delete elements of an array, as in `a(i) = []` or `a(i,:) = []`.	This syntax works for all built-in data types (including cell arrays and structures).
	The syntax `a(i) = B`, when `B` is empty, no longer deletes elements.	An empty assignment is attempted. Using the empty cell array `{}` in place of `[]` is not valid for deleting elements of a cell array.
	An attempt to delete elements of an array outside its range is no longer (incorrectly) ignored.	An error is generated.

Table 2-1: Language Changes (Continued)

Function	Change	Action
	Undefined variables	To reproduce MATLAB 4 behavior, initialize your variable to the empty matrix ([]) or empty string ('').
	Undefined outputs	To reproduce MATLAB 4 behavior, initialize your outputs to the empty matrix ([]).

Table 2-2: Obsolete Language Functions

Obsolete Function	Action
`csvread`, `csvwrite`	Use `dlmread(`*filename*`,',')` and `dlmwrite(`*filename*`,',')`.
`ellipk`	Replace with `ellipke`.
`extent`	Replaced by Extent property.
`figflag`	Use `findobj`.
`finite`	Rename to `isfinite`. `finite` will continue to work for MATLAB 5 but will probably be removed in a later release.
`fwhich`	Use `which`.
`hthelp`	`hthelp` works in MATLAB 5, but will not be further developed or supported.Use `helpwin` or `doc`.
`htpp`	Use `helpwin` or `doc`.
`inquire`	Use `set` and `get` to obtain the current state of an object or of MATLAB.
`inverf`	Rename to `erfinv`.
`isdir`	Use `dir`.
`layout`	No replacement in MATLAB 5.
`loadhtml`	Use `helpwin` or `doc`.

Table 2-2: Obsolete Language Functions (Continued)

Obsolete Function	Action
`matq2ws`	Replaced by `assignin` and `evalin`.
`matqdlg`	Replaced by `assignin` and `evalin`.
`matqparse`	Replaced by `assignin` and `evalin`.
`matqueue`	Replaced by `assignin` and `evalin`.
`menulabel`	Bug in Handle Graphics is now fixed.
`mexdebug`	Rename to `dbmex`.
`ode23p`	Use `ode23` with no lefthand arguments or set an output function with `odeset`.
`polyline`, `polymark`	Use the `line` object or `plot`.
`printmenu`	No replacement in MATLAB 5.
`saxis`	Use `soundsc`.
`ws2matq`	Replaced by `assignin` and `evalin`.

Table 2-3: Graphics Function Changes

Function	Change	Action
`get`	`get(0,'currentfigure')` and `get(0,'currentaxes')` no longer create an Axes if one doesn't exist. They return `[]` in that case.	`gcf` and `gca` always return a valid handle. Use `gcf` and `gca` instead of the `get` function in this context.
	In MATLAB 4 you could determine if a graphics object had a default value set by passing its handle in a query like `get(gca,'DefaultAxesColor')`.	In MATLAB 5 make the query on the object's ancestor, e.g.: `get(gcf,'DefaultAxesColor')` or `get(0,'DefaultAxesColor')`

Table 2-3: Graphics Function Changes (Continued)

Function	Change	Action
`plot`	MATLAB 4 plots may have elements that are the wrong color.	MATLAB 5 defaults to a white background on all platforms. (MATLAB 4 defaulted to white on the Macintosh and black everywhere else.) Use `colordef` to control your color defaults. Typically, you'll put a call to `colordef` in `startup.m`. To get the MATLAB 4 defaults, use `colordef none`.
	`plot` line styles `c1` through `c15` and `i` are no longer supported	Use a 1-by-3 RGB ColorSpec instead. `i` is the same as `get(gca,'color')` or `get(gcf,'color')` when the Axes color is `'none'`.
`uicontrol`	The default uicontrol text horizontal alignment is centered in MATLAB 5. (In MATLAB 4 we used to left align text and ignore the alignment property.)	Explicitly set the horizontal alignment when you create Uicontrol Text objects.

