MATLAB®

The Language of Technical Computing

If You Are Upgrading to Release 11 (MATLAB 5.3) from ...	Read These Sections ...
MATLAB 5.2 (Release 10)	Chapter 1 and "Upgrading From MATLAB 5.2 to MATLAB 5.3" in Chapter 4
MATLAB 5.1	Chapters 1 and 2, as well as "Upgrading from MATLAB 5.1 to MATLAB 5.3" and "Upgrading From MATLAB 5.2 to MATLAB 5.3" in Chapter 4
MATLAB 5.0	All
MATLAB 4	The separate, online document called "Upgrading from MATLAB 4 to MATLAB 5.3"

Computation

Visualization

Programming

The MATH WORKS Inc.

Release 11 New Features

How to Contact The MathWorks:

	508-647-7000	Phone
	508-647-7001	Fax
	The MathWorks, Inc. 24 Prime Park Way Natick, MA 01760-1500	Mail
	`http://www.mathworks.com`	Web
	`ftp.mathworks.com`	Anonymous FTP server
	`comp.soft-sys.matlab`	Newsgroup
@	`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

Release 11 New Features

© COPYRIGHT 1984 - 1999 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

U.S. GOVERNMENT: If Licensee is acquiring the Programs on behalf of any unit or agency of the U.S. Government, the following shall apply: (a) For units of the Department of Defense: the Government shall have only the rights specified in the license under which the commercial computer software or commercial software documentation was obtained, as set forth in subparagraph (a) of the Rights in Commercial Computer Software or Commercial Software Documentation Clause at DFARS 227.7202-3, therefore the rights set forth herein shall apply; and (b) For any other unit or agency: NOTICE: Notwithstanding any other lease or license agreement that may pertain to, or accompany the delivery of, the computer software and accompanying documentation, the rights of the Government regarding its use, reproduction, and disclosure are as set forth in Clause 52.227-19 (c)(2) of the FAR.

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and Target Language Compiler is a trademark of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History: January 1999 New for Release 11

Contents

1 Release 11 Enhancements

3 MATLAB 5.1 Enhancements

4 Upgrading to Release 11

Introduction

This document highlights the new features of Release 11 (MATLAB 5.3).

Note This document discusses the whole Release 11 product family, including products for which you might not be currently licensed. If you are interested in purchasing a license for a product, please contact your Sales Representative at The MathWorks or your distributor.

This document provides the information you need to upgrade to Release 11, whether you are currently using MATLAB 5.0, MATLAB 5.1, or MATLAB 5.2 (Release 10). If you are upgrading to Release 11 from MATLAB 4, then in addition to this document, you should read the online document entitled "Upgrading from MATLAB 4 to MATLAB 5.0."

How to Use This Document

If You Are Upgrading to Release 11 from ...	Read These Sections ...
MATLAB 5.2 (Release 10)	Chapter 1 and "Upgrading From MATLAB 5.2 to MATLAB 5.3" in Chapter 4
MATLAB 5.1	Chapters 1 and 2, as well as "Upgrading from MATLAB 5.1 to MATLAB 5.3" and "Upgrading From MATLAB 5.2 to MATLAB 5.3" in Chapter 4
MATLAB 5.0	All
MATLAB 4	The separate, online document called "Upgrading from MATLAB 4 to MATLAB 5.0"

References and Links to Other Documents

Throughout Chapters 1, 2, and 3, there are references to other documents for additional detailed information about new features highlighted in this document.

In the HTML version of this document, Chapters 1 and 2 include several direct links to reference documentation describing specific Release 11 features in more detail. By clicking on those links, you can go directly to the more detailed information. For example, clicking on a highlighted command name in a table displays the documentation for that command. Use your browser's **Back** button to return to this document.

Release 11 Product Family Documentation Set

Printed Manuals

The following manuals have been printed and distributed to existing customers of MATLAB® and its optional associated products, as part of their update package.

- *Known Software and Documentation Problems*
- *Release 11 New Features*
- *MATLAB Installation Guide for PC*
- *MATLAB Installation Guide for UNIX*
- *Writing S-Functions* (a new manual for Simulink customers)
- *Real-Time Workshop User's Guide*
- *Communications Toolbox New Features Guide* (Version 1.4)
- *Financial Toolbox User's Guide*
- *Optimization Toolbox User's Guide*
- *DSP Blockset User's Guide*
- *Fixed-Point Blockset User's Guide*
- *Using Simulink*
- *Stateflow User's Guide*

New Product Documentation

There are printed manuals for the following new Release 11 products:

- MATLAB Report Generator and Simulink Report Generator (one manual for both)
- MATLAB Web Server
- Database Toolbox
- Real-Time Windows Target

Manuals Updated Online

All the updated manuals listed above are also available online, via the Help Desk, in PDF format. In addition, the following manuals have been updated for Release 11 in PDF form.

- *Getting Started with MATLAB*
- *Using MATLAB*
- *MATLAB Function Reference* (includes language and graphics)
- *Application Program Interface Guide*
- *Application Program Interface Reference Manual*
- *SB2SL User's Guide*
- *Target Language Compiler Reference Guide*
- *Control System Toolbox User's Guide*
- *Database Toolbox User's Guide*
- *Financial Toolbox User's Guide*
- *Fuzzy Logic Toolbox User's Guide*
- *Image Processing Toolbox User's Guide*
- *Mapping Toolbox User's Guide*
- *Mapping Toolbox Reference Manual*
- *Model Predictive Control Toolbox User's Guide*
- *Mu Analysis and Synthesis Toolbox User's Guide*
- *Signal Processing Toolbox User's Guide*
- *Spline Toolbox User's Guide*
- *Statistics Toolbox User's Guide*
- *Symbolic Math Toolbox User's Guide*
- *Power System Blockset User's Guide*

The reference documentation for each product listed above is also available in HTML form. In addition, the following manuals are available in HTML form (and in PDF form, too, except for *Using MATLAB Graphics*):

- *Release 11 New Features*
- *Known Software and Documentation Problems*
- *Upgrading from MATLAB 4 to MATLAB 5.0*
- *Getting Started with MATLAB*
- *Using MATLAB Graphics*
- *Using Simulink*
- *Stateflow User's Guide*
- *Report Generator User's Guide*
- *Database Toolbox User's Guide*

There is context-sensitive help for the MATLAB Plot Editor, the Page Setup dialog box, Simulink®, Stateflow®, and the new MATLAB and Simulink Report Generator products.

1

Release 11 Enhancements

What's New in Release 11 (MATLAB 5.3)?

Release 11 includes:

- MATLAB® 5.3
- Simulink® 3.0, including SB2SL 2.0
- Real-Time Workshop® 3.0
- Real-Time Workshop Ada Coder 3.0
- Stateflow® and Stateflow® Coder 2.0
- New versions of most toolboxes
- The following new products:
  - MATLAB Report Generator
  - Simulink Report Generator
  - Real-Time Windows Target

In addition, the following products, which were introduced between the release of Release 10 and Release 11, are included as part of Release 11:

- Database Toolbox
- MATLAB Web Server

The new products are summarized at the end of this chapter.

Note You can access only the products for which you are licensed.

Enhancements to MATLAB 5.3

The language and development environment enhancements introduced with MATLAB 5.3 include:

- Simplified installation process
- Support for integer data types
- File I/O enhancements
- Sparse matrix operations enhancements
- Numerical analysis enhancements

- Development environment tools enhancements
- Online documentation enhancements
- Japanese interface

MATLAB 5.3 also includes many visualization enhancements, including:

- Figure window enhancements
- A new Plot Editor
- New plotting and 3-D visualization functions
- Support for HDF/EOS development tools
- Support for Portable Network Graphics (PNG) images

Upgrades to Simulink, Real-Time Workshop, Stateflow, Toolboxes, and Blocksets

Simulink 3.0

Simulink 3.0 includes several major enhancements, including:

- Graphical user interface (GUI) improvements, in particular
  - A new Library Browser and Model Browser for the PC
  - Zoomable views of diagrams
  - A new Simplot tool to recreate saved data in a Handle Graphics® window
- Several new and enhanced blocks
- Modeling enhancements
- Simulation enhancements
- SB2SL 2.0

Real-Time Workshop 3.0 and Real-Time Workshop Ada Coder 3.0

Real-Time Workshop 3.0 includes several important enhancements, including:

- Generation of production-quality embedded code
- External mode enhancements
- Data typing support

Real-Time Workshop Ada Coder 3.0 is a separate product that supports the generation of Real-Time Workshop Ada code.

Stateflow 2.0

Stateflow 2.0 and Stateflow Coder 2.0 include enhancements to the GUI, modeling features, printing, and code generation.

Toolboxes and Blocksets

These toolboxes and blocksets have significant enhancements for Release 11:

- Communications Toolbox 1.4
- Control System Toolbox 4.2
- DSP Blockset 3.0
- Financial Toolbox 2.0
- Fixed-Point Blockset 2.0
- Image Processing Toolbox 2.2
- Mapping Toolbox 1.1
- Excel Link 1.0.8
- Optimization Toolbox 2.0
- Power System Blockset 1.1
- Signal Processing Toolbox 4.2
- Statistics Toolbox 2.2
- Symbolic Math Toolbox 2.1

The following toolboxes and blocksets were updated for Release 11, but only in minor ways, for Release 11 compatibility or to take advantage of Release 11 features, and to fix software problems:

- Model Predictive Control Toolbox 1.0.4
- Mu-Analysis and Synthesis Toolbox 3.0.4
- NAG Foundation Toolbox 1.0.3
- Neural Network Toolbox 3.0.1
- Nonlinear Control Design Blockset 1.1.3
- Robust Control Toolbox 2.0.6
- Spline Toolbox 2.0.1
- System Identification Toolbox 4.0.5
- Wavelet Toolbox 1.2

PC Installation Enhancements

You Can Cut and Paste PLPs

Part of the overall simplification of the Release 11 installation process for the PC is that you can now cut and paste Personal License Passwords (PLPs) from the email you have received from The MathWorks or from Access.

New Desktop Shortcut to Start MATLAB

On PCs, you can start MATLAB from the MATLAB 5.3 shortcut on your desktop (or you can continue to use the **Start** menu). This starts MATLAB in the `matlab/work` directory.

Note If you have any existing files in the `matlab/bin` directory that you count on accessing when you start up MATLAB, then you need to put those files in the `matlab/work` directory.

Installing Notebook

The MATLAB installation script no longer installs the MATLAB Notebook product.

To install the Notebook, at the MATLAB command line type

```
notebook -setup
```

MATLAB Language Enhancements

Links to Function Descriptions If you are reading this in HTML form, clicking on the function name in the tables summarizing functions (or in highlighted links in the text) displays the reference documentation for that function. Use your browser's **Back** button to return to this document.

Support for Integer Data Types

New Integer Array Classes

Version 5.3 extends the support for integer data types to include several additional array classes that store integer data types. MATLAB 5.3 adds to the existing `uint8` class 8-, 16-, and 32-bit signed and unsigned integer array classe; for example, `int16` for signed 16-bit integers.

These classes are primarily meant to store integer values. Most operations that manipulate arrays without changing their elements are defined for these data types (examples are reshape, size, the logical and relational operators, subscripted assignment, and subscripted reference). In addition, MATLAB supports the `find` function for integer arrays, but the returned array is of class `double`. No math operations except for sum are defined for these classes, since such operations are ambiguous on the boundary of the set (for example, they could wrap or truncate there).

sum Function Now Supports All Integer Types

The `sum` function can now be used with all of the integer data types supported by MATLAB. Previously, `sum` only worked with `uint8` data type. With this release, `sum` also supports the `int8`, `int16`, and `int32` data types and the `uint16` and `uint32` unsigned data types. When the `sum` function is used with integer data types, the value returned by `sum` is of class `double`.

File I/O Enhancements

User-Extensible File Opening Function

The new `open` function is a user-extensible function that provides an interface to file open operations. Default behavior is provided for these standard MATLAB file types: Handle Graphics figure files, M-files, model files, and P-files. You can extend the interface to include other file types and to override the default behavior for the standard files.

Reading Data From a Uniformly Formatted File

The new `textread` function provides easy reading of data from a uniformly formatted file, such as a comma- or tab-delimitered file, into MATLAB variables. The formatted file can contain both numbers and strings. The data is converted using the types and delimiters you specify.

Enhancements to dlmread

The `dlmread` function was enhanced to significantly improve function performance.

Also, to use the `range` argument, use this new calling sequence:

```
M = dlmread (filename,delimiter,range)
```

Saving MATLAB Figures or Models

The new `saveas` function saves a MATLAB figure or model to a file using the specified file format.

Support for Single Precision Data

MATLAB now supports single precision data, solely as a storage format. Using the `fread` and `fwrite` functions, you can read input files containing single precision data and write data to files in single precision format. To convert data to single precision format, use the `single` function.

Note MATLAB does not support operations on single precision data other than conversion. In particular, mathematical operations are not supported. Currently, single precision data is primarily used in MATLAB with the Hierarchical Data Format (HDF) development tools.

String Conversion

The new `str2double` function converts a character string to a double precision value. The character string should contain the ASCII representation of a scalar value (real or complex). Use of `str2double` is recommended over the `str2num` function.

The new `texlabel` function produces the TeX format from a character string.

Constructing Complex Data

The new `complex` function constructs complex data from real and imaginary parts.

pause Accepts Fractions of Seconds

The `pause` function now accepts a fractional number of seconds as an input argument. This means that when you use the calling sequence `pause(n)`, `n` can be any real number. Previously, `n` was restricted to integer values.

Enhancements to quit

The `quit` function has been enhanced to run the script `finish.m`, if `finish.m` exists anywhere on the MATLAB path. `finish.m` is a file you create that contains commands you want to run when MATLAB terminates (i.e., you use `quit`, `exit`, or click on the **X** button to close the window on the PC).

Two sample files illustrating what you could put in `finish.m` are provided in `/toolbox/local`:

- `finishsav.m` – saves the workspace to a MAT-file when MATLAB quits
- `finishdlg.m` – displays a dialog box allowing you to cancel quitting

You can also cancel quitting from within the `finish.m` file by using `quit cancel`.

Y2K Support

All versions of The MathWorks, Inc.'s software products have always represented data in 8-byte double-precision floating-point data type form. This ensures that our products will be able to function properly in your environment in the year 2000 and beyond, including leap years, without any adjustments or action required on your part.

Date Functions Calling Sequence Change

With MATLAB 5.3, the date functions `datenum`, `datestr`, and `datevec` include a new calling sequence that allows a pivot year specification to override the default. For example, here's the new calling sequence for `datevec`:

```
[...] = datevec(t, pivotyear)
```

This new call uses the pivot year instead of the current year minus 50 years.

See "Upgrading From MATLAB 5.2 to MATLAB 5.3" in Chapter 4 for details about possible changes you might want to make to existing applications.

Operating on Cell Arrays

The new `cellfun` function is a multipurpose function that performs common operations on the elements of cell arrays, including: `isreal`, `isempty`, `islogical`, `length`, `ndims`, `prodofsize`, `size`, and `isclass`.

Diagonal Concatenation

The new `blkdiag` function concatenates input arguments diagonally in a matrix.

Enhancements to Sparse Matrix Operations

The iterative methods that operate on sparse matrices have been enhanced to accept a function as an argument. For example,

```
x = pcg(A,b)
```

solves the system of linear equations `A*x = b` for `x`. When `A` is not explicitly available as a matrix, you can express `A` as an operator that accepts vector input `x` and returns the matrix-vector product `A*x`. This operator can be the name of an M-file, a string expression, or an inline object.

Here is a simple example.

```
function y = afun(x)
% AFUN(X) returns A*X, where A = diag(7*ones(n,1)) is diagonal.
y = 7*x; % y = A*x
```

This example calls `pcg` using the function `afun` in place of the matrix `A = diag(7*ones(n,1))`.

```
x = pcg('afun',b);
```

Numerical Analysis

Enhancements to Differential Equation Solvers

Mass Matrix Support. In previous versions of the ODE suite, only the stiff solvers could handle problems of the form $M*y'=F(t,y)$ with a mass matrix M. Since it is often convenient to solve a problem in this mass matrix form, we extended all of the solvers of the ODE suite to solve problems $M*y'=F(t,y)$ with a mass matrix M that is nonsingular and (usually) sparse. In addition, for all but one solver, the mass matrix M can now be both time- and state-dependent, $M(t,y)$. (As before, the `ode23s` solver allows only constant M.)

The `Mass` property of `odeset` has been enhanced to include new possible values; the old values are still available. For examples, see the M-file help for `fem1ode`, `fem2ode`, or `batonode`.

Singular Mass Matrices and Differential-Algebraic Equations. If the mass matrix is singular, then $M*y' = F(t,y)$ is a differential-algebraic equation (DAE). DAEs have solutions only when $y0$ is consistent, that is, when there is a vector $yp0$ such that $M(t0)*yp0 = F(t0,y0)$. The two solvers `ode15s` and `ode23t` can solve DAEs of index 1 provided that M is not state-dependent and $y0$ is sufficiently close to being consistent.

If there is a mass matrix, you can use `odeset` to set `MassSingular` to `'yes'`, `'no'`, or `'maybe'`. The default of `'maybe'` causes the solver to test whether the problem is a DAE. If it is, the solver treats $y0$ as an initial estimate, attempts to compute consistent initial conditions that are close to $y0$, and proceeds to solve the problem. When solving DAEs, it is advantageous to formulate the problem so that M is diagonal (a semi-explicit DAE). For examples, see the M-file help for `hb1dae` or `amp1dae`.

Changes to Function Functions

These MATLAB function functions have new names and calling sequences to support new functionality.

Old Function Name	New Function Name
`fmin`	`fminbnd`
`fmins`	`fminsearch`

Note that if you have older M-files that use the old names and calling sequences, these calls will generally continue to work. However, the older functions may be removed from MATLAB in future releases, so it is a good idea to revise your code now to use the new names and calling sequences.

Changes to Least Squares Equation Solver

The name of the `nnls` (nonnegative least squares) function was changed to `lsqnonneg`, and its calling sequences have changed as well. These changes have been made to support new functionality.

As noted above, if you have older M-files that use the old names and calling sequences, these calls will generally continue to work.

New Mechanism for Setting Optimization Parameters

MATLAB now has a new mechanism for setting parameters used by the optimization functions. The `options` argument for these functions now takes a structure containing the parameters to set, rather than a vector. This structure is created and modified by a new function, `optimset`. The new `optimget` function extracts the value of a parameter from an `options` structure.

This change affects the new optimization functions only (`fminbnd`, `fminsearch`, and `lsqnonneg`). The older functions (`fmin`, `fmins`, and `nnls`) still use the vector returned by the `foptions` function.

Note that not all of the parameters set with `optimset` apply to every function. Many of the parameters apply only to functions in the Optimization Toolbox.

Changes to cholinc Function

In the `cholinc` function's handling of the Cholesky-Infinity factorization, the input matrix is assumed to be positive semi-definite, so negative pivots are treated as if they were 0. Because of this change, the functional form that includes a second output argument `p` (shown below) is obsolete:

```
[R,p] = cholinc(X,'inf')
```

Programming Enhancements

New evalc Function

The new `evalc` function is an extension of `eval`. Calling `evalc` executes a MATLAB expression and captures any output that would be written to the MATLAB command window in a character array output argument.

New symvar Function

The new `symvar` function searches a MATLAB expression for symbolic variables and returns the names of the variables in a cell array of strings. The identifiers `i`, `j`, `pi`, and other MATLAB constants and function names are ignored.

Enhancements to inline

The `inline` function was improved to better recognize symbolic variable names in a function expression. In addition, multiple variable names are now located.

Enhancements to MATLAB Object-Oriented Programming

Several functions have been added or enhanced to provide additional support for MATLAB object-oriented programming.

Loading and Saving Objects. The new `loadobj` function is an overloadable function called by the `load` command when reading objects from a MAT file into the MATLAB workspace. To define `load` behavior for user objects, create a `loadobj` function in the associated class directory.

The new `saveobj` function is an overloadable function called by the `save` command when writing objects from the MATLAB workspace to a MAT file. To define `save` behavior for user objects, create a `saveobj` function in the associated class directory.

Enhancements to end Statement. You can overload the `end` statement for indexing a user object. To do this, write a method `end.m` in the class directory. The `end` method must have the following calling sequence

```
end(myobj,K,N)
```

where `myobj` is the object, `K` is the index for which you are using the `end` syntax, and `N` is the number of indices in the indexing expression.

Use clear classes to Clear the Class Definition Table. To clear the class definition table, use

```
clear classes
```

This is useful when, during a MATLAB session, you change the way a class is defined.

You should no longer use

```
clear all
```

to clear the class definition table.

Application Program Interface (API) Enhancements

ActiveX Support Enhanced

The ActiveX support enhancements for MATLAB 5.3 include

- Support for interactively using `get` to return a list of properties and `send` to get a list of all events for an interface
- Enhanced data conversion
- Improved event/callback management

The support for ActiveX controls is described in the *Application Program Interface Guide*, in Chapter 7.

MATLAB 5.0 Data Types Supported in the MATLAB 5.3 API Engine

All MATLAB 5.3 data types, including those introduced in MATLAB 5.0 (cell arrays, multidimensional arrays, and structures) are now supported in the MATLAB 5.3 API Engine.

Exploratory MATLAB Java Interface

Intended as a Prototype for Soliciting Your Feedback

Release 11 includes an exploratory MATLAB Java interface. Based on your feedback and additional development and testing efforts, this MATLAB Java interface may be refined and expanded in future releases.

Caution Do *not* use this exploratory version of the MATLAB Java interface for production-level code. This interface will almost certainly change in future releases, and The MathWorks does not commit to ensuring that code written using this exploratory version of the interface will work with future versions of the interface or MATLAB.

What You Can Do with the MATLAB Java Interface

You can use the MATLAB Java interface to:

- Create Java objects in the MATLAB workspace
- Invoke Java methods on objects in the MATLAB workspace
- Invoke Java static methods within MATLAB
- Pass Java objects to MATLAB M-file functions and built-in functions
- Add Java code to toolboxes
- List the methods defined by a Java class

How to Start the MATLAB Java Interface

To start the MATLAB Java interface, at the command line type:

```
java on
```

Development Environment Enhancements

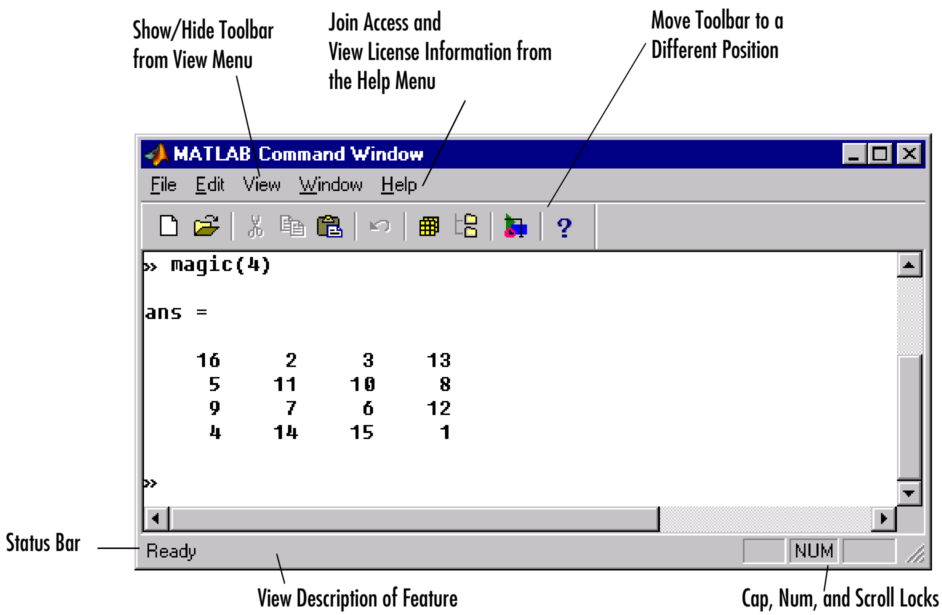
MATLAB 5.3 includes a number of enhancements to these existing development environment tools:

- Command Window
- Array Editor
- Profiler
- Figure Window
- PrintFrame Editor

MATLAB 5.3 also introduces two powerful tools, the Plot Editor and Print Preview.

Enhancements to the Command Window (PC Only)

The MATLAB Command Window for the PC now includes new user interface features.



Show or Hide the Toolbar

To remove the toolbar from the Command Window, select **Toolbar** from the **View** menu, which unchecks it. To display the toolbar, select **Toolbar** from the **View** menu, which checks it.

Note that this feature does not affect the setting for **Show Toolbar** in the **Preferences** dialog box. The preferences setting pertains to the toolbar status when you first start MATLAB.

View License Information

Select **Show License** from the **Help** menu to view license information for your MATLAB configuration.

Join Access

Select **Join MATLAB Access** from the **Help** menu to become a MATLAB Access member. Access helps you stay up-to-date on the latest developments for the MATLAB product family and provides many other benefits. You can become a MATLAB Access member at no cost. To join Access from the **Help** menu, you need to be connected to the Internet. Access membership is not available for the Student Edition of MATLAB.

Dock the Toolbar

You can move the toolbar to another position (that is, undock the toolbar). Click and hold on any separator bar (dividing line between groups of buttons) and then move the toolbar to a new location. To dock the toolbar, move it to an inside edge of the Command Window. The new toolbar position is maintained when you restart MATLAB.

View Description of Feature

When you move the mouse over a toolbar button or menu item, a brief description of the item appears in the left side of the status bar. This provides more information than the tooltip provides.

Cap, Num, and Scroll Locks

In the right side of the status bar is a display area that shows whether the caps, num, and scroll lock keys are active.

Enhancements for the edit Command (UNIX)

The `edit` command uses the MATLAB Editor/Debugger unless you turned off the `builtinEditor` in your `~home/.Xdefaults` file. There is now a way to turn off the `builtinEditor` during a MATLAB session:

```
system_dependent('builtinEditor','off')
```

`edit` then uses the editor defined for your `UNIX$EDITOR` environment variable. To turn the MATLAB Editor/Debugger back on during the session, use

```
system_dependent('builtinEditor','on')
```

You can include the `system_dependent` command in your `startup.m` file or in your `matlabrc.m` file if you have access to it. Type `doc matlabrc` for more information.

Workspace Variables in the Array Editor

The `openvar` function now opens the named workspace variable in the Array Editor for graphical debugging.

Enhanced Display for Structure Members

The MATLAB variable display feature has been enhanced to provide more detailed information for structure members that contain an empty variable or a cell.

For example, this assignment previously displayed the empty matrix (`[]`) for the variable contents. The improved variable display is

```
s.a = zeros(0,4)

s =

    a: [0x4 double]
```

This assignment to a cell previously displayed only the cell class and size, as shown here.

```
s.c = {[4 5 6] 'foo'}

s =

     c: {1x2 cell}
```

The improved variable display is

```
s.c = {[4 5 6] 'foo'}

s =

     c: {[4 5 6]  'foo'}
```

If the contents of the cell cannot be displayed on one line, MATLAB displays the variable using the cell class and size as in previous versions of the software.

Enhanced MATLAB Profiler

The MATLAB profiler was significantly enhanced to expand data collection about function performance. The profiler improvements include:

- Simultaneous data collection for all functions
- More information about each function, including: number of calls, list of parent functions, list of child functions, and execution count and execution time for each line of code
- Optional recording of function call history
- Report generation in HTML format

Use the `profile` command to start the profiler.

New profreport Function

You can use the new `profreport` command to generate a report of the function call statistics logged by the M-file profiler. The report is in HTML format and is displayed in your Web browser. You can generate a report for the current profiler session or for statistics that were saved in an earlier session.

Figure Window Enhanced

The figure window has been enhanced significantly for MATLAB 5.3:

- The **File** menu now includes options for
  - Saving a figure using a standard graphics file format (e.g., TIFF)
  - Setting up a page for printing
- A **Tools** menu has been added
- A new toolbar has been added

The **Tools** menu and the new toolbar give you access to the new Plot Editor.

For more details, see "Figure Window Enhancements" later in this chapter.

PrintFrame Editor Enhancements

While the functionality for the PrintFrame Editor has not changed, its user interface has a new look. For more details, see the "New Look for the PrintFrame Editor" section later in this chapter.

Online Documentation Enhancements

Some User's Guides Available in HTML Form

The following User's Guides are available in HTML form for Release 11:

- *Using MATLAB Graphics*
- *Using Simulink*
- *Stateflow User's Guide*
- *Report Generator User's Guide*
- *Database Toolbox User's Guide*

Getting Started with MATLAB, *Release 11 New Features*, and *Release 11 Known Software and Documentation Problems*, as well as documentation for the Plot Editor, PrintFrame Editor, and Page Setup dialog box, are also available in HTML form.

This allows these documents to take advantage of links to the reference material, as well as additional online navigation features.

Microsoft HTML Help Viewer

On PCs, *Using Simulink*, the *Stateflow User's Guide*, and the *MATLAB Report Generator User's Guide,* as well as the Plot Editor and PrintFrame Editor documentation, when accessed via **Help** menus or buttons from within the respective product, use the Microsoft HTML Help Viewer. That viewer is provided with Internet Explorer Version 4.01 and higher. If you access these documents via the Help Desk, they are displayed using your system's Web browser.

The Microsoft HTML Help Viewer provides a two-pane help display mechanism: one for navigation, and the other for display of the text. The navigation pane includes a collapsible/expandable table of contents and an index tab and a search tab.

If you do not have the HTML Help Viewer, you can get it at no cost by downloading and installing the minimum configuration for Internet Explorer from the Microsoft Web site – `http://www.microsoft.com/ie/download/`.

On PCs that do not have Microsoft's HTML Help Viewer installed, and on UNIX platforms, these HTML documents use your system's Web browser. The Web browser interface uses a two-pane interface, but the table of contents is not collapsible/expandable and it does not have the search tab.

Context-Sensitive Help

There is a new form of help available for the dialog boxes in the Plot Editor and the **Page Setup** dialog box. For these dialog boxes, click the **Help** button in a dialog box to go directly to help for that dialog box. This context-sensitive help uses Microsoft HTML Help Viewer, if available, as described above.

Context-sensitive help is also available from dialog boxes or within tools, via **Help** buttons or menus, for these products:

- MATLAB Report Generator and Simulink Report Generator
- Simulink
- Stateflow
- Fixed-Point Blockset

Japanese Interface

The interfaces to MATLAB 5.3 have been translated into Japanese. This includes the Editor/Debugger, the figure window, and command line help. The interface for products in Release 11 other than MATLAB have not been translated.

As with MATLAB 5.2, the Help Desk has also been translated into Japanese.

Visualization Enhancements

Figure Window Enhancements

Accessing Off-Screen Visible Figures

You can use the new `findfigs` function to find all visible figures that are positioned completely off-screen and make them visible on screen, at the top left corner of the screen.

New Menu Items in the Figure Window

When you create a plot, the **File** menu in the figure window now includes three new items:

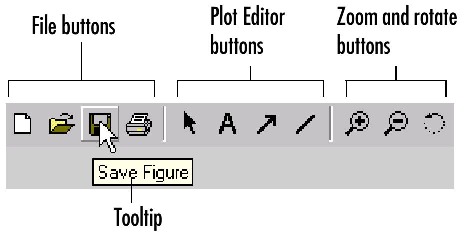
- **Export**: saves a figure using a standard graphics file format, such as TIFF or EPS.
- **Property Editor**: allows you to modify any property of any Handle Graphics object.
- **Page Setup**: replaces **Page Position**; for more information, click the **Help** button in the **Page Setup** dialog box.
- **Print Preview**: displays a preview of how the figure will appear on the printed page.

A new **Tools** menu now appears in the menu bar of the figure window. It contains menu items for the Plot Editor and for zoom and rotate functions, which are described below.

For UNIX platforms, the Figure Window menu bar has been enhanced to match the functionality of the PC Figure Window menu bar.

New Toolbar in Figure Window

When you create a plot, the figure window now includes a toolbar for quick access to popular features that also appear in the **File** menu. Position the cursor over a button, and a tooltip describing that feature appears.



The zoom in and out buttons allow you magnify or reduce the size of the figure. For 2-D plots, the zoom buttons use the `zoom` command, and for 3-D plots, they use the `camzoom` command.

The rotate button rotates a 3-D plot, using the `rotate3d` command.

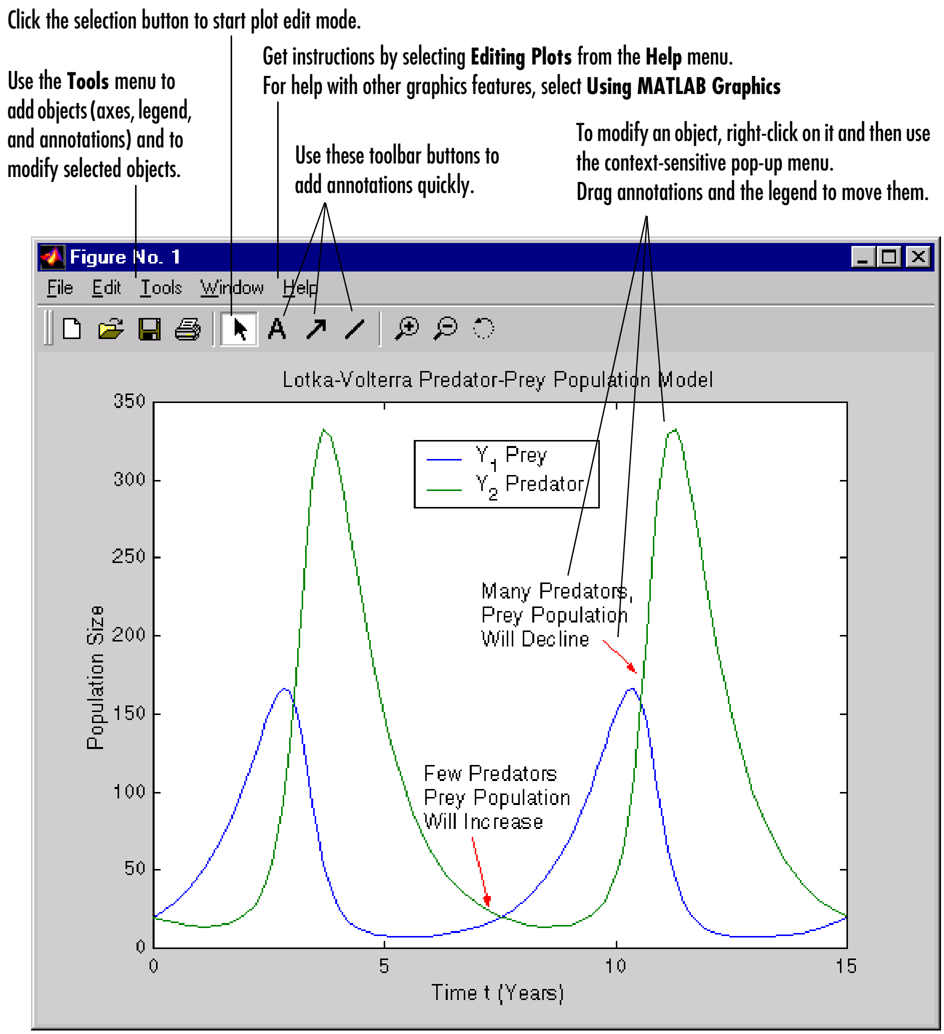
The Plot Editor is an easy-to-use tool you use to add and modify:

- Text, arrow, and line annotations
- Axes labels, title, legend, tick steps, and grid
- Plot line properties such as line style, thickness, color, and marker

The Plot Editor is described in more detail in the next section.

The Plot Editor

This illustration shows the main features of the Plot Editor.



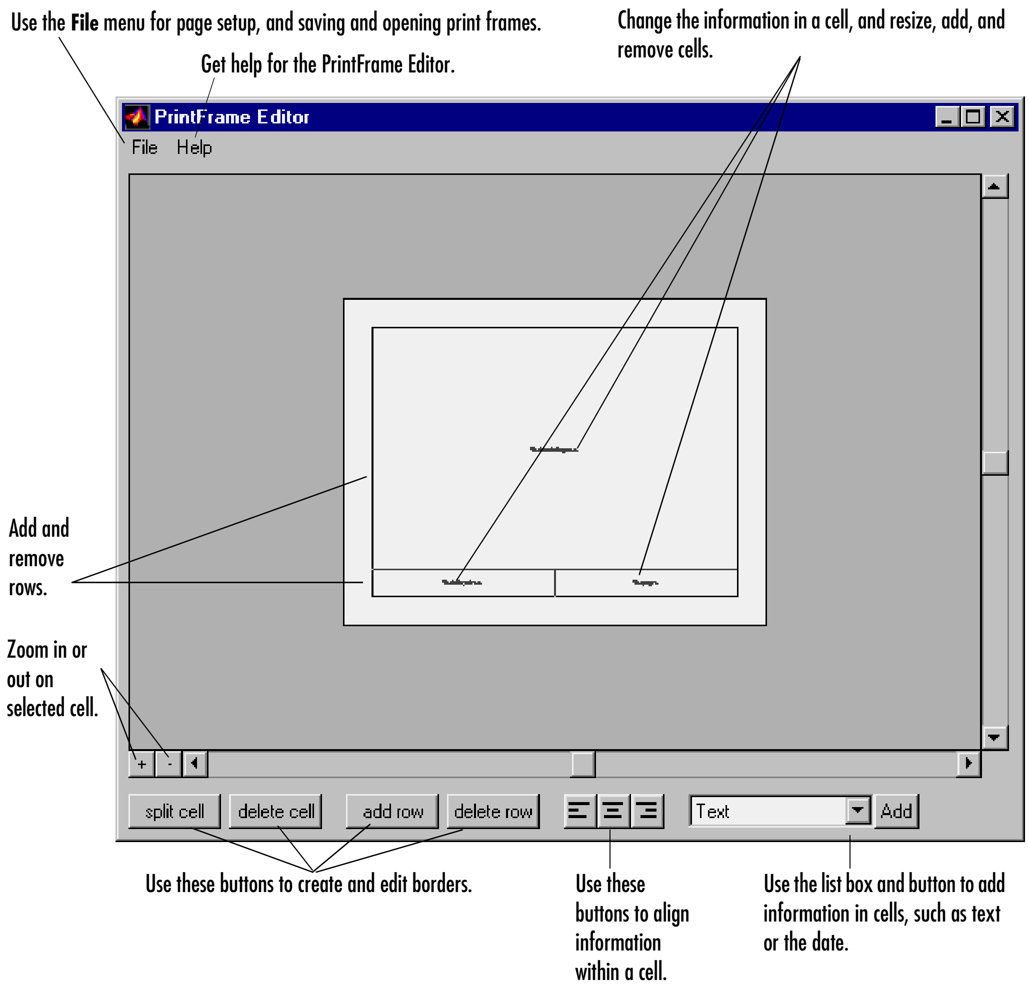
New Context-Sensitive Help

There is a new form of help available for the dialog boxes in the Plot Editor and the **Page Setup** dialog box. For these dialog boxes, click the **Help** button in a dialog box to go directly to help for that dialog box.