	In MATLAB 4, Uicontrols of style `'edit'` executed their callback routine whenever you moved the pointer out of the edit box. In MATLAB 5, edit controls execute their callbacks after you perform a specific action.	The callback is called when:
		• <return> key is pressed (single-line edits only)
		• focus is moved out of the edit by:
		  • clicking elsewhere in the Figure (on another Uicontrol or on another graphical object)
		  • clicking in another Figure
		  • clicking on the menubar (X Windows only)

Table 2-4: Graphics Property Changes

Property	Change	Action
`AspectRatio`	Obsolete	Replace with `DataAspectRatio` and `PlotBoxAspectRatio`.
`BackgroundColor`	Obsolete	Do not use.
`CDataScaling`	Renamed	`CDataMapping`
`CurrentMenu`	Becoming obsolete. No warning message produced.	Replace with the function `gcbo`.
`EraseMode`	We now `xor` against the Axes color rather than the Figure color.	Modify code as appropriate.
`ExpFontAngle`	Obsolete	Do not use.
`ExpFontName`	Obsolete	Do not use.
`ExpFontSize`	Obsolete	Do not use.
`ExpFontStrikeThrough`	Obsolete	Do not use.
`ExpFontUnderline`	Obsolete	Do not use.
`ExpFontUnits`	Obsolete	Do not use.
`ExpFontWeight`	Obsolete	Do not use.
`FontStrikeThrough`	Obsolete	Do not use.
`FontUnderline`	Obsolete	Do not use.
`FVCData`	Renamed	`FaceVertexCData`
`HiddenHandle`	Obsolete	Replace with `HandleVisibility`.

Table 2-4: Graphics Property Changes (Continued)

Property	Change	Action
`LineStyle`	Setting the `LineStyle` property to a marker value (such as `'+'`) now produces a warning. Setting the marker style of a line now affects the `MarkerStyle` property instead of the `LineStyle` property. Although you will be able to set a line marker using the `LineStyle` property (with a warning), you will not be able to get marker style information from `LineStyle`.	Set the `MarkerStyle` property instead. Note that `plot` will continue to take line-color-marker line styles. If your code relies on markers in the `LineStyle`, you'll have to change it to use the `MarkerStyle` instead.
`Mode`	Renamed	`Style`
`ProjectionType`	Becoming obsolete. No warning message produced.	Will be replaced in a future release.
`RenderLimits`	Obsolete	Do not use.
`Units`	Units/Position is always order dependent for all objects. In MATLAB 4, it was inconsistent.	The Units property should precede any properties that depend upon it. A command such as `axes('position',[100 200 300 100],'units','pixels')` is not the same as `axes('units','pixels','position',[100 200 300 100])`. In the first case, the default axes units are normalized; the numbers are interpreted in normalized coordinates.

Table 2-4: Graphics Property Changes (Continued)

Property	Change	Action
`WindowID`	Possibly becoming obsolete.	May be removed in a future release.
`XLoc2D`	Becoming obsolete. No warning message produced.	Will be replaced in a future release.
`XMinorTicks`	Renamed	`XMinorTick`
`XTickLabels`	Renamed	`XTickLabel`
`YLoc2D`	Becoming obsolete. No warning message produced.	Will be replaced in a future release.
`YMinorTicks`	Renamed	`YMinorTick`
`YTickLabels`	Renamed	`YTickLabel`
`ZMinorTicks`	Renamed	`ZMinorTick`
`ZTickLabels`	Renamed	`ZTickLabel`

Converting MEX-Files from MATLAB 4 to MATLAB 5

MATLAB 5 may or may not run existing MATLAB 4 MEX-files and binaries. If your binaries or source files are not compatible with the MATLAB 5 API, you must convert your MATLAB 4 MEX-file source code to MATLAB 5.

MEX-File Binary Incompatibility

General Considerations

MATLAB 4 binaries will not run in MATLAB 5 if they:

- directly manipulate strings.
- were built with the `-V3.5` compile switch.

PC-Specific Considerations

- 16-bit DLLs are no longer supported.

Macintosh-Specific Considerations

- MEX-files compiled for MATLAB 4 for the Macintosh Power PC are not supported. You must regenerate these MEX-files from the source code before using them with MATLAB 5.

MEX-File Source Incompatibility

General Considerations

- Non-ANSI MEX-files are no longer supported.
- MATLAB 4 Fortran MEX-files on Sun 4, MS-Windows, and Macintosh 68K platforms that access string arrays will not work.
- MATLAB 4 C and Fortran MEX-files that directly manipulate strings will not work. V4 strings were stored as double precision floating point numbers. MATLAB 5 strings are stored as 16-bit unsigned integers.
- MEX-file source code that required the `-V3.5` compile switch will not compile.
- `mexdebug` is now called `dbmex`. Only the name has changed; in all other respects `dbmex` behaves exactly like `mexdebug`.