See the “Online Documentation Enhancements” earlier in this chapter for details about context-sensitive help.

New Look for the PrintFrame Editor

While the functionality for the PrintFrame Editor has not changed, its user interface has a new look. In addition, help for the PrintFrame Editor now is available directly from the **Help** menu. To access the PrintFrame Editor, use the `frameedit` command.



Support for HDF/EOS Development Tools

MATLAB 5.3 provides three additional functions that act as gateways to the Hierarchical Data Format/Earth Observing System (HDF/EOS), for grid, point, and swath objects.

HDF/EOS, an extension of the NCSA (National Center for Supercomputing Applications) HDF standard, is the scientific data format standard selected by

NASA as the baseline standard for EOS. The functions in the HDF-EOS C library are developed and maintained by EOSDIS (Earth Observing System Data and Information System).

The new MATLAB functions are listed below.

Function	Description
`hdfgd`	HDF-EOS GD (grid) interface
`hdfpt`	HDF-EOS PT (point) interface
`hdfsw`	HDF-EOS SW (swath) interface

In addition to the MATLAB online help for HDF functions, you should also have the document *HDF-EOS Library User's Guide for the ECS Project, Volume 1: Overview and Examples* and *Volume 2: Function Reference Guide*. This document is available on the Web at `http://hdfeos.gsfc.nasa.gov`. If you are unable to obtain the document from this location, please contact MathWorks Technical Support (`support@mathworks.com`).

New Histogram Function

You can use the new `histc` function for binning vector elements. `histc` differs from the `hist` function in that it uses bin edges to define the bins. The output vector can be plotted with the `bar` function.

New Plotting Functions

MATLAB 5.3 provides a set of new plotting functions to graph mathematical expressions. Key features of these functions include:

- Direct evaluation of symbolic expressions
- Automatic labeling employing mathematical symbols
- Improved axis scaling
- Elimination of singularities of mathematical expressions in the graph

These functions provide easy to use plotters.

Function	Purpose
ezcontour	Contour plotter
ezcontourf	Filled contour plotter
ezmesh	3-D mesh plotter
ezmeshc	Combination mesh/contour plotter
ezplot	Function plotter
ezplot3	3-D parametric curve plotter
ezpolar	Polar coordinate plotter
ezsurf	3-D colored surface plotter
ezsurfc	Combination surf/contour plotter

New Volume Visualization Functions

MATLAB supports a set of new functions for visualizing 3-D scalar and vector data.

Function	Purpose
coneplot	Plot velocity vectors as cones in a 3-D vector field
contourslice	Draw contours in volume slice planes
isocaps	Compute isosurface end-cap geometry
isonormals	Compute normals of isosurface vertices
isosurface	Extract isosurface data from volume data
reducepatch	Reduce the number of patch faces
reducevolume	Reduce the number of elements in a volume data set
shrinkfaces	Reduce the size of patch faces

Function	Purpose
smooth3	Smooth 3-D data
stream2	Compute 2-D stream line data
stream3	Compute 3-D stream line data
streamline	Draw stream lines from 2-D or 3-D vector data
surf2patch	Convert surface data to patch data
subvolume	Extract subset of volume data set

findobj More Flexible

The findobj function now accepts any property value that is allowed with set. For example,

```
findobj('Type','line','Color',[1 0 0])
```

can now be written

```
findobj('Type','line','Color','r')
```

Rectangle Object Added

MATLAB 5.3 adds a new rectangle Handle Graphics object. Use the rectangle function to create a rectangle object.

legend Enhancements

MATLAB 5.3 enhances the legend function to:

- Support multiline labels, allowing you to wrap long labels
- Integrate with the Plot Editor

To support these enhancements, MATLAB 5.3 treats the legend text as one text object, grouping all the text together.

New Figure Properties

DoubleBuffer Figure Property

Figure objects have a new property called `DoubleBuffer`, which accepts the values `on` and `off`, with `off` being the default. Double buffering works only when the figure `Renderer` property is set to `painters`.

Double buffering is the process of drawing to an off-screen pixel buffer and then displaying (blitting) the buffer contents on the screen once the drawing is complete. Double buffering generally produces flash-free rendering for simple animations (such as those involving lines, as opposed to objects containing large numbers of polygons). Use double buffering with the animated objects' `EraseMode` property set to `normal`. Use the `set` command to enable double buffering:

```
set(figure_handle,'DoubleBuffer','on')
```

XDisplay, XVisual, XVisualMode Properties - UNIX Only

XDisplay. You can display a figure window on a different display using the `XDisplay` property. For example, to display the current figure on a system called `fred`, use the command:

```
set(gcf,'XDisplay','fred:0.0')
```

XVisual. You can select the visual used by MATLAB by setting the `XVisual` property to the desired visual ID. This can be useful if you want to test your application on an 8-bit or grayscale visual. To see what visuals are available on your system, use the UNIX `xdpyinfo` command. From MATLAB type:

```
!xdpyinfo
```

The information returned contains a line specifying the visual ID. For example:

```
visual id:    0x21
```

To use this visual with the current figure, set the `XVisual` property to the ID:

```
set(gcf,'XVisual','0x21')
```

XVisualMode. `XVisualMode` can take on two values – `auto` (the default) and `manual`. In `auto` mode, MATLAB selects the best visual to use based on the number of colors, availability of the OpenGL extension, etc. In `manual` mode, MATLAB does not change the visual from the one currently in use. Setting the `XVisual` property sets this property to `manual`.

New FontName Property Value

The text and axes `FontName` properties accept a new value of `fixedwidth`. When `FontName` is set to `fixedwidth`, MATLAB uses the font name defined by the new root property `FixedWidthFontName`, which is `Courier` by default.

uint16 CData for Images

You can now define images with `CData` of class `uint16`.

Support for Portable Network Graphics Images

MATLAB can read or write images stored in the Portable Network Graphics (PNG) format. The `imread`, `imwrite`, and `imfinfo` functions can now handle files stored in any of the following PNG formats:

- 1-bit, 2-bit, 4-bit, 8-bit, and 16-bit grayscale images
- 8-bit and 16-bit indexed images
- 24-bit and 48-bit RGB images

GUI Development Enhancements

Support for BackgroundColor for Push Buttons (PC only)

On the PC, you can now control the color displayed within the push button rectangle by specifying the `BackgroundColor` property.

Support for Fixed-Width Fonts

To use a fixed-width font that looks good in any locale (and displays properly in Japan) where multibyte fonts are used, set `FontName` to the string `FixedWidth` (this string is case sensitive):

```
set(uicontrol_handle,'FontName','FixedWidth')
```

MATLAB Compiler 2.0

Summary of New Features

MATLAB Compiler 2.0 supports much of the functionality of MATLAB 5. The new features of the Compiler are:

- Data constructs
  - Multidimensional arrays
  - Cell arrays
  - Structure arrays
  - Sparse arrays
- Programming tools
  - Variable input and output argument lists (`varargin`/`varargout`)
  - `try … catch … end`
  - `switch … end`
- Language enhancements
  - Persistent variables
  - `load` and `save` commands
- Improved Compiler options
- Macro options
- Error/warning messages
- Improved `mex` and `mbuild` scripts
- Stand-alone Compiler

Data Constructs

Multidimensional Arrays

Multidimensional arrays in MATLAB are an extension of the two-dimensional matrix. You access a two-dimensional matrix element with two subscripts: the first represents the row index and the second represents the column index. In multidimensional arrays, use additional subscripts for indexing. For example, a three-dimensional array has three subscripts and a four-dimensional array has four subscripts.

Cell Arrays

Cell arrays are a special class of MATLAB arrays where elements, or *cells*, contain MATLAB arrays. Cell arrays allow you to store dissimilar classes of arrays in the same array.

Structure Arrays

Structures are a class of MATLAB arrays that can store dissimilar arrays together. Structures differ from cell arrays in that you reference them by named fields.

Sparse Arrays

Sparse arrays provide an efficient representation of arrays that contain a significant number of zero-valued elements.

Programming Tools

Variable Input Arguments

The special argument `varargin` can be used to pass any number of input arguments to a function.

Variable Output Arguments

The special argument `varargout` can be used to return any number of output arguments from a function.

try ... catch ... end

In an M-file, the statements between `try` and `catch` are executed until an error occurs. Then, the statements between `catch` and `end` are executed. If no error occurs, the statements between `catch` and `end` are not executed.

switch ... end

The `switch` statement lets you conditionally execute code depending on the value of a variable or expression.

Language Enhancements

Persistent Variables

Variables that are defined as persistent do not change value from one call to another. Persistent variables may be used within a function only and they remain in memory until the M-file is cleared or changed.

load and save Commands

The support for `load` and `save` has been enhanced to include loading into a structure.

Improved Compiler Options

The collection of new and improved options provides you with greater flexibility to control Compiler 2.0. You also have full access to Compiler 1.2 and its set of existing options.

Macro Options

These options (`-m`, `-p`, `-x`, and `-S`) let you quickly and easily generate C and C++ stand-alone applications, and MATLAB and Simulink C MEX-files, respectively. Each macro replaces a sequence of several Compiler options making it much easier to generate your output.

Error/Warning Messages

The MATLAB Compiler 2.0 contains a comprehensive set of error and warning messages that help you isolate problems with your code.

Improved mex and mbuild Scripts

The `mex` script, which allows you to compile MEX-functions, and the `mbuild` script, which allows you to customize the building and linking of your code, have been enhanced to automatically search your system for supported third-party compilers.

Stand-Alone Compiler

You can run the MATLAB Compiler 2.0 from the DOS or UNIX command line, making it unnecessary to have MATLAB running on your system. You can call the Compiler directly from a makefile. This stand-alone MATLAB Compiler is faster than previous versions of the Compiler because it does not have to start MATLAB each time you invoke a compilation.

MATLAB C/C++ Math Library 2.0

MATLAB C Math Library 2.0

Summary of New Features

The MATLAB C Math Library 2.0 supports these new features:

- Over 60 new functions
- Automated memory management for temporary arrays
- Data types
  - Multidimensional arrays
  - Cell arrays
  - MATLAB structures
  - Sparse matrices
- New indexing functions
- Variable input and output argument lists (`varargin`/`varargout`)
- try blocks and catch blocks
- Improved `mbuild` script

Over 60 New Functions

New functions in the library support multidimensional arrays, cell arrays, MATLAB structures, and sparse arrays.

Automated Memory Management for Temporary Arrays

The functions `mlfAssign()`, `mlfEnterNewContext()`, `mlfRestorePreviousContext()`, and `mlfReturnValue()` provide automated memory management for *temporary* arrays. Using these functions, you can embed calls to library functions as function arguments. You don't need to declare `mxArray*` variables to store temporary values, or explicitly delete those temporary arrays.

Data Types

Multidimensional Arrays. Multidimensional arrays in MATLAB can have more than two dimensions. You access a two-dimensional matrix element with two indices: a row index and a column index. In a multidimensional array, use additional subscripts for indexing. For example, a three-dimensional array has three indices and a four-dimensional array has four.

Cell Arrays. Cell arrays are a special class of MATLAB arrays where elements, or *cells*, contain MATLAB arrays. Cell arrays allow you to store dissimilar classes of arrays in the same array. Cell arrays can be multidimensional.

MATLAB Structures. MATLAB structures are a class of MATLAB arrays that can store dissimilar arrays together. MATLAB structures differ from cell arrays in that you reference them by named fields. Structure arrays can be multidimensional.

Sparse Matrices. Sparse arrays are two-dimensional matrices that contain a significant number of zero-valued elements. An efficient matrix storage format stores only the nonzero values.

New Indexing Functions

Three new indexing functions handle indexing for n-dimensional arrays, including cell arrays, structure arrays, and sparse arrays:

- `mlfIndexRef()`
- `mlfIndexAssign()`
- `mlfIndexDelete()`

Calls to the Version 1.2 indexing functions, `mlfArrayRef()`, `mlfArrayAssign()`, and `mlfArrayDelete()`, are still valid. However, you must use the new indexing functions to access support for multidimensional indexing, cell array indexing, structure indexing, and sparse indexing.

Variable Input and Output Argument Lists

MATLAB `varargin` functions accept any number of input arguments. The library supports `varargin` functions through standard ANSI C variable-length argument lists.

MATLAB varargout functions return any number of output arguments. The library supports varargout functions through the functions mlfVarargout() and mlfIndexVarargout().

try and catch Blocks

The library's mlfTry and mlfCatch macros let you handle errors with try and catch blocks.

Improved mbuild Script

The mbuild script automatically detects the location of your C/C++ compiler and determines whether you are compiling C or C++ code. You no longer need to use the -setup option to configure mbuild; however, it is available if you need to change compilers or customize the options that mbuild uses.

mbuild now creates DLLs in addition to executables.

MATLAB C++ Math Library 2.0

Summary of New Features

The MATLAB C++ Math Library 2.0 supports these new features:

- Over 60 new functions
- Data types
  - Multidimensional arrays
  - Cell arrays
  - MATLAB structures
  - Sparse matrices
- New indexing functions
- Variable input and output argument lists (varargin/varargout)
- Improved mbuild script

Over 60 New Functions

New functions in the library support multidimensional arrays, cell arrays, MATLAB structures, and sparse arrays.

Data Types

Multidimensional Arrays. Multidimensional arrays in MATLAB can have more than two dimensions. You access a two-dimensional matrix element with two indices: a row index and a column index. In a multidimensional array, use additional subscripts for indexing. For example, a three-dimensional array has three indices and a four-dimensional array has four.

Cell Arrays. Cell arrays are a special class of MATLAB arrays where elements, or *cells*, contain MATLAB arrays. Cell arrays allow you to store dissimilar classes of arrays in the same array. Cell arrays can be multidimensional.

MATLAB Structures. MATLAB structures are a class of MATLAB arrays that can store dissimilar arrays together. MATLAB structures differ from cell arrays in that you reference them by named fields. Structure arrays can be multidimensional.

Sparse Matrices. Sparse arrays are two-dimensional matrices that contain a significant number of zero-valued elements. An efficient matrix storage format stores only the nonzero values.

New Indexing Functions

The `mwArray` member function `cell()` implements indexing into cell arrays.

The `mwArray` member function `field()` implements indexing into structure arrays.

Variable Input and Output Argument Lists

MATLAB `varargin` functions accept any number of input arguments. The library supports `varargin` functions through the `mwVarargin` class.

MATLAB `varargout` functions return any number of output arguments. The library supports `varargout` functions through the `mwVarargout` class.

Improved mbuild Script

The `mbuild` script automatically detects the location of your C/C++ compiler and determines whether you are compiling C or C++ code. You no longer need to use the `-setup` option to configure `mbuild`; however, it is available if you need to change compilers or customize the options that `mbuild` uses.

Simulink 3.0

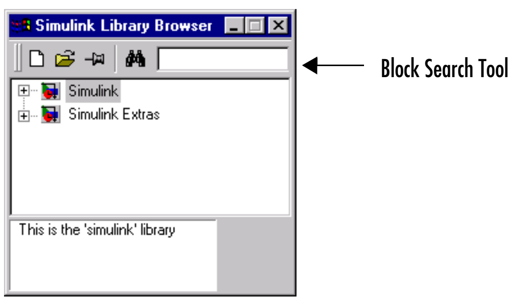
Simulink 3.0 introduces many significant enhancements in the following areas:

- User interface
- Blocks
- Modeling
- S-functions
- Simulation
- Printing
- SB2SL 2.0

User Interface Enhancements

Library Browser (PC Only)

On the PC, Simulink 3.0 provides a library browser, a tree-structured view of all block libraries installed on your system. The browser enables you quickly to locate and copy any library block into your model. Simulink displays the library browser when you start Simulink.



See "Browsing Block Libraries" in Chapter 3 of *Using Simulink*.

Model Browser (PC Only)

On the PC, Simulink 3.0 model windows optionally display a model browser. To display the browser, select **Model Browser** from the Simulink **View** menu. See "The Model Browser" in Chapter 3 of *Using Simulink*.

Block Data Tips (PC Only)

On the PC, Simulink 3.0 displays information about a block in a pop-up window when you hover the pointer over the block in the diagram view. To disable this feature or control what information a data tip includes, select **Block Data Tips** from the Simulink **View** menu.

Zoomable Diagram View

Simulink 3.0 allows you to enlarge or shrink the view of the block diagram in the current Simulink window. See "Zooming Block Diagrams" in Chapter 3 of *Using Simulink*.

New Standard Dialog Button Layout

Simulink 3.0 changes the names and layout of the standard set of buttons that appear on all dialog boxes. Previously, all dialog boxes contained the following buttons: **Apply**, **Revert**, **Help**, and **Close**. These are now **Ok**, **Help**, **Cancel**, and **Apply**. **Ok** applies any changes and dismisses the dialog box. **Cancel** dismisses the dialog box without applying any changes. **Apply** applies any changes without dismissing the dialog box.

Recreating Saved Data in a Handle Graphics Window

The new Simplot tool (invoked with the `simplot` command) recreates saved data in a Handle Graphics window. This provides an easy way to interact with the saved data and add annotations, etc., to plots. Some of the key features of the Simplot tool include:

- Plot Simulink output data, producing scope-like graphics for all data produced by Simulink output blocks. This includes matrices and structures with or without time data
- Compare multiple runs
- Plot difference between runs
- Specify separate time vector
- Obtain handles for all graphics objects.

Dynamic Masked Dialogs

Simulink 3.0 permits creation of dynamic masked dialogs, that is, dialogs that alter their appearance in response to changes in control settings. Dynamic masked dialogs permit you to replace several similar static dialogs with a single dynamic dialog. See "Creating Dynamic Dialogs for Masked Blocks" in Chapter 6 of *Using Simulink*.

Masked Dialog Parameter Limit Increased

Simulink 3.0 lets you define masked dialogs having as many as 100 parameters.

New Mask Display Command: port_label

Simulink 3.0 introduces a new mask display command, `port_label`, that lets you specify the labels of ports displayed on the icon. See "Displaying Text on the Block Icon" in Chapter 6 *Using Simulink*.

Signal Properties Dialog

The **Signal Properties** dialog allows you to view and edit properties of signals. See "Setting Signal Properties" and "The Signal Properties Dialog" in Chapter 3 of *Using Simulink*.

Block Enhancements

Reorganized Block Library

The Simulink 3.0 block library has a new organization that categorizes many blocks differently than the old organization did. Use the Library Browser (see "Library Browser" in Chapter 3 of *Using Simulink*) to obtain a quick overview of the new library structure.

Masked S-Function Blocks

Simulink 3.0 allows masking of the S-Function and Subsystem blocks.

Images in Masked Dialogs

Simulink 3.0 introduces a new pair of masked dialog functions, `image` and `patch`, that enable you to display bitmapped images and draw patches on masked block icons. See "Displaying Images on Masks" in Chapter 6 in *Using Simulink*.

New Blocks

Simulink 3.0 provides several new blocks. Each of these is described in more detail in Chapter 8 of *Using Simulink*.

Block	Description
Bus Selector	Selects a subset of bus signals from a bus defined by a Mux or another Bus Selector block. You can use the Bus Selector's parameters dialog box to specify which signals to select.
Configurable Subsystem	Allows you to choose from a library of alternative implementations of a subsystem
Function-Call Generator	Allows a model to execute a function-call subsystem at a specified rate
Probe	Outputs a signal's width, sample time, and type (real or complex)

Enhanced Blocks

Simulink 3.0 also enhanced several blocks. Each of these is described in more detail in Chapter 8 of *Using Simulink*.

Clock Block. The icon of the Simulink 3.0 Clock block optionally displays the current simulation time.

Mux Block. The Simulink 3.0 Mux block incorporates new features that allow it to function as a data bus. The new features are:

- The output of a Mux block is a bus whose signals have names corresponding to the labels of input lines. The names of signals corresponding to unlabeled inputs default to `SignalN` where `N` is the number of the input.
- You can enter a comma-separated list of signal names as the value of the Mux block's **Number of Inputs** parameter.
- The Mux block has a new parameter, **Display option**, that affects the block icon. The new parameter can have the following values:
    - `none` (the default) displays Mux
    - `signals` displays the signal labels next to each port
    - `bar` displays the block as a solid bar

Round Sum Block. Simulink 3.0 allows you to choose a circular or rectangular shape for a sum block, whichever is appropriate for your environment. To change a sum block's shape, open its parameters dialog and select the desired shape from the **Icon Shape** drop-down list. Select the **Apply** or **Ok** button to apply the change.

Simulink 3.0 also allows you to manipulate the position of sum block input ports by inserting spacers between ports.

Scope Block. The Simulink 3.0 Scope block has the following new features:

- Multiple ports and axes. All axes share the same time base, but have independent *y*-scales.

  To set the number of axes, select the **Properties** button on the Scope's toolbar.
- No limit on the number of traces (except for the floating scope).
- New data structure gets written to the workspace when logging data.
- Property dialogs for each axis are opened via the context menu (right click) on the axes.
- Zoom handles multiaxes. The basic rule is that all axes must always share the same *x*-limits. So, if you zoom one axes, the *x*-limits of all other axes are modified to match.

- Axes can be given titles. If desired, these titles can be inherited from signal names.

Modeling Enhancements

New Data I/O Formats

Simulink now allows structures and lists of structures and/or matrices to be used for the input and output of data to and from either files or the MATLAB workspace. In previous versions of Simulink, input and output data could be only in matrix form. Blocks that support the new data I/O formats include:

- Outport
- Inport
- From File
- To File
- From Workspace
- To Workspace

For more information, see "The Workspace I/O Page" in Chapter 4 and "To Workspace" in Chapter 8 of *Using Simulink*.

Data Type Conversion

The Data Type Conversion block allows you to convert a signal of one data type (e.g., `float`) to a signal of another type (e.g., `int32`). See "Data Type Conversion" in Chapter 8 in *Using Simulink*.

Data Types

Simulink 3.0 supports multiple data types for most signal and block parameter values. See "Data Types" in Chapter 3 in *Using Simulink*.

Complex/Real Conversions

Simulink 3.0 provides blocks for converting complex signals to real signals, and vice versa. See the following sections in Chapter 8 of *Using Simulink*:

- "Complex to Magnitude-Angle"
- "Complex to Real-Imag"
- "Magnitude-Angle to Complex"
- "Real-Imag to Complex"

Version Control

Simulink 3.0 tracks changes to a model and optionally displays version information, including that maintained by an external version control system, in an annotation block in a model's diagram. For more information, see "Tracking Model Versions" in Chapter 3 and "Model Information" in Chapter 8 of *Using Simulink*.

S-Function Enhancements

S-functions support has been enhanced. New features include:

- Multiple ports
- Port-based sample times — you can independently set input and output port sample times
- Tighter integration with the Real-Time Workshop code generation process
- Wrapper S-functions — eliminates calls through a function pointer
- Fully embedded (inlined) S-functions — eliminates the function call to your algorithm

Port-Based Sample Times Supported for S-Functions

Simulink supports port-based sample times for S-functions. This feature allows you to set sample times for input and output ports of your S-functions independently.

Use port-based sample times if your application requires unequal sample rates for input and output execution or if you don't want the overhead associated with running input and output ports at the highest sample rate of your block (which is the case with block-based sample times).

Simulation Enhancements

Reduced Memory Requirement

Simulink 3.0 reuses block I/O memory buffers to reduce simulation memory requirements. You can turn this feature off to facilitate debugging a model. See “Disable Optimized I/O Storage” in Chapter 4 of *Using Simulink*.

Simulation Error Navigation

Clicking on an error message in the simulation error dialog displays the block that caused the error. See “Simulation Errors Dialog” in Chapter 4 of *Using Simulink*.

Printing Enhancements

Simulink 3.0 printing has been enhanced to now support TIFF previews in EPS files. Also, large models now print without resource leaks.

SB2SL 2.0

The SB2SL Version 2.0 reduces the work of migrating from Xmath® and SystemBuild™ to MATLAB and Simulink. The translator reads a SystemBuild Version 5 ASCII format model file and creates a Simulink model that represents the structure and hierarchy of the SystemBuild model. Xmath data from the SystemBuild model is translated into MATLAB workspace variables.

SB2SL is available to you free if you are licensed for Simulink. To install this feature with Simulink, look for the **SB2SL** entry in the installation script interface when you are installing MATLAB and Simulink.

Stateflow 2.0

Stateflow 2.0 introduces many significant enhancements in the following areas:

- GUIs
- Modeling
- Code generation

GUI Enhancements

Enhanced Debugger User Interface

Drop down option lists in the debugger dialog allow you to quickly choose the kind and scope of debug information to display when running a model. For example, you can choose to show active states for all charts or only loaded charts, all data or only watched data, all breakpoints or only breakpoints for loaded charts, and so on.

Enhanced Explorer Interface

The Stateflow 2.0 Explorer allows you to edit state and data properties displayed in the Explorer's content pane. You no longer have to bring up a dialog box in order to change state or data properties. The Stateflow 2.0 Explorer allows you to apply property changes to groups of objects as well as individual objects. You can also create objects by copying other objects.

Chart Styles

Stateflow 2.0 allows you to define and apply chart styles to a chart. A chart style specifies the colors of various chart elements, such as states, transitions, backgrounds, and so on. A chart style also specifies the font used to render labels for states, transitions, and other chart objects. A chart style allows you to specify the colors and fonts of all chart elements with a single menu selection. Stateflow 2.0 comes with nine standard styles: Classic, Antique, Rose, GrayScale, Neon, Desert, Slate, Valerie, Factory.

Enhanced Target Builder Interface

The new Stateflow 2.0 target builder interface greatly speeds and simplifies the process of specifying automatic code generation and custom code options for building simulation and stand-alone targets.

Modeling Features

Boxes

Boxes are graphical groupings of objects that can be cut, pasted, moved, and annotated as a unit. Boxes allow you to break a large, complex chart into more manageable modules without having to introduce additional states into your model.

Chart Libraries

Stateflow 2.0 allows you to create and use chart libraries. A chart library is a Simulink block library that includes Stateflow charts. You can include charts from a library in a model by cutting-and-pasting or dragging-and-dropping the charts from the library to the model. Chart libraries facilitate chart reuse. For example, updating a chart in a library automatically updates all instances of the chart included in Stateflow models.

Arrays

Stateflow 2.0 models can define and manipulate data arrays having an arbitrary number of dimensions.

Support for Simulink Data Types

Stateflow 2.0 supports the same set of data types as Simulink 3.0. Stateflow checks to ensure that the data types of input and output ports match the data types of the Simulink blocks to which those ports are connected.

Directed Broadcasting of Implicit Events

Implicit events, such as entering a chart or state, previously woke up the entire chart in which they occurred. In Stateflow 2.0, implicit events wake up only states that listen for them. This speeds up execution of a model.

Enhance Model Printing

The Stateflow 2.0 **Print Book** option on the **File** menu of the Chart Editor generates a detailed, cross-referenced report on the Stateflow components of a Stateflow model, including the diagram of each chart in the model and the properties of each chart's elements. The command supports both PDF (Adobe Acrobat) and PostScript output formats. The optional Simulink Report Generator allows you to produce a comprehensive report on a model that includes Simulink as well as Stateflow blocks.

Code Generation

Incremental Code Generation

When generating code from a model, Stateflow 2.0 generates code only for charts that have changed nongraphically since the last time code was generated. In particular, Stateflow 2.0 codes and builds each chart as a separate module when generating a simulation target. Each chart is coded and built only if the chart has changed in such a way as to affect code generation since the last time the chart was built. This dramatically speeds the rebuilding of large models that have changed slightly.

Coder Optimizations

The Stateflow 2.0 code generator removes dead code and inlines functions to reduce the size and increase the speed of targets significantly.

The Real-Time Workshop 3.0

External Mode

External mode has been enhanced. A new comprehensive GUI allows you to:

- Download parameters for on-the-fly tuning
- Upload data to your simulation
- Save data in MAT-file format to:
  - sequential files
  - sequential directories
- Introduce triggers (with or without delays) into data collection

Code Generation for Embedded Applications

The Real-Time Workshop supports code generation for embedded applications. The generated code has these features:

- Highly optimized — block I/O optimizations (including buffer reuse and local block outputs), invariant signals, and inlined parameters
- Support for interfacing of parameters and signals
- More readable code — contains signal and parameter names
- Interfaces with existing handwritten code
- Integrated tightly with Stateflow Coder
- Open and customizable — new target configuration options

Real-Time Workshop S-Function Target

This target (code format) generates your model as an S-function. This target has these advantages:

- Incremental code generation
- Speeding up simulation
- Sharing the model with other users without providing the source code. This is useful if your code or algorithm is proprietary.
- Code reuse by multiply instantiating one model inside another

mdlRTW Supports Data Typing

The `mdlRTW` routine now supports data typing.

Simulink Data Types

The Real-Time Workshop is fully compatible with Simulink data types.

Real-Time Workshop Ada Coder 3.0

Note The Real-Time Workshop Ada Coder is a separate product from the Real-Time Workshop.

The Real-Time Workshop Ada Coder supports generation of Ada code.

Features

Like the Real-Time Workshop, the Real-Time Workshop Ada Coder provides a real-time development environment that features:

- A rapid and direct path from system design to hardware implementation
- Seamless integration with MATLAB and Simulink
- A simple, easy to use interface
- An open and extensible architecture

Restrictions

The Real-Time Workshop Ada Coder has the same constraints imposed upon it as the C version of the Real-Time Workshop. The code generator does not produce code that solves algebraic loops, and Simulink blocks that are dependent on absolute time can be used only if the program is not intended to run for an indefinite period of time.

There are additional constraints for the Ada code generation. The Real-Time Workshop Ada Coder:

- Does not support nonreal-time variable step integration models
- Does not support blocks that contain continuous states (for example, the continuous time integration block)
- Does not provide an Ada interface for interactive real-time parameter tuning (you must use the C interface)

Real-Time Windows Target 1.0

The Real-Time Windows Target 1.0 allows you to run C code generated by the Real-Time Workshop on a PC in real time. For details, see “Real-Time Windows Target” in the “New Products” section at the end of this chapter for details.

Communications Toolbox 1.4

The Communications Toolbox Version 1.4 contains the following changes and new features:

- Reorganized and streamlined Simulink block libraries
- The following new Simulink blocks:
  - Convolutional Encoder
  - Viterbi Decoder
  - Data Mapper
  - Error Rate Calculation
- Support for the new Simulink complex data type
- Other Simulink block enhancements

For details about these new features, see the *Communications Toolbox New Features Guide* for Version 1.4.

Control System Toolbox 4.2

The Control System Toolbox 4.2 contains several major new features, which are documented in the *Control System Toolbox User's Guide*. The major enhancements include:

- Convenient transfer function and zero-pole-gain model specification using rational expressions in `s` or `z` (Chapter 2)
- Ability to display multiple response types in a single LTI Viewer (Chapter 6)
- Right-click menus for customizing response plots such as those generated by `bode`, `impulse`, `nichols`, `nyquist`, `pzmap`, `sigma`, and `step` (Chapters 5 and 6)
- Frequency Response Data (FRD) object (Chapter 2): New LTI object that helps manipulate and analyze frequency response functions and experimental frequency response data
- LTI arrays (Chapter 4):
  - You can use LTI arrays to store a set of LTI models under one variable name.
  - You can perform operations on the entire set of models in an LTI array at once.
  - You can analyze response plots of LTI arrays using the LTI Viewer (Chapter 6).
- New LTI properties for assigning time delays to LTI models (Chapter 2):
  - `InputDelay`: for delays on the inputs of LTI models (This property replaces `Td`, the former (yet compatible) LTI property for assigning input delays.)
  - `OutputDelay`: for delays on the outputs of LTI models
  - `ioDelayMatrix`: for assigning independent delays to each transfer function I/O pair in MIMO transfer functions
- Support for assigning time delays to discrete-time models when the delays are integer multiples of the sampling period (Chapter 2)

Other enhancements to the Control System Toolbox 4.2 include:

- `InputGroup` and `OutputGroup`: two LTI properties that allow you to group a set of input or output channels in MIMO LTI models into named categories (Chapters 2 and 3)
- Name-based subsystem extraction (Chapter 3): the ability to specify subsystems of an LTI model by referring to the names you assign to any of following LTI properties of the model:
  - `InputName`
  - `OutputName`
  - `InputGroup`
  - `OutputGroup`
- Simulink LTI Viewer enhancements (Chapter 6):
  - The Simulink LTI Viewer can now linearize both continuous-time and discrete-time Simulink models.
  - You can now drop Input and Output Point blocks on vectorized signals in a Simulink model.
- An improved algorithm for computation of minimal state-space realizations using `minreal`

In addition, the MAT-file `LTIexamples.mat` contains sample LTI models of different types, as well as an LTI array. You can use these sample models to:

- Learn about the data format of the different types of models
- Learn how to use the Control System Toolbox GUIs

This MAT-file replaces `LTIView.mat` supplied with earlier versions of the Control System Toolbox.

Helper Commands, New Commands, and Changed Commands

Typing the following two *helper* commands provides you with information on LTI models or their properties:

- `ltimodels`
- `ltiprops`

The following table lists all the other new or changed functions in Version 4.2.

Function Name	Description
`chgunits`	Convert the `units` property for FRD models.
`delay2z`	Convert delays in discrete-time models or FRD models.
`frd`	Create or convert to a frequency response data (FRD) model.
`frdata`	Retrieve frequency response data from an FRD model.
`hasdelay`	Test true if LTI model has any type of delay.
`lft`	Calculate the star product (LFT interconnection).
`ndims`	Get the number of dimensions for LTI models or LTI arrays.
`reshape`	Change the shape of an LTI array
`set`	Set LTI model properties.
`size`	Display LTI model sizes and order.
`sminreal`	Calculate structured model reduction.
`stack`	Concatenate LTI models along array dimensions.
`totaldelay`	Provide the aggregate delay for an LTI model.
`zero`	Calculate zeros of an LTI model.

For the functions that have been modified, note the following:

- The syntax to get the order of an LTI model system has been changed to

  ```
  size(sys,'order')
  ```

  Type `help ss/size` for details.

- The description of the `set` command provides information on the data formats for LTI arrays of transfer function, state-space, zero-pole-gain, and frequency response data models.
- `lft` replaces `star`.
- `zero` replaces `tzero`.

Financial Toolbox 2.0

Portfolio Analysis

The portfolio analysis and optimization functions now support constraints on portfolios, compute asset allocation, and allow a more flexible specification of asset expected returns and covariances.

Release 2.0 provides additional functions that handle asset time series, including conversions between return and price series, expected return and covariance computations, and Monte Carlo simulation.

Function	Description
`corr2cov`	Convert standard deviation and correlation to covariance.
`cov2corr`	Convert covariance to standard deviation and correlation coefficient.
`ewstats`	Expected return and covariance from return time series.
`frontcon`	Efficient frontier with basic constraints.
`pcalims`	Asset allocation bounds.
`pcgcomp`	Group to group composition bounds.
`pcglims`	Asset group allocation bounds.
`pcpval`	Total value.
`portalloc`	Capital allocation.
`portcons`	Specify constraints.
`portopt`	Efficient frontier with arbitrary constraint set.
`portsim`	Random simulation of correlated asset returns.
`portstats`	Risk and expected rate of return.
`portvrisk`	Portfolio value at risk.
`ret2tick`	Price tick series from incremental returns and initial price.
`tick2ret`	Incremental return series from a tick price series.

Fixed Income Functions

Coupon functions now handle the SIA conventions for bonds with possible odd first and last coupon periods. The functions also return an expanded set of coupon parameters including lists of cash flow dates and amounts, accrued interest, and time factors.

Function	Description
accrfrac	Accrued interest coupon period fraction.
bndprice	Price of an SIA standard fixed income security.
bndyield	Yield of an SIA standard fixed income security.
cfamounts	Cash flow and time mapping for bond portfolio.
cfdates	Cash flow dates.
cftimes	Time factors corresponding to bond cash flow dates.
cpncount	Coupons payable between dates.
cpndaten	Next coupon date after date.
cpndatenq	Next quasi coupon date after date.
cpndatep	Previous coupon date before date.
cpndatepq	Previous quasi coupon date before date.
cpndaysn	Number of days between date and next coupon date.
cpndaysp	Number of days between date and previous coupon date.
cpnpersz	Size in days of period containing date.

Univariate GARCH Processes

Release 2.0 provides functions for performing univariate ARCH/GARCH analysis. Parameter estimation, volatility forecasting, and simulation are possible for a GARCH process with Gaussian residuals.

Function	Description
ugarch	GARCH parameter estimation.
ugarchllf	Log-likelihood objective function.
ugarchpred	Forecast conditional variance.
ugarchsim	Simulate GARCH process.

Pricing and Analyzing Derivatives

The Black-Derman-Toy model for valuing bond options is now included. The function builds a recombining binary tree from input rate curve, volatility curve, and credit spread.

Function	Description
`bdtbond`	Black-Derman-Toy pricing of option-embedded bonds.
`bdttrans`	Translate a tree returned by `bdtbond`.

Time Series Demonstration

The Financial Toolbox now includes a demonstration time series object. It has an example implementation of an interface to a charting and analysis package. Approximately 80 functions and overloaded methods are included.