UNIX-Specific Considerations

- The `mexrc.sh` file is no longer supported. The new options file, `mexopts.sh`, contains the same information, but in a different format. `$MATLAB/bin/mexopts.sh` is the default UNIX options file.
- The `cmex` and `fmex` Bourne shell scripts have been superseded by `mex`, a new Bourne shell script that includes both C and Fortran support, as well as additional support for C++.

PC-Specific Considerations

- Existing MATLAB 4 REX MEX-files are usable but cannot be created under MATLAB 5.
- The `cmex` and `fmex` batch files have been superseded by `mex`, a PERL script.

MEX-File Conversion Techniques

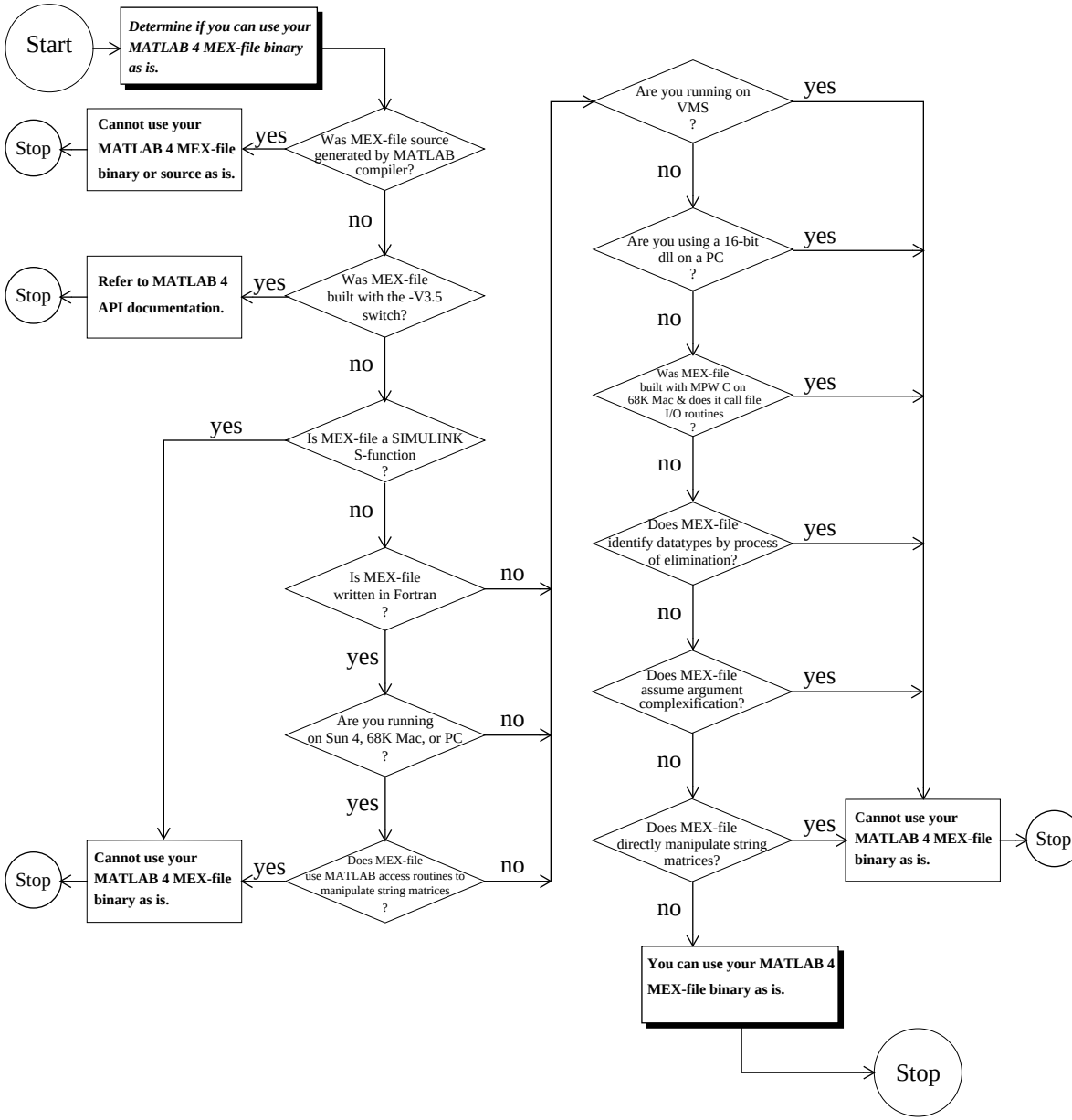
If your existing MEX-file binaries do not run, you must convert MATLAB 4 MEX-file sources to MATLAB 5 by:

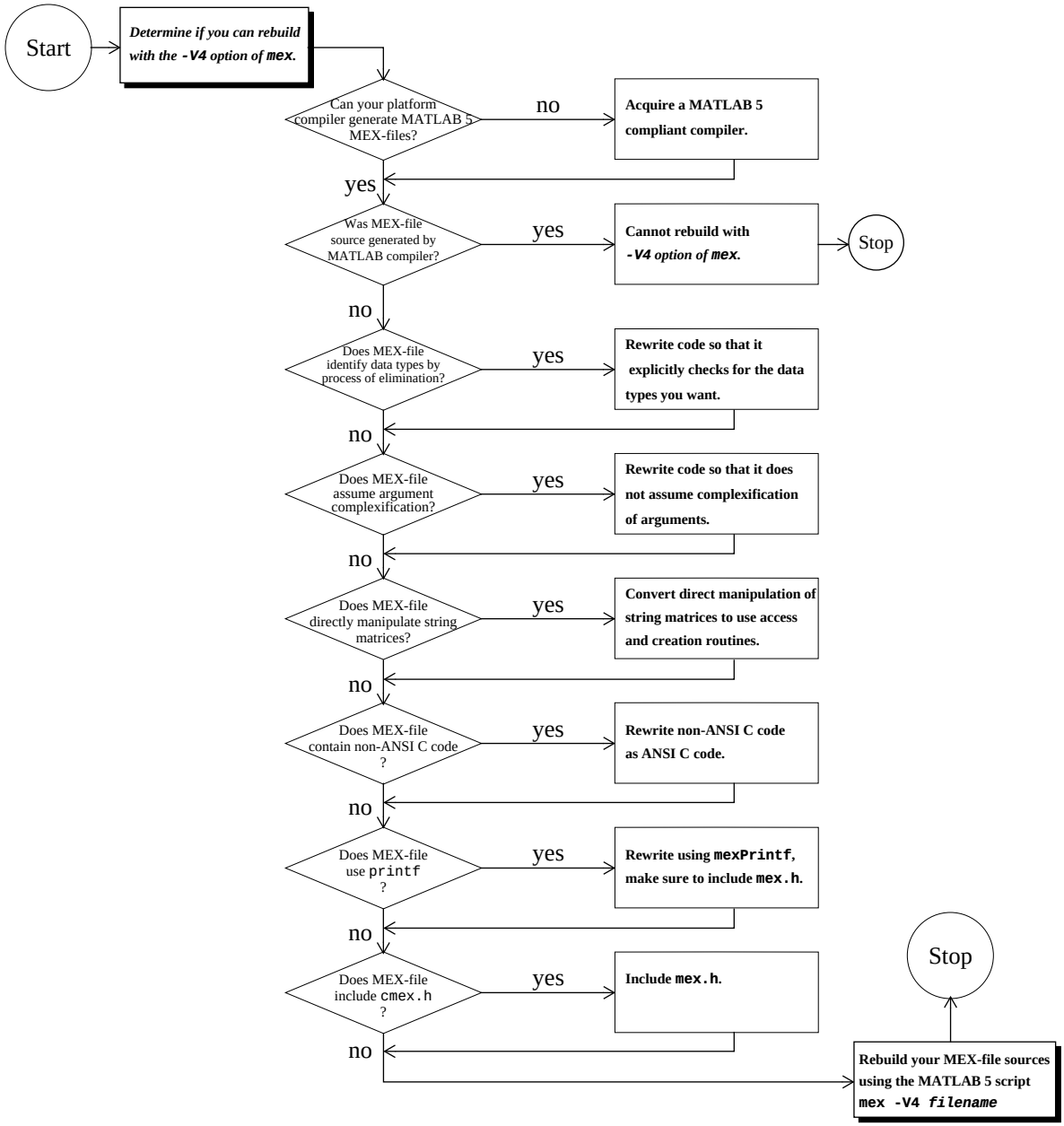
- Rebuilding MATLAB 4 source code by invoking `mex` with the `–V4` option, or
- Recoding MATLAB 4 source code to make it MATLAB 5 compliant.