Image Processing Toolbox 2.2

Support for 16-bit Image Data

Most of the functions in the toolbox have been rewritten to add support for processing 16-bit image data. Chapter 1, "Introduction," of the *Image Processing Toolbox User's Guide* discusses working with `uint16` images.

Data Type Conversion

The new function `im2uint16` converts `uint8` and `double` images to `uint16`. Chapter 1, "Introduction," of the *Image Processing Toolbox User's Guide* discusses using `im2uint16`. (See "Converting the Data Types of Images.")

Improved Speed

The following functions have been improved for faster performance: `bwfill`, `bwselect`, `bwlabel`, `dilate`, `erode`, `histeq`, `imresize`, `imrotate`, `ordfilt2`, `medfilt2`, and `im2uint8`.

New Border-Handling Options

New border-handling options have been added to `medfilt2` and `ordfilt2`.

Image-Related MATLAB 5.3 Changes

Several enhancements to MATLAB 5.3 have a direct impact on the feature set and usability of the Image Processing Toolbox 2.2:

- Improved support for integer types (`uint8`, `int8`, `uint16`, `int16`, `uint32`, `int32`). When working with these data types, you can use relational operators (<,>,==,~=), logical operators (&,|,~), bit functions, and the following functions: `any`, `all`, `find`, `min`, `max`, `permute`, `transpose`, `sum`, and `reshape`.
- Added support for the PNG graphics file format
- Added support for 16-bit image display
- Added support for 16-bit TIFF file I/O

Bug Fixes

Version 2.2 of the Image Processing Toolbox also incorporates several bug fixes. Type `info images` at the command prompt for a detailed list of bug fixes.

Mapping Toolbox 1.1

New External Data Interface Functions

Function	Purpose
avhrrgoode	Read AVHRR data stored in the Goode Projection
avhrrlambert	Read AVHRR data stored in the Lambert Projection
dted	Read U.S. Department of Defense Digital Terrain Elevation Data (DTED) data
egm96geoid	Read 15-minute gridded geoid heights from the EGM96 geoid model
gshhs	Read Global Self-consistent Hierarchical High-resolution Shoreline data
satbath	Read global 2-minute (4 km) topography form satellite bathymetry
usgs24kdem	Read USGS 1:24,000 (30 m) digital elevation maps
vmap0data	Extract selected data from the Vector Map Level 0 CD-ROMs

New Generalized Functions

These new functions make reading custom formats easier.

Function	Purpose
readfields	Read field or records from a fixed format file
grepfields	Identify matching records in fixed record length files
readmtx	Read a matrix stored in a file

New Projection Functions

Function	Purpose
aitoff	Create a Aitoff projection
bries	Create a Briesemeister's projection
hammer	Create a Hammer projection
ups	Create a Universal Polar Stereographic projection
utm	Create a Universal Transverse Mercator projection
vperspec	Create a Vertical Perspective Azimuthal projection

New Calculate and Plot Projection Distortion Characteristic Functions

Function	Purpose
distortcalc	Calculate distortion parameters for a map projection
mdistort	Display contours of constant distortion on a map

New Atlas Data Interface Functions

Function	Purpose
coast	Access world coastline data
usahi	Access United States vector data
usalo	Access United States vector data
worldlo	Access world vector data

New Map Creation Functions

Function	Purpose
`usamap`	Create a map of the United States
`worldmap`	Create a map of a country or a region

New Data Projection Functions

Function	Purpose
`contourfm`	Project a filled contour map
`country2mtx`	Create a matrix map for a country in the worldlo database
`makemapped`	Make an object a mapped object
`vec2mtx`	Create a regular matrix map from vector data

New or Updated Map Appearance and Interaction Functions

Function	Purpose
axesscale	Resize axes for equivalent scale
scaleruler	Add graphic scale
parallelui	Interactively modify map parallels
polcmap	Create colormaps for political maps
previewmap	View map at printed size
restack	Restack objects within the axes
rotatetext	Rotate text to the projected graticule
scatterm	(Not new) — Updated to include color scatter plots
tightmap	Remove white space around a map

New Moon Topography Datasets

The Mapping Toolbox 1.1 adds two moon topography datasets from the Clementine satellite dataset: `moontopo.mat` and `moonalb.mat`.

Updated Graphical Interface

The `maptool` command invokes a GUI for working with map data.

Excel Link 1.0.8

Support for Microsoft Excel 97

Excel Link 1.0.8 provides support for Microsoft Excel 97.

Note If you are using Microsoft Excel 5 or Excel 7, you must use MATLAB Excel Link 1.0.3, which is available to licensed Excel Link customers via the MATLAB Access Web page.

Optimization Toolbox 2.0

Large-Scale Algorithms

The focus of Version 2.0 of the Optimization Toolbox is new algorithms for solving large-scale problems, including:

- Linear programming
- Nonlinear least squares with bound constraints
- Nonlinear system of equation solving
- Unconstrained nonlinear minimization
- Nonlinear minimization with bound constraints
- Nonlinear minimization with linear equalities
- Quadratic problems with bound constraints
- Quadratic problems with linear equalities
- Linear least squares with bound constraints

The new large-scale algorithms have been incorporated into the toolbox functions. The new functionality improves the ability of the toolbox to solve large sparse problems.

Function Names and Calling Syntax

To accommodate this new functionality, many of the function names and calling sequences have changed. Some of the improvements include:

- Command line syntax has changed:
  - Equality constraints and inequality constraints are now supplied as separate input arguments.
  - Linear constraints are supplied as separate arguments from the nonlinear constraint function.
  - The gradient of the objective is computed in the same function as the objective, rather than in a separate function, in order to provide more efficient computation (because the gradient and objective often share similar computations). Similarly, the gradient of the nonlinear constraints is computed by the (now separate) nonlinear constraint function.

- The Hessian matrix is provided by the `objective` function when using the large-scale algorithms.

- Optimization parameters are now contained in a structure, with functions to create, change, and retrieve values.
- Each function returns an exit flag that denotes the termination state.

For more information on how to convert your old syntax to the new function calling sequences, see the *Optimization Toolbox User's Guide*.

Signal Processing Toolbox 4.2

Version 4.2 of the Signal Processing Toolbox delivers a number of improvements and enhancements, described below.

Also see the Signal Processing Toolbox `readme` file for a summary of the new additions. To view the `readme` file, type

```
info signal
```

New Functions

Name	Purpose
`ac2poly`	Autocorrelation sequence to prediction polynomial conversion
`ac2rc`	Autocorrelation sequence to reflection coefficients conversion
`arburg`	AR parametric modeling via Burg's method
`arcov`	AR parametric modeling via the covariance method
`armcov`	AR parametric modeling via the modified covariance method
`aryule`	AR parametric modeling via the Yule-Walker method
`buffer`	Buffer a signal vector into a matrix of data frames
`pcov`	Power Spectrum estimate via the covariance method
`pmcov`	Power Spectrum estimate via the modified covariance method
`poly2ac`	Prediction polynomial to autocorrelation sequence conversion
`pwelch`	Power Spectrum estimate via Welch's modified periodogram method

Name	Purpose
`rc2ac`	Reflection coefficients to autocorrelation sequence conversion
`rlevinson`	Reverse Levinson-Durbin recursion
`sgolay`	Design a Savitzky-Golay smoothing filter
`sgolayfilt`	Filter a signal with a Savitzky-Golay smoothing filter
`sosfilt`	Filter a signal using second-order sections (biquad)
`tf2sos`	Transfer function to second-order sections conversion

New Demos

Name	Purpose
`sgolaydemo`	Demonstrates Savitzky-Golay filtering

Enhanced Functions

firrcos

The `firrcos` function now:

- Accepts either a bandwidth or a roll-off factor
- Designs either a normal or square root raised cosine filter
- Accepts an arbitrary variable delay for the impulse response
- Accepts a window parameter for the filter design

pburg, pmtm, pmusic, pyulear

These functions have changed in a way that may affect your results.

When no sampling frequency (Fs) is specified, these functions return the PSD estimate, Pxx(ω), as a function of normalized angular frequency,

$$\omega = \frac{2\pi f}{Fs}$$

in rads/sample. If Fs is specified, the functions return the PSD estimate, Pxx(f)/Fs, as a function of linear frequency, f, in Hz. Fs defaults to 1 Hz. Note that the new functions `pcov`, `pmcov`, and `pwelch` also adhere to this specification.

poly2rc

Returns the zero-lag autocorrelation when called with the optional second input argument, the final prediction error.

rc2poly

The `rc2poly` function has changed in ways that may affect your results:

- Returns a column vector rather than a row vector
- Returns the final prediction error when called with the optional second input argument, the zero lag autocorrelation

sos2ss, sos2tf, sos2zp

The `sos2ss`, `sos2tf`, and `sos2zp` functions now accept an optional second input argument, the gain returned by the functions that convert to second-order-sections (SOS) form (`ss2sos`, `tf2sos`, and `zp2sos`).

ss2sos, zp2sos

The `ss2sos` and `zp2sos` functions have changed in ways that may affect your results. These functions provide an additional output argument corresponding to the gain of the second-order-sections structure and also accept an additional input argument that specifies the desired scaling of the structure. Scaling choices are: ∞-norm, 2-norm, and none.

detrend Now Part of MATLAB Language

The `detrend` function now ships in the `toolbox/matlab/datafun` directory as part of the standard MATLAB language.

Interactive Tool Enhancements

The following tools have changed in ways that may affect your results.

- SPTool loads a default session upon startup.
- Signal Browser offers printing with preview.
- Filter Designer provides a Pole/Zero Editor to supplement the existing design methods.
- Spectrum Viewer provides new covariance and modified covariance spectral estimation methods, as well as printing with preview. Additionally,
  - The maximum entropy method (MEM) has been removed. Use the Yule-Walker AR method instead.
  - Welch's method now uses the `pwelch` function instead of `psd`, and therefore no longer offers the scaling or detrending options. (`pwelch` internally scales the PSD magnitude by 1/Fs, and does not detrend the original signal.)
  - The Burg and Yule-Walker AR methods now scale the PSD magnitude by 1/Fs
  - The option to specify an autocorrelation matrix as input to the Yule-Walker AR method has been removed

Statistics Toolbox 2.2

The Statistics Toolbox 2.2 supports functions that enable you to perform cluster analysis on a dataset. Cluster analysis, also called segmentation analysis or taxonomy analysis, is a way to partition a set of objects into groups, or *clusters*, in such a way that the profiles of objects in the same cluster are very similar and the profiles of objects in different clusters are distinct.

Cluster analysis can be performed on many different types of datasets. For example, a dataset might contain a number of observations of subjects in a study where each observation contains a set of variables.

The new cluster analysis functions are summarized below.

Cluster Analysis Functions

Function	Description
`cluster`	Create clusters from the output of the `linkage` function
`clusterdata`	Create clusters from a dataset
`cophenet`	Check the validity of the clusters formed by the `linkage` function
`dendrogram`	Display the hierarchical cluster tree created by the `linkage` function as a dendrogram plot
`inconsistent`	Get information about the relative difference between a particular link in the cluster tree and the links immediately below it
`linkage`	Group objects in a dataset into binary clusters, based on the distance information generated by the `pdist` function. The `linkage` function links objects together using the Single linkage, Complete linkage, Average linkage, Centroid linkage, or Ward linkage algorithms.

Function	Description
`pdist`	Calculate the distance between pairs of objects in a dataset, using the Euclid, Standardized Euclid, Minkowski, Mahalanobis, or City Block metrics
`zscore`	Normalize data. Used before calculating the pair-wise distance between objects in the dataset

Symbolic Math Toolbox 2.1

The Symbolic Toolbox 2.1 has been enhanced to provide:

- More plotting capabilities
- New Maple libraries
- A graphical user interface (GUI) for Taylor series analysis

Enhanced Plotting Capabilities

The following new functions provide additional plotting capabilities.

Mathematical Expression	Type of Plot	MATLAB Command
$y = f(x)$	Planar curve	`ezplot`
$f(x,y) = 0$	Implicitly defined function	`ezplot`
$x = f(t),\ y = g(t)$	Parametric curve (2-D)	`ezplot`
$r = f(\theta)$	Polar coordinates	`ezpolar`
$x = f(t),\ y = g(t),\ z = h(t)$	Parametric curve (3-D)	`ezplot3`
$z = f(x,y)$	Surface	`ezsurf`, `ezsurfc`, `ezmesh`, `ezmeshc`
$z = f(x,y)$	Surface contours	`ezcontour`, `ezcontourf`
$x = f(s,t),\ y = g(s,t),\ z = h(s,t)$	Parametric surface	`ezsurf`, `ezsurfc`, `ezmesh`, `ezmeshc`
$x = f(s,t),\ y = g(s,t),\ z = h(s,t)$	Parametric surface contours	`ezcontour`, `ezcontourf`

New Maple Libraries

The new Maple V Release 5 libraries (also known as MathEdge 2) are incorporated into the Symbolic Math Toolbox 2.1. These libraries provide better memory management and fix many bugs in previous versions of the Maple kernel.

Taylor Series Expansion

The new `taylortool` command invokes a GUI that shows how a Taylor series converges to a given function.

DSP Blockset 3.0

Version 3.0 of the DSP Blockset is a major release, and introduces a substantial set of new features:

- All blocks now transparently handle both real and complex data.

  Dedicated complex blocks (such as Complex To Workspace and Mag/Angle Join) have been removed from the blockset.

  Some of these complex blocks have been merged with their real counterparts. For example, the Version 2.2 FFT and Complex FFT blocks are now a single block, FFT. Other complex-data blocks, including most of those in the 2.2 Complex library, are now a part of the Simulink library (usually under different names). Examples are the 2.2 Real and Imag blocks, which are now combined as the Complex to Real-Imag block.

- All blocks support frame-based processing for increased throughput rates.

  Most blocks that operate on sequential time-samples now offer a **Frame-based inputs** checkbox in the parameter dialog box. When you check the box, the block accepts *frames* of buffered time-samples rather than a scalar sequence over time. Frame-based processing can return great increases in efficiency for both simulated and compiled models.
- All blocks support the multirate sample time enhancements in Simulink 3.0.

  The **Sample time** field in the parameter dialog box has been removed from almost all blocks. Blocks now automatically detect the sample times of inputs. Source blocks (such as Signal From Workspace) still retain the **Sample time** parameter.

- Many blocks support internal *buffer reuse* for in-place algorithms and global sharing. Inplace algorithms reuse the same block of memory to store the intermediate results of a series of related operations. Global sharing allows a previously allocated block of memory that is no longer in use at a given time step to be recruited for a different operation. These buffer reuse modes help to reduce the memory footprint of both the simulation and generated code.
- Real-time audio support for the PC.

- New speech, audio, and wavelet demos have been added.
- Many additional DSP block algorithms are now inlined and optimized in code generated using Real-Time Workshop Target Language Compiler™ templates.

There are also a number of new and enhanced blocks, and new libraries. The next few pages outline the new additions, and provide pointers to the complete feature descriptions in the *DSP Blockset User's Guide*. See Chapter 1 of the *DSP Blockset User's Guide* for an overview of the blockset's contents.

Also see the DSP Blockset `readme` file for a summary of the new additions. To view the `readme` file, at the MATLAB command line type

```
info dspblks
```

Note The DSP Blockset 3.0 requires Simulink 3.0.

Running Different Blockset Versions

When you install the DSP Blockset 3.0 on your computer, Version 2.2 of the blockset is also installed.

Run Version 3.0 by typing `dsplib`. To run Version 2.2, type `dsplib 2`.

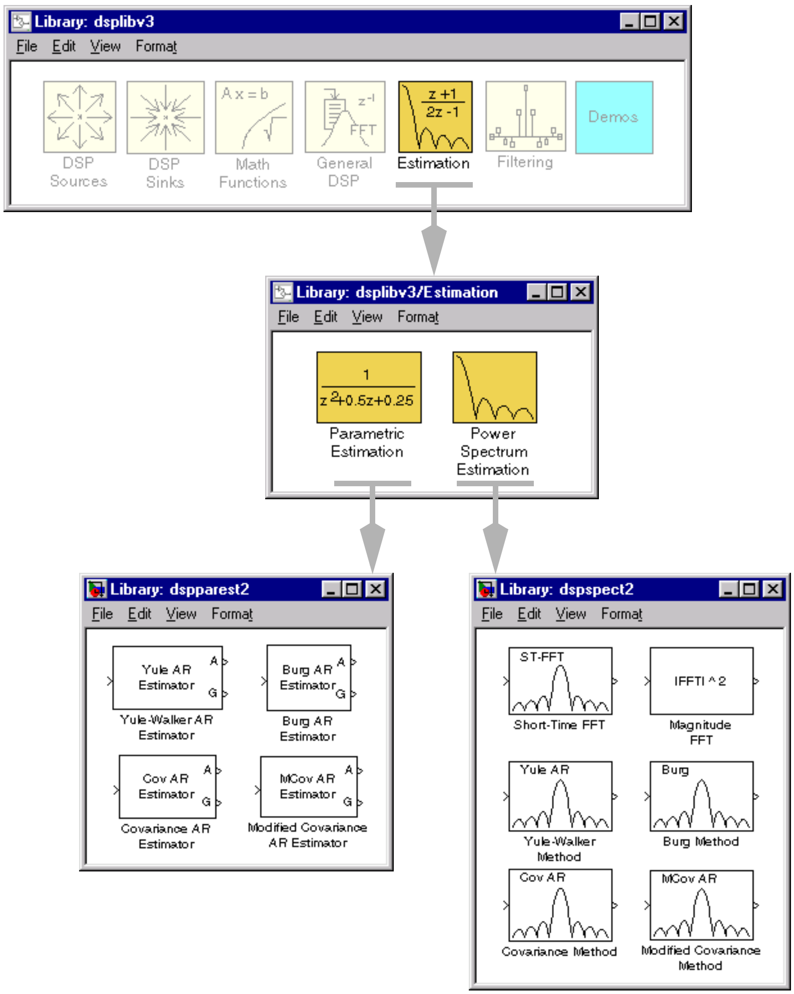
Incompatibilities Between 3.0 and 2.2

Because of the extensive changes introduced in this release to support the new Simulink complex data format, incompatibilities can arise when 3.0 blocks are used in models containing 2.2 blocks. See "Upgrading to DSP Blockset 3.0 and Communications Toolbox 1.4" in Chapter 4 for information about migrating a model to the current version.

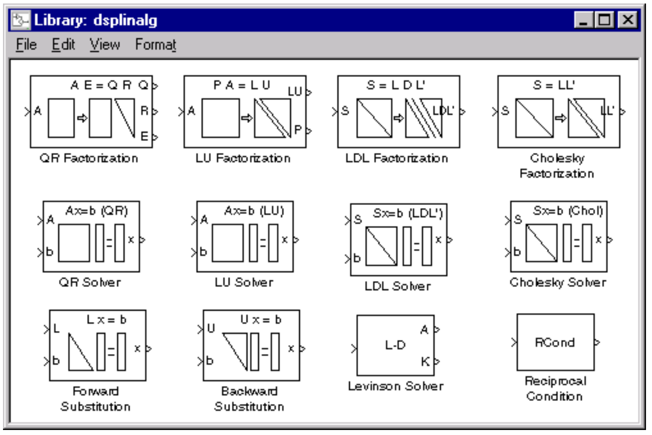
Library Structure

The library structure has undergone further refinement for Version 3.0. The major alterations are:

- The Spectrum Analysis library in Version 2.2 has been replaced by the Version 3.0 Estimation library. This library contains two additional libraries, Parametric Estimation and Power Spectrum Estimation.



- A new Linear Algebra library has been added in the Math Functions library. The library primarily offers blocks for matrix factorization and linear equation solvers.



- The Complex library has been removed from the Math Functions library as a result of the change in the complex data format. Blocks that were formerly in the Complex library have either been combined with their real-data counterparts, or relocated to other libraries. See Table 1-2.

Data Frames

Most blocks whose operation can benefit from block processing now accept data *frames*, vectors whose elements represent consecutive time samples from a single signal. Framed data is a common format in real-time systems, where the data acquisition hardware often operates most efficiently by accumulating a large number of signal samples at a high rate, and then propagating these samples to the real-time system as a block, or frame, of data. Data frames can also be constructed through the usual DSP Blockset buffering operations (using the Buffer block, for example).

See "Working with Arrays and Frames" in Chapter 3 of the *User's Guide* for a complete discussion of the frame data format, and how to use it to improve model efficiency.

Upgrading Your Models to Use Data Frames

You can realize large improvements in the efficiency of your models by using data frames whenever possible. Although throughput gains are particularly pronounced in systems where the sampled data is introduced in a framed format (such as speech and audio), non-real-time simulations also benefit as a result of the reduction in block-to-block communication overhead.

Complex Data

All blocks in the DSP Blockset are now capable of processing both real and complex data (using Simulink's new complex data type). In cases where two separate blocks were previously provided for real and complex inputs (e.g., FFT and Complex FFT), there is now a single block (FFT) that operates on both real and complex data. This enhancement greatly simplifies the contents of most libraries, in addition to allowing the removal of the Complex library from Math Functions. Blocks in the Complex library that could not be combined with a real data counterpart (e.g., Imag) are now in the Simulink Math library (usually under a different name). Table 1-2 lists the new names and locations of all former 2.2 blocks.

If any of your models use complex data, be sure to read "Why You Need to Update Your Models to Use the New Complex Data Format" in Chapter 4 before adding any Version 3.0 blocks.

Multirate Sample Time Enhancements

As a result of the multirate sample time enhancements in Simulink 3.0, all nonsource DSP blocks now inherit and propagate their sample times. This means that you do not need to track sample times manually throughout a model; when you make a change to the sample time of a source block, all other DSP blocks in the model automatically adjust to the propagated sample time.

New and Enhanced Blocks

Table 1-1 lists the new blocks in Version 3.0. Among the most significant additions are the linear algebra blocks and real-time audio blocks.

Table 1-1: New Blocks in the DSP Blockset 3.0

Block Library	Block Name	Purpose
DSP Sources	Chirp	Generate a swept-frequency cosine.
	Discrete Constant	Generate a constant.
	From Wave Device	Read audio data from a standard audio device in real-time (Windows 95/98/NT only).
	From Wave File	Read audio data from a Microsoft Wave (.wav) file (Windows 95/98/NT only).
	Triggered Signal From Workspace	Acquire and output a workspace signal when triggered.
	Sine Wave	Generate one or more sine waves.
DSP Sinks	Buffered FFT Frame Scope	Compute and display the frequency content of an input sequence.
	FFT Frame Scope	Compute and display the frequency content of a framed input.
	Frequency Frame Scope	Display frame-based data.
	Matrix Viewer	Display a matrix as an image with values mapped to colors.
	Time Frame Scope	Display frame-based data.
	To Wave Device	Send audio data to a standard audio device in real-time (Windows 95/98/NT only).
	To Wave File	Write audio data to file in the Microsoft Wave (.wav) format (Windows 95/98/NT only).
	User-defined Frame Scope	Display frame-based data.
Elementary Functions	Contiguous Copy	Recreate the input in a contiguous block of memory (for code generation).
	Convert Complex DSP to Simulink	Convert complex data from the DSP Blockset v2.2 format to the Simulink v3 format.
	Convert Complex Simulink to DSP	Convert complex data from the Simulink v3 format to the DSP Blockset v2.2 format.
	Inherit Complexity	Change the complexity of the input to match that of a reference signal.
	Variable Selector	Select a subset of elements (submatrix) in a matrix.
Matrix Functions	Create Diagonal Matrix	Create a matrix from a vector diagonal.
	Extract Diagonal	Create a vector from the elements of a matrix diagonal.
	Extract Triangular Matrix	Extract the lower or upper triangle from an input matrix.
	Matrix Product	Multiply the elements on a specified matrix row or column.
	Matrix Scaling	Scale the rows or columns of a matrix by a specified vector.
	Matrix Sum	Sum the elements on a specified matrix row or column.
	Permute Matrix	Reorder the rows or columns of a matrix.

Table 1-1: New Blocks in the DSP Blockset 3.0 (Continued)

Block Library	Block Name	Purpose
Linear Algebra	Backward Substitution	Solve the equation Ux=b for upper triangular matrix U.
	Cholesky Factorization	Factor a Hermitian positive definite matrix into triangular components.
	Cholesky Solver	Solve the equation Sx=b for Hermitian positive definite matrix S.
	Forward Substitution	Solve the equation Lx=b for lower triangular matrix U.
	LDL Factorization	Factor a Hermitian positive definite matrix into lower, upper, and diagonal components.
	LDL Solver	Solve the equation Sx=b for Hermitian positive definite matrix S.
	LU Factorization	Factor a square matrix into lower and upper triangular components.
	LU Solver	Solve the equation Ax=b for square matrix A.
	QR Factorization	Factor a rectangular matrix into unitary and upper triangular components.
	QR Solver	Find a minimum-norm-residual solution to the equation Ax=b.
	Reciprocal Condition	Compute the reciprocal condition of a square matrix in the 1-norm.
Buffers	Queue	Buffer inputs into a FIFO (first input, first output) register.
	Rebuffer	Increase or decrease the size of the input frame.
	Stack	Buffer inputs into a LIFO (last input, first output) register.
Switches and Counters	Counter	Count up or down through a specified range of numbers.
	Edge Detector	Detect transition of input from zero to non-zero value.
	Event-Count Comparator	Detect threshold crossing of accumulated non-zero events.
	Multiphase Clock	Generate multiple binary clock signals.
Parametric Estimation	Burg AR Estimator	Compute an estimate of AR model parameters using the Burg method.
	Covariance AR Estimator	Compute an estimate of AR model parameters using the covariance method.
	Modified Covariance AR Estimator	Compute an estimate of AR model parameters using the modified covariance method.
	Yule-Walker AR Estimator	Compute an estimate of AR model parameters using the Yule-Walker method.

Table 1-1: New Blocks in the DSP Blockset 3.0 (Continued)

Block Library	Block Name	Purpose
Power Spectrum Estimation	Covariance Method	Compute a parametric spectral estimate using the covariance method.
	Magnitude FFT	Compute a nonparametric estimate of the spectrum using the periodogram method.
	Modified Covariance Method	Compute a parametric spectral estimate using the modified covariance method.
	Short-Time FFT	Compute a nonparametric estimate of the spectrum using the modified, averaged periodogram method.
Filter Realizations	Biquadratic Filter	Apply a cascade of biquadratic (second-order-section) filters to the input.
	Direct-Form II Transpose Filter	Apply an IIR filter to the input.
	Time-Varying Direct-Form II Transpose Filter	Apply a variable IIR filter to the input.
	Time-Varying Lattice Filter	Apply a variable lattice filter to the input.
Multirate Filters	Dyadic Analysis Filter Bank	Decompose a signal using a dyadic multirate filter bank.
	Dyadic Synthesis Filter Bank	Reconstruct a signal using a dyadic multirate filter bank.

In addition to the new blocks above, most Version 2.2 blocks have received enhancements for Version 3.0, and many have changed names (primarily as a result of the new complex data format). Table 1-2 below lists *all* of the 2.2 blocks alphabetically, and shows the corresponding 3.0 block and library location.

Note The shaded rows indicate blocks that have either changed names or library locations.

Table 1-2: Enhanced Blocks in the DSP Blockset 3.0

2.2 Block Name	3.0 Block Name	Library Location
Analog Filter Design	same	same
Analytic Signal	same	same
Angle	Complex to Magnitude-Angle	Simulink
Autocorrelation	same	same
Buffer	same	same
Buffered FFT Scope	Buffered FFT Frame Scope	same
Burg Method	same	Power Spectrum Estimation
Commutator	same	same
Complex Autocorrelation	Autocorrelation	same

Table 1-2: Enhanced Blocks in the DSP Blockset 3.0 (Continued)

2.2 Block Name	3.0 Block Name	Library Location
Complex Buffer	Buffer	same
Complex Buffered FFT Scope	Buffered FFT Frame Scope	same
Complex Cepstrum	same	same
Complex Constant	Constant	Simulink
Complex Delay	Integer Delay	same
Complex Demux	Demux	Simulink
Complex Diagonal Matrix	Constant Diagonal Matrix	same
Complex Dot Product	Dot Product	Simulink
Complex Exponential	same	Elementary Functions
Complex FFT Scope	FFT Frame Scope	same
Complex Flip	Flip	same
Complex From Workspace	Signal From Workspace	same
Complex Gain	Gain	Simulink
Complex Kalman Adaptive Filter	Kalman Adaptive Filter	same
Complex Levinson-Durbin	Levinson Solver	same
Complex LMS Adaptive Filter	LMS Adaptive Filter	same
Complex LPC	LPC	same
Complex Matrix Constant	Matrix Constant	same
Complex Matrix From Workspace	Matrix From Workspace	same
Complex Matrix Multiplication	Matrix Multiplication	same
Complex Matrix To Workspace	Matrix To Workspace	same
Complex Multiply	Product	Simulink
Complex Mux	Mux	Simulink
Complex Normalization	Normalization	same
Complex Partial Unbuffer	Partial Unbuffer	same
Complex Reciprocal	Math Function	Simulink
Complex RLS Adaptive Filter	RLS Adaptive Filter	same
Complex Selector	Selector	Simulink
Complex Submatrix	Submatrix	same
Complex Sum	Sum	Simulink
Complex To Workspace	To Workspace	Simulink
Complex Transpose	Transpose	same
Complex Unbuffer	Unbuffer	same
Complex Unit Delay	Integer Delay	same
Complex Width	Width	Simulink
Complex Zero Pad	Zero Pad	same
Conjugate	Math Function	Simulink

Table 1-2: Enhanced Blocks in the DSP Blockset 3.0 (Continued)

2.2 Block Name	3.0 Block Name	Library Location
Constant Exponent	Math Function	Simulink
Convolution	same	same
Convolution C-C	Convolution	same
Convolution C-R	Convolution	same
Correlation	same	same
Correlation C-C	Correlation	same
Correlation C-R	Correlation	same
Cumulative Sum	same	same
dB	same	Elementary Functions
dB Gain	same	Elementary Functions
DCT	same	same
Delay	Integer Delay	same
Detrend	same	same
Diagonal Matrix	Constant Diagonal Matrix	same
Difference	same	same
Digital FIR Filter Design	same	same
Digital IIR Filter Design	same	same
Distributor	same	same
Dot Product	same	Simulink
Downsample	same	same
FFT	same	same
FFT Scope	FFT Frame Scope	same
Filter	Discrete Filter	Simulink
Filter Realization Wizard	same	same
FIR Decimation	same	same
FIR Interpolation	same	same
FIR Rate Conversion	same	same
FIR Rate Conversion (Frame)	FIR Rate Conversion	same
Fixed Truncation	Rounding Function	Simulink
Flip	same	same
Frequency Vector Scope	Frequency Frame Scope	same
Hermitian Transpose	Transpose	same
Histogram	same	same
IDCT	same	same
IFFT	same	same
Imag	Complex to Real-Imag	Simulink
Inverse-FFT FIR Filter Design	obsolete	same

Table 1-2: Enhanced Blocks in the DSP Blockset 3.0 (Continued)

2.2 Block Name	3.0 Block Name	Library Location
Join	Real-Imag to Complex	Simulink
Kalman Adaptive Filter	same	same
Least Squares FIR Filter Design	same	same
Levinson-Durbin	Levinson Solver	Linear Algebra
LMS Adaptive Filter	same	same
LPC	same	same
Mag/Angle Join	Magnitude-Angle to Complex	Simulink
Mag/Angle Split	Complex to Magnitude-Angle	Simulink
Magnitude	Abs	Simulink
Magnitude Squared	Math Function	Simulink
Math Function	same	Simulink
Matrix Constant	same	same
Matrix From Workspace	same	same
Matrix Multiplication	same	same
Matrix To Workspace	same	same
Maximum	same	same
Mean	same	same
Median	same	same
Minimum	same	same
Multichannel IIR Filter	Direct-Form II Transpose Filter	same
Multichannel IIR Filter (Frame)	Direct-Form II Transpose Filter	same
N-Sample Enable	same	same
N-Sample Enable w/Reset	N-Sample Enable	same
N-Sample Switch	same	same
Normalization	same	same
Overlap-Add FFT Filter	same	same
Overlap-Save FFT Filter	same	same
Partial Unbuffer	same	same
Periodogram	Short-Time FFT	Power Spectrum Estimation
Quantizer	same	Simulink
Real	Complex to Real-Imag	Simulink
Real Cepstrum	same	same
Real DCT	DCT	same
Real FFT	FFT	same
Real IDCT	IDCT	same
Real IFFT	IFFT	same
Real To Complex	Real-Imag to Complex	Simulink

Table 1-2: Enhanced Blocks in the DSP Blockset 3.0 (Continued)

2.2 Block Name	3.0 Block Name	Library Location
Remez FIR Filter Design	same	same
Repeat	same	same
Reshape	same	same
RLS Adaptive Filter	same	same
RMS	same	same
Rounding Function	same	Simulink
Running Histogram	Histogram	same
Running Maximum	Maximum	same
Running Mean	Mean	same
Running Minimum	Minimum	same
Running RMS	RMS	same
Running Standard Deviation	Standard Deviation	same
Running Variance	Variance	same
Sample and Hold	same	same
Shift Register	same	same
Sign	same	Simulink
Signal From Workspace	same	same
Sort	same	same
Split	Complex to Real-Imag	Simulink
Standard Deviation	same	same
Submatrix	same	same
Time Varying FIR Filter	Time-Varying Direct-Form II Transpose Filter	same
Time Varying IIR Filter	Time-Varying Direct-Form II Transpose Filter	same
Time Vector Scope	Time Frame Scope	same
To Workspace	Signal To Workspace	same
Toeplitz	same	same
Transpose	same	same
Triggered Complex Matrix To Workspace	Triggered Matrix To Workspace	same
Triggered Complex To Workspace	Triggered Signal To Workspace	same
Triggered Matrix To Workspace	same	same
Triggered Shift Register	same	same
Triggered To Workspace	Triggered Signal To Workspace	same
Trigonometric Function	same	Simulink
Unbuffer	same	same
Unit Delay	Integer Delay	same
Unwrap	same	same

Table 1-2: Enhanced Blocks in the DSP Blockset 3.0 (Continued)

2.2 Block Name	3.0 Block Name	Library Location
Upsample	same	same
Variable Fractional Delay	same	same
Variable Integer Delay	same	same
Variance	same	same
Width	same	Simulink
Window Function	same	same
Yule-Walker AR	Yule-Walker Method	Power Spectrum Estimation
Yule-Walker IIR Filter Design	same	same
Zero Pad	same	same

Fixed-Point Blockset 2.0

Release 11 contains two versions of the Fixed-Point Blockset: Version 1.2, which was included with Release 10 (MATLAB 5.2), and Version 2.0. The 2.0 blockset is located in `/toolbox/fixpoint` and the 1.2 blockset is located in `/toolbox/fixpoint/obsolete`.

The Fixed-Point Blockset 2.0 features are discussed below.

Fixed-Point Blocks

The Fixed-Point Blockset 2.0 includes a number of building blocks to assist you in designing and simulating dynamic systems using fixed-point arithmetic. The fixed-point blocks are grouped together as shown below.

Arithmetic Blocks

Block Name	Description
FixPt Constant	Generate a constant value
FixPt Gain	Multiply the input by a constant
FixPt Matrix Gain	Multiply the input by a constant matrix
FixPt Product	Multiply or divide inputs
FixPt Sum	Add or subtract inputs

Conversion Blocks

Block Name	Description
FixPt Conversion	Convert from one Fixed-Point Blockset data type to another
FixPt Conversion Inherited	Convert input two to the data type of input one

Block Name	Description
FixPt Gateway In	Convert a Simulink data type to a Fixed-Point Blockset data type
FixPt Gateway Out	Convert a Fixed-Point Blockset data type to a Simulink data type

Look-Up Table Blocks

Block Name	Description
FixPt Look-Up Table	Approximate a one-dimensional function using a selected look-up method
FixPt Look-Up Table (2D)	Approximate a two-dimensional function using a selected look-up method

Logical and Comparison Blocks

Block Name	Description
FixPt Logical Operator	Perform the specified logical operation on the inputs
FixPt Relational Operator	Perform the specified relational operation on the inputs
FixPt Relay	Switch output between two constants
FixPt Saturation	Bound the range of the input
FixPt Switch	Switch output between input one or input three based on the value of input two

Discrete-Time Blocks

Block Name	Description
FixPt FIR	Implement a fixed-point finite impulse response (FIR) filter
FixPt Unit Delay	Delay a signal one sample period
FixPt Zero-Order Hold	Implement a zero-order hold of one sample period

Filters and Systems

The Fixed-Point Blockset 2.0 provides several useful fixed-point filter and system realizations. These realizations are intended to be used as design templates so you can easily see how to build filters and systems suited to your particular needs. The filters and systems are described below.

Filter or System Name	Description
FixPt State-Space Realization	Implement a fixed-point realization of a state-space system
FixPt Integrator: Trapezoidal	Implement a fixed-point realization of an integrator based on trapezoidal numerical integration
FixPt Integrator: Backward	Implement a fixed-point realization of an integrator based on backward numerical integration
FixPt Integrator: Forward	Implement a fixed-point realization of an integrator based on forward numerical integration
FixPt Filtered Derivative	Implement a fixed-point realization of a filtered derivative

Filter or System Name	Description
FixPt Derivative	Implement a fixed-point realization of a derivative
FixPt Lead or Lag Filter	Implement a fixed-point realization of a lead filter or lag filter

Data Types

The Fixed-Point Blockset 2.0 supports several fixed-point and floating-point data types, which are collectively referred to as the "Fixed-Point Blockset data types." The supported data types and related features are described below.

Fixed-Point Data Types

- Integer, fractional, and generalized fixed-point data types are supported.
- Unsigned and two's complement formats are supported.
- The fixed-point word size can range from 1 to 128 bits.
- The radix (binary) point is not required to be contiguous with the fixed-point word.

Floating-Point Data Types

- IEEE-style singles and doubles are supported.
- A nonstandard IEEE-style data type is supported. For this data type, the fraction (mantissa) can range from 1 to 52 bits and the exponent can range from 1 to 11 bits.