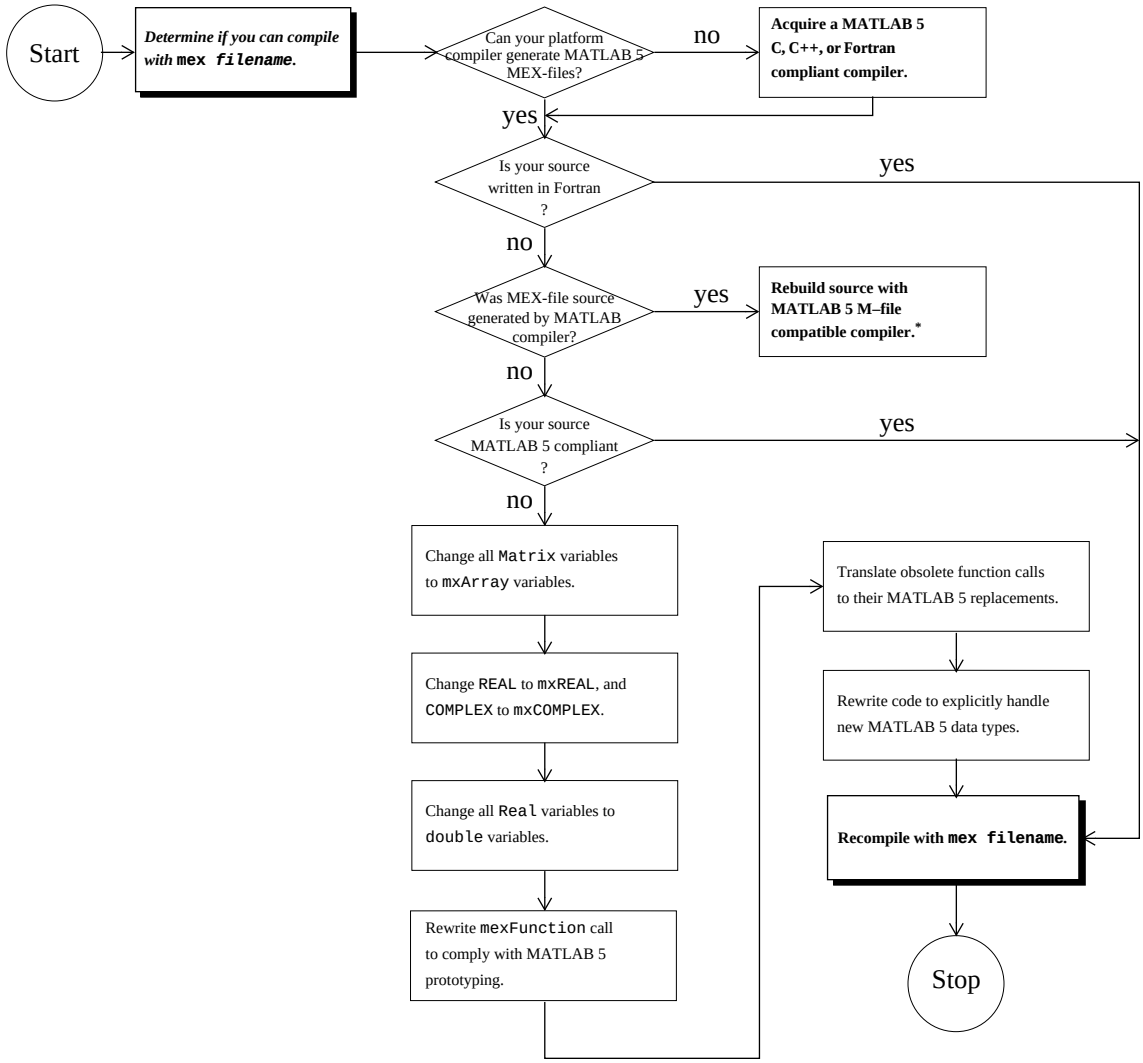
These flowcharts help you determine what steps you should take to run your MATLAB 4 MEX-files under MATLAB 5. In particular, they help you determine if you can:

- Use your MATLAB 4 MEX-file binary as is
- Rebuild your MEX-file source using the `-v4` option of `mex`
- Recompile your MEX-file source using `mex` *filename*

The sections “Rebuilding with the -V4 Option” and “Recoding for MATLAB 5 Compliance” following the flowcharts provide additional porting information.