The label "Fixed-Point Blockset data types" indicates that data types supported by this blockset are unique to it, and not directly compatible with Simulink. This means that a double generated by Simulink cannot be passed directly into a Fixed-Point Blockset block, and a double generated by the Fixed-Point Blockset cannot be passed directly into a Simulink block. Instead, the FixPt Gateway In and FixPt Gateway Out blocks must be used as interfaces between the Fixed-Point Blockset and Simulink.

Scaling

The Fixed-Point Blockset 2.0 supports two general scaling modes: radix point-only scaling and slope/bias scaling. Additionally, some blocks support scaling modes that maximize the precision for constant vectors or matrices. These scaling modes are described below.

General Scaling Modes

Fixed-point numbers can be scaled in these ways:

- **Radix Point-Only**

  This is "powers-of-two" scaling since it only involves moving the radix point. Radix point-only scaling does not require the radix point to be contiguous with the data word. The advantage of this scaling mode is the number of processor arithmetic operations are minimized.

- **Slope/Bias**

  With this scaling mode, you can provide a slope and a bias. The advantage of slope/bias scaling is that it typically provides more efficient use of a finite number of bits.

Constant Scaling for Best Precision

In addition to the general scaling modes described above, the Fixed-Point Blockset provides you with block-specific scaling modes for constant vectors and constant matrices. These scaling modes are based on radix point-only scaling and are designed to maximize precision.

- **Constant Vector Scaling**

  With this mode, you have the option of scaling a constant vector such that the precision is maximized for each element, or a common radix point can be found based on the maximum precision for the largest value of the vector.

- **Constant Matrix Scaling**

  With this mode, you have the option of scaling a constant matrix such that the precision is maximized for each element, or a common radix point can be found based on the maximum precision for the largest value of each row, each column, or the whole matrix.

The advantage of finding a common radix point is increased simulation speed, while the disadvantage is reduced precision.

Automatic Scaling Tool

A script is provided that automatically changes the scaling for each block that has generalized fixed-point output and does not have its scaling locked. The script uses the maximum and minimum values logged during the last simulation run. The scaling is changed such that the simulation range is covered and the precision is maximized.

As an alternative to (and extension of) the automatic scaling script, an automatic scaling GUI is provided. This interface allows you to easily control the parameters associated with automatic scaling and display the simulation results for a given model. With the automatic scaling GUI, you can:

- Turn on or turn off logging for all blocks
- Override the output data type with doubles for all blocks
- Invoke the automatic scaling script
- Run the simulation
- Display the scaling results for each block that had its scaling changed

Locking the Output Scaling

If the output data type is a generalized fixed-point number, then you have the option of locking its scaling. When locked, the automatic scaling tool will not change the output scaling. Otherwise, the automatic scaling tool is free to adjust the scaling.

Rounding

Fixed-point numbers can be rounded in these ways:

- **Toward Zero**

  This mode rounds toward zero and is equivalent to MATLAB's `fix` command.
- **Toward Nearest**

  This mode rounds toward the nearest representable number, with the exact midpoint rounded toward positive infinity. Rounding toward nearest is equivalent to the MATLAB `round` command.
- **Toward Ceiling**

  This mode rounds toward positive infinity and is equivalent to MATLAB's `ceil` command.

- **Toward Floor**

  This mode rounds toward negative infinity and is equivalent to MATLAB's `floor` command.

Overflow Handling

Operations on fixed-point numbers that produce an overflow condition can be dealt with in these ways:

- **Saturate**

  Overflows are set to either the maximum or minimum value represented by the word.

- **Wrap**

  Overflows can be set to any value represented by the word.

Overriding with Doubles

The fixed-point data type can be overridden with doubles either globally or for individual blocks. This feature is useful when debugging a simulation.

Specialized Storage Capabilities

The maximum and minimum values encountered during a simulation can be logged to the MATLAB workspace. These values can then be accessed by the automatic scaling tool.

Standardization with Simulink

The fixed-point blocks are feature-compatible with standard Simulink blocks. Standardization with Simulink provides these expanded capabilities:

- Vectorization of inputs and outputs
- Variable number of input ports on appropriate blocks such as FixPt Sum
- More powerful blocks that combine and expand the features of the basic blocks. For example, scalar addition and subtraction are combined in the vectorized FixPt Sum block.

Enhanced Model Construction

The Fixed-Point Blockset 2.0 makes it easier to construct models of systems due to these main blockset features:

- Expanded features of old blocks such as vectorization
- Inclusion of two new blocks — FixPt Matrix Gain and FixPt FIR

Updating Obsolete Fixed-Point Blocks

Obsolete fixed-point blocks from previous Fixed-Point Blockset releases can be updated to current fixed-point blocks using the `fpupdate` command.

Code Generation

With the Real-Time Workshop, you can generate C code for execution on a fixed-point embedded processor. The generated code uses only integer types and automatically includes all operations, such as shifts, needed to account for differences in fixed-point locations. The code is structured so that key operations can be readily replaced by optimized target-specific libraries that you supply. You can also use the Target Language Compiler to customize the generated code.

All fixed-point blocks support code generation, but not every simulation feature is supported. For example, 13-bit numbers can be used for simulation but not for code generation.

Demos

The Fixed-Point Blockset 2.0 provides several demos that illustrate the main product features. You can access these demos via the fixed-point library's Demos block.

Online Help

Each Fixed-Point Blockset block, system, and filter has online HTML-based help. The help is accessed through the dialog box **Help** button.

Power System Blockset 1.1

DC Machine Block Added

A new block, DC Machine, has been added to the Extra Library of `powerlib`. This block implements a separately exited DC machine. Access is provided to the field connections so that the machine can be used as a shunt-connected or a series-connected DC machine. The model is built with both Simulink and Power System Blockset blocks and provides a good example of building a user-defined block.

New Products

New MATLAB and Simulink Report Generators

The Report Generator is a software package that can take any information from your MATLAB workspace and export it to a document in the form of a report. The reports you create with the Report Generator can include figures, data, variables, and functions from your models or M-files, as well as snapshots of all system graphics or figures.

Two Report Generator Products

There are two Report Generator products: the MATLAB Report Generator and the Simulink Report Generator. If you want to create reports for MATLAB M-files, you need the MATLAB Report Generator. If you want to create reports for Simulink or Stateflow models, you need both the MATLAB Report Generator and the Simulink Report Generator, which is built on top of the MATLAB Report Generator.

Both the MATLAB Report Generator and the Simulink Report Generator are documented in the *Report Generator User's Guide*.

Multiple Report Formats

One of the key features of the Report Generator is that you can create reports in multiple documentation formats, such as:

- RTF
- HTML
- XML
- SGML

Creating Reports with the Report Generator

A report is a formatted document that contains the information specified by a setup file. A setup file specifies which components will be in the report, component attributes, and component relationships.

A component is a self-contained, modular element that controls the report generation process and inserts elements into a report. Components control such aspects of your report as formatting, how Handle Graphics objects are handled, the logical flow for processing the report, etc. You can use the components provided with the Report Generator, or you can create your own components with the Component Creation Wizard.

You can create reports using the setup files that are provided with the Report Generator, or you can create customized reports with the Setup File Editor. The Setup File Editor is the primary graphical user interface (GUI) for the Report Generator; you can view, modify, or create setup files with it.

Real-Time Windows Target

The Real-Time Windows Target allows you to run C code generated by Real-Time Workshop on a PC in real time. In this environment, your PC is the host for MATLAB, Simulink, and the Real-Time Workshop. Once C code is generated and compiled, your same PC, running Microsoft Windows 95 or Windows 98 is then used as a target for running the generated code. Typical applications for the Real-Time Windows Target include real-time control, signal processing, and hardware-in-the-loop simulation.

Note The Real-Time Windows Target requires Real-Time Workshop 3.0 and the Watcom 11.0 C/C++ compiler.

Features

The Real-Time Windows Target has many useful features:

- The generated code runs fast (in real time, at ring zero) under Microsoft Windows 95 or Windows 98 operating systems using standard yet cost-effective I/O boards.
- When running your models in real time, Real-Time Windows Target captures a sample of data from one or more input channels, uses the data as inputs to your block diagram model, immediately processes the data, and sends it back to the outside world via an output channel on your I/O board.
- The Real-Time Windows Target uses the unique real-time capabilities of a special kernel for each of the supported Windows operating systems, combined with the speed of compiled C code.
- The Real-Time Windows Target provides a custom Simulink block library and more than 60 ready-to-use hardware I/O drivers.

- Signals may be captured and graphed in Simulink Scope blocks by using Simulink's external mode, which enables you to observe the behavior of your real-time system.
- Simulink's external mode also allows you to alter parameters on-the-fly by simply editing the block diagram while running Simulink in external mode. New parameter values are automatically transferred to the compiled version of your block diagram during real-time execution.

Supported Boards

The Real-Time Windows Target has support for over 60 boards from these manufacturers:

- Advantech
- Analog Devices
- Axiom
- Computer Boards
- Data Translation
- Humusoft
- Keithley-Metrabyte
- National Instruments
- Scientific Solutions

Refer to the *Real-Time Windows Target User's Guide* for more specific information about which boards are supported.

Database Toolbox

Note The Database Toolbox was made available via FTP prior to Release 11. However, the Database Toolbox is appearing on a MATLAB CD-ROM for the first time with Release 11.

The Database Toolbox enables you to move data (both importing and exporting) between MATLAB and popular relational databases. With the Database Toolbox, you can bring data from an existing database into MATLAB, use any of MATLAB's computational and analytic tools, and store the results back in the database or in another database. The Database Toolbox imports and exports data directly to and from databases (without your needing to use intermediary files).

For example, a financial analyst working on a mutual fund could import a company's financial data into MATLAB, run selected analyses, and store the results for future tracking. The analyst could then export the saved results to a database.

The Database Toolbox connects MATLAB to a database using MATLAB commands. Data is retrieved from the database as a string, parsed into the correct data types, and stored in a MATLAB cell array. At that point, you can use MATLAB's extensive set of tools to work with the data.

The Database Toolbox has the following features:

- Data types are automatically preserved in MATLAB – No data massaging or manipulation is required. The data is stored in MATLAB cell arrays, which support mixed data types.
- Different databases can be used in a single session – For example, you can import data from one database, perform calculations, and export the modified or unmodified data to another database. Multiple databases can be open during a session.
- You can dynamically import data from within MATLAB – Modify your SQL queries with MATLAB statements to retrieve the data you need.
- Single environment promotes for faster data analysis – Access both database data and MATLAB functions at the MATLAB command prompt.
- Database connections remain open until explicitly closed – Once the connection to a database has been established, it remains open during the entire MATLAB session until you explicitly close it. This improves database access and reduces the number of commands necessary to import/export data.
- Multiple cursors are supported for a single database connection – Once a connection has been established with a database, the connection can support the use of multiple cursors. You can execute several queries on the same connection.
- Retrieval of large data sets or partial data sets is supported – You can retrieve large data sets from a database in a single fetch or in discrete amounts using multiple fetches.

MATLAB Web Server

Note The MATLAB Web Server was made available via FTP prior to Release 11. However, the MATLAB Server is appearing on a MATLAB CD-ROM for the first time with Release 11.

The MATLAB Web Server enables you to create MATLAB applications that use the capabilities of the World Wide Web to send data to MATLAB for computation and to display the results in a Web browser.

In the simplest configuration, a Web browser runs on your client workstation while MATLAB, the MATLAB Web Server, and the Web server daemon run on another machine. In a more complex network, the Web server daemon can run on a machine apart from the others.

The MATLAB Web Server depends upon TCP/IP networking for transmission of data between the client system and MATLAB.

MATLAB Web Server applications are a combination of M-files, HTML, and graphics. Knowledge of MATLAB programming and basic HTML are the only requirements.

The process of creating a MATLAB Web Server application involves the creation of:

- An HTML input document for data submission to MATLAB
- An HTML output document for display of MATLAB's computations
- A MATLAB M-file to process input data and compute results
- A test file to validate code before distributing the application over the Web

The MATLAB Web Server is packaged with templates to simplify the process described above. Each template provides actual code that you need to incorporate into your application plus instructions on how to modify the template where necessary.

2

Release 10 (MATLAB 5.2) Enhancements

What Was New in Release 10 (MATLAB 5.2)?

Note All the features introduced in Release 10 are also in Release 11.

Release 11 completed bringing the whole MATLAB product family up to a MATLAB 5 level: the MATLAB Compiler and MATLAB C and C++ Math Libraries now work with MATLAB 5 and its associated products.

MATLAB 5.2 also added many important application development and visualization features.

In addition, other licensed products were updated with the release of MATLAB 5.2:

- Simulink 2.2 added several enhancements, including new user interface features and additional simulation features.
- Real-Time Workshop 2.2 added important enhancements.
- New versions of most toolboxes were made available with Release 10 (see the "Toolboxes and Blocksets" on page 2-36 for a complete list).
- The Power System Blockset was introduced with Release 10.

Enhancements to MATLAB

The language and development environment enhancements introduced with MATLAB 5.2 include:

- New MATLAB language functions, implementing features such as `try/catch` error handling, additional ODE functions, and M-file locking.
- Expanded application development tools for the Microsoft Windows 95 and NT and UNIX platforms.
- Improved Help Desk, with a much faster search engine that supports full-text searches, and easier HTML reference page navigation.
- Support for two new ActiveX technologies: ActiveX control containment and ActiveX Automation client capabilities, so that now MATLAB can both control and be controlled by other ActiveX components.
- Support for HDF (Hierarchical Data Format) files.

MATLAB 5.2 also added the following visualization and GUI development enhancements:

- Support for OpenGL rendering to improve performance dramatically for many visualization applications.
- Enhanced visualization features, including enhanced camera control, simplified placement of Light objects, tighter and more consistent control of graphics object hiding.
- A print frame editor that enables you to create custom layouts for printing Simulink model diagrams.
- Additional GUI development features, including the ability to define the location and size of user interface objects in units that are based on the size of the default system font, and to add tooltips, toggle buttons, context menus, etc.

Upgrades to Simulink, Real-Time Workshop, Toolboxes, and Blocksets

Simulink 2.2 introduced several enhancements, including Level 2 S-function support, new user interface features for the PC, additional simulation features, some new blocks and commands, and the ability to add print frames (header and footer annotation) to printouts of Simulink models.

Real-Time Workshop 2.2 utilized the Simulink Level-2 S-function feature to support Interrupt Service Routines (ISRs) for VxWorks, customized ISRs for your target system, multiple input/multiple output S-functions, and parameter checking while running.

Almost all toolboxes and blocksets were updated for Release 10. Toolboxes and blocksets with especially significant enhancements for Release 10 included:

- Communications Toolbox 1.3
- Control System Toolbox 4.1
- DSP Blockset 2.2
- Financial Toolbox 1.1
- Fuzzy Logic Toolbox 2.0
- Image Processing Toolbox 2.1
- Neural Network Toolbox 3.0

- Signal Processing Toolbox 4.1
- Spline Toolbox 2.0

New Power System Blockset

The Power System Blockset 1.0 was introduced with Release 10. This new blockset is described in more detail later in this chapter.

MATLAB Language Enhancements

Links to Command Descriptions Clicking on the command name in the following tables displays the documentation for that command. Use your browser's **Back** button to return to this document.

Support for try/catch

MATLAB 5.2 added functions to support `try/catch` error handling.

Function	Description
`catch`	Begin catch block.
`try`	Begin try block.

Warning Messages

The new `lastwarn` function, depending how it is called, returns either a string containing the last warning message issued by MATLAB or an empty string matrix until the next warning is encountered, or sets the last warning message to a specified string.

Setting the Recursion Limit

You can now set the limit for recursion so that you will receive an error instead of being forced out of MATLAB when the recursion limit is reached. The default recursion limit is `500`. To change the recursion limit, change the following line

```
set(0, 'recursionlimit', limitnumber)
```

in your `matlabrc.m` file in the `toolbox/local` directory.

New Mathematical Functions

MATLAB 5.2 provided these new mathematical functions.

Function	Description
`cholinc`	Sparse Incomplete Cholesky and Cholesky-Infinity factorizations.
`cholupdate`	Rank 1 update to Cholesky factorization.
`ifftshift`	Inverse fast Fourier transform shift.
`ode23t`	Solve moderately stiff problems for a solution without numerical damping.
`ode23tb`	Solve stiff systems using crude error tolerances. May also be used if there is a mass matrix.
`qrupdate`	Rank 1 update to QR factorization.

New String Comparison Functions

MATLAB 5.2 provided two additional string comparison functions.

Function	Description
`strcmpi`	Compare strings ignoring case.
`strncmpi`	Compare first `n` characters of strings ignoring case.

M-File Locking

You can now lock (and unlock) an M-file so that `clear` does not clear that M-file from memory.

Function	Description
`mislocked`	True if M-file cannot be cleared.
`mlock`	Prevent M-file clearing.
`munlock`	Allow M-file clearing.

Persistent Variables

A variable may be defined as persistent (with the keyword `persistent`) so that it does not change value from one call to another. Persistent variables may be used within a function only. Persistent variables remain in memory until the M-file is cleared or changed. `persistent` is exactly like `global`, except that the variable name is not in the global workspace, and the value is reset if the M-file is changed or cleared.

Three MATLAB functions support the use of persistent variables (see "M-File Locking" above):

- `mislocked`
- `mlock`
- `munlock`

File and Directory Handling

You can copy a file and make a directory from within MATLAB.

Function	Description
`copyfile`	Copy file.
`mkdir`	Make directory.

Enhancement to load

MATLAB 5.2 added a new option to the `load` function.

```
S = load(...)
```

returns the contents of a MAT-file as a structure instead of directly loading the file into the workspace. The field names in `S` match the names of the variables that were retrieved. When the file is ASCII, `S` will be a double precision array.

Cell Array of Strings

You can now use a cell array of strings with the following functions:

- `intersect`
- `ismember`
- `lower`
- `setdiff`
- `setxor`
- `sort`
- `union`
- `unique`
- `upper`

Enhancement to strjust

The `strjust` function now does right, left, and center justification.

Change in clc and home Behavior

The `clc` and `home` commands now both clear the command window. After issuing either of these commands, it is no longer possible to scroll back to see previous contents of the command window.

Additional Functions Changed in MATLAB 5.2

In addition to the above functions, the MATLAB 5.2 version of the following functions changed in minor ways, generally to reflect the addition of the new functions described above (e.g., `clear` does not clear if `mlock` is called first).

- `clear`
- `end`
- `horzcat`
- `lasterr`
- `paren`
- `spline`
- `strcmp`
- `strncmp`

Development Environment Tools Enhancements

MATLAB 5.2 provided enhanced environment tools for the PC (Microsoft Windows 95 and NT) platform and introduced environment tools for the UNIX platform. These enhancements are described in detail in Chapter 2 of the online (PDF) version of *Using MATLAB*.

Changes to the MATLAB Editor/Debugger

With MATLAB 5.2, the Editor/Debugger provided a new **Tools** menu. Some of the options that were under the **View** menu in previous releases on the PC are now found under the **Tools** menu. MATLAB 5.2 provided a tabbed dialog that allows you to set **General** and **Editor** options. You do so from the **Tools** menu, by selecting **Options**.