Start
Determine if you can use your MATLAB 4 MEX-file binary as is.
Was MEX-file source generated by MATLAB compiler?
yes
Cannot use your MATLAB 4 MEX-file binary or source as is.
Stop
no
Was MEX-file built with the -V3.5 switch?
yes
Refer to MATLAB 4 API documentation.
Stop
no
Is MEX-file a SIMULINK S-function ?
yes
no
Is MEX-file written in Fortran ?
no
yes
Are you running on Sun 4, 68K Mac, or PC ?
no
yes
Does MEX-file use MATLAB access routines to manipulate string matrices ?
yes
Cannot use your MATLAB 4 MEX-file binary as is.
Stop
no
Are you running on VMS ?
yes
no
Are you using a 16-bit dll on a PC ?
yes
no
Was MEX-file built with MPW C on 68K Mac & does it call file I/O routines ?
yes
no
Does MEX-file identify datatypes by process of elimination?
yes
no
Does MEX-file assume argument complexification?
yes
no
Does MEX-file directly manipulate string matrices?
yes
Cannot use your MATLAB 4 MEX-file binary as is.
Stop
no
You can use your MATLAB 4 MEX-file binary as is.
Stop


Start
Determine if you can rebuild with the -V4 option of mex.
Can your platform compiler generate MATLAB 5 MEX-files?
no
Acquire a MATLAB 5 compliant compiler.
yes
Was MEX-file source generated by MATLAB compiler?
yes
Cannot rebuild with -V4 option of mex.
Stop
no
Does MEX-file identify data types by process of elimination?
yes
Rewrite code so that it explicitly checks for the data types you want.
no
Does MEX-file assume argument complexification?
yes
Rewrite code so that it does not assume complexification of arguments.
no
Does MEX-file directly manipulate string matrices?
yes
Convert direct manipulation of string matrices to use access and creation routines.
no
Does MEX-file contain non-ANSI C code ?
yes
Rewrite non-ANSI C code as ANSI C code.
no
Does MEX-file use printf ?
yes
Rewrite using mexPrintf, make sure to include mex.h.
no
Does MEX-file include cmex.h ?
yes
Include mex.h.
no
Stop
Rebuild your MEX-file sources using the MATLAB 5 script
mex -V4 filename



*Not available at time of printing;
Contact The MathWorks, Inc. for availability.*

Rebuilding with the -V4 Option

The simplest strategy for converting C MEX-file programs is to rebuild them with the special `-V4` option of `mex`. This option uses `mex` to include MATLAB 4 header files. Therefore, any C MEX-file source code that compiled cleanly under MATLAB 4 should compile cleanly with the `-V4` option. The resulting MEX-file should run under MATLAB 5 just as it ran under MATLAB 4. For example, given C MEX-file MATLAB 4 source code in file `MyEig.c`, recompiling under UNIX with

```
mex -V4 myeig.c
```

yields a MEX-file that MATLAB 5 can execute. It is also possible to use `cmex` and `fmex` for compiling C and Fortran source code, but both of these functions simply call `mex`.

Even with the `-V4` option, you need to recode if your source code manipulates string matrices. The `-V4` option cannot handle the different ways in which MATLAB 4 and MATLAB 5 represent string data. The MATLAB 4 `Matrix` structure held each “character” in a string matrix as a double-precision, floating-point number. MATLAB 5 represents each “character” in a string matrix as an `mxChar`, a 16-bit unsigned integer data type.

The obvious advantage to the `-V4` strategy is that it requires very little work on your part. However, this strategy provides only a temporary solution to the conversion problem; there is no guarantee that future releases of MATLAB will continue to support the `-V4` option. If you have the time, recoding for MATLAB 5 compliance is a better strategy.

Recoding for MATLAB 5 Compliance

Recoding your MATLAB 4 C or Fortran code for MATLAB 5 compliance involves:

- Rewriting any non-ANSI C code as ANSI C code. (For details, see an ANSI C book.)
- Changing all `Matrix` variables to `mxArray` variables.

  The MATLAB 4 `Matrix` data type is obsolete; you must change all `Matrix` variables to `mxArray` variables. For example, the `mxCreateSparse` function returns a Matrix pointer in MATLAB 4:

  ```
  Matrix *MySparse;
  MySparse = mxCreateSparse(10, 10, 110, REAL);
  ```

  To be MATLAB 5 compliant, change the code to:

  ```
  mxArray *MySparse;
  MySparse = mxCreateSparse(10, 10, 110, mxREAL);
  ```

- Rewriting all function prototypes.

  The function prototype of almost every MATLAB 4 `mx` and `mex` function is different in MATLAB 5. The two primary prototype changes are

  - All `Matrix` arguments are now `mxArray` arguments.
  - Pointers to read only data are now declared as `const *`.

- Changing `REAL` to `mxREAL` and `COMPLEX` to `mxCOMPLEX`.

  In any function that requires the specification of real or complex data types, instead of `REAL` and `COMPLEX`, use `mxREAL` and `mxCOMPLEX`. For example, in MATLAB 4 you would write

  ```
  mxCreateSparse(m,n,nzmax,REAL);
  ```

  to create an m-by-n sparse matrix with `nzmax` nonzero real elements. In MATLAB 5, the correct syntax for this same function is:

  ```
  mxCreateSparse(m,n,nzmax,mxREAL);
  ```

- Translating obsolete function calls into their MATLAB 5 replacements.

  A number of functions have become obsolete. However, MATLAB 5 offers replacements for nearly all of the obsolete functions. See “How to Convert Each MATLAB 4 Function” for details.

- Protecting against MATLAB 5 new data types.

If your code identifies a data type by a process of elimination, you must rewrite it in MATLAB 5. For example, if your code checks a variable and finds that it is neither a double nor a sparse matrix, you can no longer assume that the variable must be a string.

- Rewriting any code that assumes complex arguments.

  In MATLAB 4, if one argument to a MEX-function was complex, all arguments were considered complex. This is not true in MATLAB 5. For example, consider a MEX-function `myeig(A,B,C)` that calculates eigenvalues of three matrices. In MATLAB 4 if matrix `A` is complex, `B` and `C` are assumed to be complex matrices as well. In this instance additional memory is allocated for the complex part of `B` and `C`, whether or not these matrices are complex. In MATLAB 5, `B` and `C` are assumed to be real unless otherwise specified.

- Converting string matrices using API routines for access and manipulation.

  In MATLAB 4, the `Matrix` structure held each "character" in a string matrix as a double-precision, floating-point number. If the string flag was set, then MATLAB displayed each double-precision, floating-point number as an ASCII value.

  MATLAB 5 represents each "character" in a string matrix as an `mxChar`, a 16-bit unsigned integer data type. The `mxArray` does not provide a string flag; if the `mxArray`'s class is `mxCHAR_CLASS`, MATLAB treats each number in the `mxArray` as an element from the current character set. Character sets are platform specific.

  If your MATLAB 4 source code created string matrices by calling `mxCreateString`, you do not have to recode sections that create strings. However, if your MATLAB 4 source code called `mxSetString` to create string matrices, you must recode. `mxSetString` is obsolete in MATLAB 5; if you used `mxSetString` to create a two-dimensional string matrix, call `mxCreateCharMatrixFromStrings` instead.

  If your MATLAB 4 source code used something other than `mxGetString` to copy string data into a C string, you must recode. As you recode, don't forget that each character in a string `mxArray` is now stored as a 16-bit integer rather than as a double-precision, floating-point number.

How to Convert Each MATLAB 4 Function

This table shows each MATLAB 4 function along with a description of how to port that function to MATLAB 5.

Table 2-5: Converting MEX-Functions to MATLAB 5

MATLAB 4 Function	MATLAB 5 Conversion
`mexAtExit`	No change
`mexCallMATLAB`	Ssecond and fourth arguments are `mxArray *`
`mexErrMsgTxt`	No change
`mexEvalString`	No change
`mexFunction`	Second and fourth arguments are `mxArray *`. Fourth argument is a `const`.
`mexGetEps`	Obsolete; call `mxGetEps` instead

`mexGetFull`	Obsolete; call this sequence instead:
	`mexGetArray(array_ptr, "caller");`
	`name = mxGetName(array_ptr);`
	`m = mxGetM(array_ptr);`
	`n = mxGetM(array_ptr);`
	`pr = mxGetPr(array_ptr);`
	`pi = mxGetPi(array_ptr);`

`mexGetGlobal`	Obsolete; call `mexGetArrayPtr` instead, setting the second argument to `"global"`. Note: it is better programming practice to call `mexGetArray(,"global");`
`mexGetInf`	Obsolete; call `mxGetInf` instead
`mexGetMatrix`	Call `mexGetArray(name, "caller");`
`mexGetMatrixPtr`	Call `mexGetArrayPtr(name, "caller");`
`mexGetNaN`	Obsolete; call `mxGetNaN` instead.
`mexIsFinite`	Obsolete; call `mxIsFinite` instead.

Table 2-5: Converting MEX-Functions to MATLAB 5 (Continued)

`mexIsInf`	Obsolete; call `mxIsInf` instead.
`mexIsNaN`	Obsolete; call `mxIsNaN` instead.
`mexPrintf`	No change

`mexPutFull`	Obsolete; call this sequence instead:
	`mxArray *parray;`
	`int retval;`
	
	`parray=mxCreateDouble(0,0,0);`
	`if(parray==(mxArray*)0) return(1);`
	`mxSetM(parray,m);`
	`mxSetN(parray,n);`
	`mxSetPr(parray,pr);`
	`mxSetPi(parray,pi);`
	`mxSetName(parray,name);`
	
	`retval=mxPutArray(parray,"caller");`
	`mxFree(parray);`
	`return(retval);`

`mexPutMatrix`	Obsolete; call `mexPutArray` instead.
`mexSetTrapFlag`	No change
`mxCalloc`	No change
`mxCreateFull`	Obsolete; call `mxCreateDoubleMatrix` instead.
`mxCreateSparse`	Returns `mxArray *`.
`mxCreateString`	Returns `mxArray *`.
`mxFree`	No change
`mxFreeMatrix`	Obsolete; call `mxDestroyArray` instead.
`mxGetIr`	First argument is `mxArray *`.
`mxGetJc`	First argument is `mxArray *`.

Table 2-5: Converting MEX-Functions to MATLAB 5 (Continued)

`mxGetM`	First argument is `mxArray *`.
`mxGetN`	First argument is `mxArray *`.
`mxGetName`	First argument is `mxArray *`.
`mxGetNzmax`	First argument is `mxArray *`.
`mxGetPi`	First argument is `mxArray *`.
`mxGetPr`	First argument is `mxArray *`.
`mxGetScalar`	First argument is `mxArray *`.
`mxGetString`	First argument is `mxArray *`.
`mxIsComplex`	First argument is `mxArray *`.
`mxIsDouble`	First argument is `mxArray *`. Note that MATLAB 4 stores all data as `double`'s; MATLAB 5 stores data in a variety of integer and real formats.
`mxIsFull`	Obsolete; call `!mxIsSparse` instead.
`mxIsNumeric`	First argument is `mxArray *`.
`mxIsSparse`	First argument is `mxArray *`.
`mxIsString`	Obsolete; call `mxIsChar` instead.
`mxSetIr`	First argument is `mxArray *`.
`mxSetJc`	First argument is `mxArray *`.
`mxSetM`	First argument is `mxArray *`.
`mxSetN`	First argument is `mxArray *`.
`mxSetName`	First argument is `mxArray *`.
`mxSetNzmax`	First argument is `mxArray *`.
`mxSetPi`	First argument is `mxArray *`.

Table 2-5: Converting MEX-Functions to MATLAB 5 (Continued)

`mxSetPr`	First argument is `mxArray *`.
`mxSetString`	Obsolete; MATLAB 5 provides no equivalent call since the `mxArray` data type does not contain a string flag. Use `mxCreateCharMatrixFromStrings` to create multidimensional string `mxArray`'s.

Index

C

D

E

F

G

H

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z