You can now use MATLAB to add your own commands to the Editor, by using the **Customize** option that appears as a submenu of the **Tools** menu. Commands that you add will also work with the Path Browser and Array Editor although the results may differ. Chapter 2 of *Using MATLAB* provides a table that explains these differences.

You can set up the Editor so that the values of MATLAB variables are expanded and displayed in the Editor window as the cursor *hovers* over a variable. To do so, under **General** options, check **Show Data Tips**.

Also under **General** options, if you check **Show worksheet style tabs**, the main Editor window displays a tab at the bottom for each open file. This allows quick navigation among all open files.

You can also control the Editor's font, style, and size. In previous releases of MATLAB, font control was available only for the command window. In MATLAB 5.2, you could select **Font** from the **Tools** menu to control Editor fonts.

Array Editor Added

MATLAB 5.2 provided an Array Editor. This tool allows you to view and edit two-dimensional numeric arrays.

New Tools for UNIX Environments

Several tools that were available on PCs in earlier versions of MATLAB were made available in MATLAB 5.2 for some UNIX machines (Sol2, Sun4, SGI, and HP 700):

- Built-in MATLAB Editor/Debugger
- Workspace Browser
- Path Browser

SGI64 Fully Supported

MATLAB 5.2 provided full support for the SGI64 platform. The SGI64 platform was supported only as a Beta product in previous MATLAB 5 versions.

Note The Symbolic Math Toolbox 2.0.1 and Extended Symbolic Math Toolbox 2.0.1 are not available for the SGI64 platform; however, the SGI image can be used on the SGI64 platform.

Online Documentation Enhancements

Full-Text Search Facility

The 5.2 Help Desk added a full-text search facility for the HTML online documentation. You can access the full-text search facility from the top page of the Help Desk or from the "Search" link on reference pages.

Reference Page Navigation

The 5.2 HTML reference pages introduced additional navigational aids. The "Examples" and "See Also" links at the top of the first reference page for a function allow you to jump directly to the examples or to links to associated functions.

Also at the top of the reference pages is a "Go to function" edit box. Enter the name of the function and press the **Enter** key to see the reference page for that function.

The doc Command

The `doc` command now accesses the HTML reference documentation for all MathWorks products for which HTML reference documentation has been installed. Before Version 5.2, the `doc` command only accessed the documentation for MATLAB functions.

Japanese Help Desk

MATLAB 5.2 provided a Japanese version of the Help Desk, in addition to the English version.

Note Most of the Japanese documentation is at a pre-Release 11 level.

During the installation process you can specify what, if any, Japanese documentation you want to install.

If you install any Japanese documentation, the Japanese Help Desk will be displayed when you use the `helpdesk` command.

ActiveX Support Enhanced

MATLAB 5.2 provided support for two new ActiveX technologies: ActiveX control containment and ActiveX Automation client capabilities. ActiveX controls are application components that can be both visually and programmatically integrated into an ActiveX control container; in the MATLAB context, this would be figure windows. Some examples of useful ActiveX controls are the Microsoft Internet Explorer Web Browser control, the Microsoft Windows Communications control for serial port access, and the graphical user interface (GUI) controls delivered with the Visual Basic development environment.

Before 5.2, MATLAB supported ActiveX Automation *server* capabilities. When MATLAB is controlled by another component, it is acting as an automation server. MATLAB 5.2 added support for ActiveX Automation *client* capabilities. When MATLAB controls another component, MATLAB is the automation client, and the other component is the automation server. In other words, MATLAB 5.2 ActiveX Automation allowed MATLAB to both control and be controlled by other ActiveX components.

This feature is described in more detail in Chapter 7 of the *Application Program Interface Guide*.

HDF File Format Support

MATLAB 5.2 extended the support for HDF files beyond that previously provided by `imread` and `imwrite`. This additional support is provided through an interface to different HDF formats via new MATLAB functions that enable you to access the HDF library developed and supported by the National Center for Supercomputing Applications (NCSA). MATLAB 5.2 also provided an extensible gateway for reading and writing HDF files.

To use these functions, you must be familiar with the HDF library. Documentation for the library is available on the NCSA HDF Web page at `http://hdf.ncsa.uiuc.edu`. MATLAB provides extensive command line help for each of these functions.

Function	Interface
`hdfan`	Multifile annotation
`hdfdf24`	24-bit raster image
`hdfdfr8`	8-bit raster image
`hdfh`	HDF H interface
`hdfhd`	HDF HD interface
`hdfhe`	HDF HE interface
`hdfml`	Gateway utilities
`hdfsd`	Multifile scientific data set
`hdfv`	Vgroup
`hdfvf`	Vdata VF functions
`hdfvh`	Vdata VH functions
`hdfvs`	Vdata VS functions

Visualization Enhancements

Support for OpenGL Renderers

The OpenGL renderer is available on many computer systems. This renderer is generally faster than MATLAB's painters or Z-buffer renderers. If your system has graphics hardware that is available to OpenGL, MATLAB uses it to achieve even greater performance improvements. This results in greatly improved drawing performance, particularly with graphics cards that support OpenGL. See the Figure `Renderer` property in the online MATLAB Function Reference for more information.

New View Control Commands

MATLAB 5.2 contained a number of new commands that simplify camera positioning and aspect ratio control. These commands implement operations similar to those associated with movie camera operation – dollying, panning, rolling, as well as some that are more typically associated with computer graphics, such as orbiting the camera around the scene and selecting a method for projecting the three-dimensional scene on the computer screen.

Complex Camera Operations

This table lists commands that simplify the process of moving the camera in a well defined manner through three-dimensional space.

Function or Property	Purpose
`camdolly`	Move camera position and camera target.
`camorbit`	Orbit about camera target.
`campan`	Rotate camera target about camera position.
`camroll`	Rotate camera about viewing axis.
`camzoom`	Zoom camera in or out.

Camera and Axis Control

This table lists new commands that provide a convenient way to set axes properties. These properties control camera positioning as well as axis limits and aspect ratio.

Function or Property	Purpose
`campos`	Set or get camera position.
`camproj`	Set or get projection type.
`camtarget`	Set or get camera target.
`camup`	Set or get camera up-vector.
`camva`	Set or get camera view angle.
`daspect`	Set or get data aspect ratio.
`pbaspect`	Set or get plot box aspect ratio.
`xlim`	Set or get the current x-axis limits.
`ylim`	Set or get the current y-axis limits.
`zlim`	Set or get the current z-axis limits.

New Lighting Convenience Commands

MATLAB 5.2 added two new commands to simplify the placement of light objects in the axes.

Function or Property	Purpose
`camlight`	Create or posititition a light object in the camera's coordinate system.
`lightangle`	Create or position a light object in spherical coordinates.

Support for Predefined Paper Types

MATLAB supports a number of new predefined paper types. For a list of these paper types, see the figure `PaperType` property.

Mechanism to Hide Objects from Selection

All graphics objects have a new property called `HitTest` that enables you to determine if this object can become the current object or in appropriate cases, the current figure or current axes (see the figure `CurrentObject` and `CurrentAxes` properties and the root `CurrentFigure` property). This feature is useful to exclude certain graphics objects from user interaction (for example, to prevent MATLAB from selecting text annotations that overlay an image as the user clicks on the image to obtain information returned by a callback routine). See the `HitTest` property for an example.

New Behavior for newplot, clf, and cla

The behavior of the `newplot`, `clf`, and `cla` commands is now clearly defined with respect to hidden-handle objects. There are basically three options when drawing graphics in existing figures:

- Add the new graphics without changing any properties or deleting any objects.
- Delete all existing objects whose handles are not hidden before drawing the new objects.
- Delete all existing objects regardless of whether or not their handles are hidden and reset most properties to their defaults before drawing the new objects.

These features are particularly useful for protecting Uicontrol objects that compose part of a user interface constructed with MATLAB.

Behavior of newplot

The newplot function now always sets the Figure NextPlot property to add after obeying the current setting. Previously, newplot:

- Did not reset the Figure NextPlot property if its current value was replacechildren.
- Did set the NextPlot property to its currently defined default after obeying its value of replace. (While the factory default is add, user-defined settings can change this.)

With MATLAB 5.2, newplot:

- Always reset the Figure NextPlot property to add after obeying the current setting (regardless of user-defined defaults set for NextPlot).
- Deleted all handle-visible children (i.e., children whose HandleVisibility property is set to on) when the Figure or Axes NextPlot property is replacechildren.
- Deleted all children (regardless of the setting of the HandleVisibility property) when the Figure or Axes NextPlot property is replace.

Behavior of clf and cla

The behavior of the clf command without the reset argument has not changed: clf deletes all children of the current Figure whose handles are not hidden (i.e., their HandleVisibility property is set to on).

clf reset now deletes all children of the current Figure, regardless of the setting of their HandleVisibility property. In addition, clf reset also resets all Figure properties to their defaults with the exception of Position, Units PaperPosition, and PaperUnits. Previously, clf reset deleted only handle-visible objects.

cla behaves in a way directly analogous to that of clf: cla deletes all children of the current Axes whose handles are not hidden (i.e., their HandleVisibility property is set to on).

cla reset deletes all children of the current Axes, regardless of the setting of their HandleVisibility property. In addition, cla reset also resets all Axes properties to their defaults with the exception of Position and Units.

GUI Development Enhancements

MATLAB 5.2 added several new features to make it easier for you to develop an effective graphical user interface (GUI) for your applications. In the online HTML version of this document, you can use the highlighted links to get more information about these new features.

New Units Property Value

If you write user interfaces intended to be used on more than one computer platform, you may find that you need to adjust the size of controls to accommodate the differences in the size of the fonts used to label the controls.

The new `characters` value for the `Units` property enables you to define Uicontrol objects whose sizes are based on the default system font size.

Tooltips

A tooltip is a small rectangle that contains textual information. A tooltip is associated with a Uicontrol and appears below the control when the cursor is held over the control for a certain amount of time (determined by system settings).

You define a tooltip for a Uicontrol by specifying a string value for the new `TooltipString` property.

Toggle Buttons

MATLAB 5.2 provides a new style of Uicontrol object called a toggle button. Toggle buttons have two states, down (selected) and up (unselected). When you click on a toggle button, its state changes and its callback is executed.

Displaying Truecolor Images on Controls

MATLAB 5.2 supported the ability to display truecolor images on push buttons and toggle buttons.

Context Menus

A context menu is a menu that is attached to an object and is activated by a right-button click. Defining a context menu requires that you define a Uicontextmenu object and Uimenu children and associate the Uicontextmenu with the object to which it is attached.

MATLAB Compiler

Compatibility Release

Version 1.2 of the MATLAB Compiler was a compatibility release that brought the MATLAB Compiler into compliance with MATLAB 5. Although the 1.2 Compiler works with MATLAB 5, it does not support several of the new features of MATLAB 5.

Improved Installation and Configuration Process

Installing and configuring the MATLAB Compiler 1.2 was made easier than before. The *MATLAB Compiler User's Guide* includes a complete set of recommended steps to perform during installation to ensure that everything is working properly. It includes troubleshooting sections to help you diagnose and correct some of the more common installation problems.

Enhanced Support for Windows 95 and NT Compilers

In MATLAB 5.2 and the Compiler, all of the main compiler vendors and product releases were supported "out of the box" (no additional steps required). These compilers include:

- Watcom 10.6
- Borland 5.0
- MSVC 4.2

Building Simulink CMEX S-Functions

The MATLAB Compiler now supports building Simulink CMEX S-functions from the MATLAB Function block in Simulink. See the *MATLAB Compiler User's Guide* for details.

Additional Enhancements

Version 1.2 of the MATLAB Compiler also includes these enhancements:

- Loading MATLAB MAT-files (using the `load` command) is now supported.
- Compiler-generated command line applications can accept input arguments (text-strings) from a `POSIX` shell and return a status. In this way command line M-files can be turned into command line executable applications in a `POSIX` shell.

MATLAB C Math Library 1.2

Compatibility Release

Version 1.2 of the MATLAB C Math Library is a compatibility release that brings the library into compliance with MATLAB 5. Although the library works with MATLAB 5, it does not support several of the new features of MATLAB 5.

Note Many functions have changed between MATLAB 4 and MATLAB 5. These changes are reflected in the MATLAB C Math Library. If you are using the MATLAB Compiler to generate your C Math Library programs, you will need to regenerate your C files from your MATLAB 4 M-files before the C files will work with the new libraries. If you have written C Math Library programs by hand, you need to make the changes manually.

New Features

Version 1.2 of the C Math Library added 47 new functions, providing several significant new features, including:

- Indexing (operations like MATLAB's `'x(:, 4) = 1'`). This adds three new functions: `mlfArrayAssign`, `mlfArrayRef`, and `mlfArrayDelete`.
- The MATLAB 5 suite of ODE solvers. The Version 1.1 library supported only two ODE routines; Version 1.2 supports six of them.
- String handling functions.
- Support for `feval` with multiple output arguments (return values); Version 1.1 supported `feval` of functions with a single output argument.
- `load` and `save` support.
- Improved user-defined error handling.

MATLAB C++ Math Library 1.2

Compatibility Release

Version 1.2 of the MATLAB C++ Math Library was a compatibility release that brought the library into compliance with MATLAB 5. Although the library works with MATLAB 5, it does not support several of the new features of MATLAB 5.

Note Many functions have changed between MATLAB 4 and MATLAB 5. However, through the use of C++ function overloading, most of the old functions remain for backward compatibility, and new functions have been added to handle the new functionality (in most cases, with additional function arguments).

If you are generating C++ Math Library programs using the MATLAB Compiler, the changes in functions between MATLAB 4 and 5 should not affect you very much, because the MATLAB Compiler knows about the new functions and generated the correct code. You will, in some cases, however, have to regenerate your C++ code from your M-files to use the new libraries. If you have written stand-alone programs by hand, you may have to edit some of your code before you can link with the new libraries.

New Features

Version 1.2 of the C++ Math Library added 47 new functions, providing several significant new features, including:

- The MATLAB 5 suite of ODE solvers. The Version 1.1 library supported only two ODE routines; Version 1.2 supports six of them.
- String handling functions.
- Support for `feval` with multiple output arguments (return values); Version 1.1 supported `feval` functions with a single output argument.
- `load` and `save` support.
- Improved user-defined error handling.

Simulink 2.2

Simulink 2.2 added many enhancements relating to these aspects of the product:

- User Interface
- Simulation
- Model Construction Commands
- Printing

These enhancements are described in more detail in the online (PDF) version of *Using Simulink*.

User Interface

Note See Chapter 3 of *Using Simulink* for more information about each of these new user interface features.

Toolbar

The PC (Microsoft Windows 95 and NT) version of Simulink displays an optional toolbar below the menu bar in the model and block library windows. You can use the toolbar's buttons to create, save, edit, print, and run models.

Status Bar

The PC version of Simulink displays an optional status bar at the bottom of model and block library windows. The status bar displays the current time and solver when a simulation is running.

Context-Sensitive Menus

The PC version of Simulink displays a popup menu when you press the right mouse button over a model or block library window. If a block is selected, the menu displays editing, formatting, and property commands applicable to blocks and annotations; otherwise, the menu displays commands applicable to the model or library as a whole.

Automatic Block Connection

You can insert a block having a single input and output into a model by dropping it onto a line segment.

Block Properties Dialog Box

Simulink 2.2 added a **Block Properties** dialog box, accessed from the **Edit** menu. You can set the following block parameter values:

- Description
- Priority
- Open function
- Attribute format

Undoing Breaking of Library Links

Simulink 2.2 allows you to undo the breaking of library links.

Simulation

Block Priorities

You can assign evaluation priorities to nonvirtual blocks in a model. Higher priority blocks evaluate before lower priority blocks, though not necessarily before blocks that have no assigned priority. You can do this with either the **Block Properties** dialog box from the **Edit** menu or with the `set_param` command. See Chapter 3 of *Using Simulink* (online version) for more information.

Additional Solvers

Simulink 2.2 adds two stiff solvers, `ode23t` and `ode23tb`. See the section about solvers in Chapter 4 of *Using Simulink* (online version) for more information.

Debugger

The Simulink debugger allows you to run a model step by step and inspect the values of any variables at any step. See Chapter 11 of *Using Simulink* for more information.

Tunable Mask Parameters

You can specify whether a mask parameter is tunable, that is, modifiable while a simulation is running. See Chapter 6 of *Using Simulink* for more information.

Level 2 S-Functions

Simulink 2.2 supports Level 2 S-functions in a C MEX S-function. In particular, these Level 2 S-functions support:

- Multiple input and output ports
- More simulation S-function routines:
  - `mdlProcessParameters`, which is called during simulation after parameters have been changed and verified by `mdlCheckParameters`
  - `mdlStart`, which performs actions such as allocating memory and attaching to PWork elements.
- `mdlRTW`, a method for code generation in which your S-function influences the code generation process.

  In `mdlRTW`, you can write additional subrecords into the `model.rtw` file for the S-function block record. The Target Language Compiler (TLC) file that inlines your S-function can use this information. For more details about Level 2 S-functions, see *Writing S-Functions*.

Merge Block

The Merge block allows you to combine multiple input lines into a single output line for reduced memory utilization and increased model flexibility. See Chapter 8 of *Using Simulink* (online version) for more information.

Non-Algebraic Feedback Loops

Prior to Version 2.2, Simulink treated as algebraic loops any loops that involved Triggered Subsystems and that were also composed entirely of blocks with direct feedthrough.

With Version 2.2, for Variable-Step solvers, Simulink now takes advantage of the implicit sequencing inherent in triggered execution (i.e., inputs must be stable prior to the trigger, and outputs appear after the trigger) to break such loops, thus:

- Eliminating the need to invoke the algebraic loop solver
- Providing more meaningful results

For Fixed-Step solvers, it is still necessary to insert a memory block in the appropriate location (usually at the output of the triggered subsystem) to break such algebraic loops.

See “Algebraic Loops” in Chapter 9 of *Using Simulink* (online version) for more information.

Model Construction Commands

Object Parameters

The command `get_param(obj,'ObjectParameters')` where `obj` is an object name returns a cell array describing the object’s parameters. See `get_param` in *Using Simulink* (online version) for more information.

Dialog Parameters

The command `get_param(b,'DialogParameters')`, where `b` is the name of a block, returns a cell array describing the parameters that appear in a block’s parameter dialog. See `get_param` in *Using Simulink* (online version) for more information.

Lines/Annotations API

You can use the `find_system` command to get handles to all the lines and annotations in a model. The returned handles can be used with `get_param` and `set_param` to read and write the line or annotation properties. See `find_system` in Chapter 10 of *Using Simulink* (online version) for more information.

Printing

Print Frames

You can add print frames (customized headers and footers) to printouts of Simulink model diagrams. To edit a print frame, use the new `frameedit` command. See "Printing a Block Diagram" in Chapter 3 of *Using Simulink* (online version) for more information.

Real-Time Workshop 2.2

Asynchronous Processes

The Real-Time Workshop added support for asynchronous interrupt handling in VxWorks and provides templates so that you can create your own interrupt handlers for your target hardware. These blocks include:

- Interrupt block
- Task Synchronization block
- Asynchronous Buffer Reader/Writer blocks
- Asynchronous Rate Transition block

For a discussion of asynchronous processes, see the chapter on `RTWlib` in the *Real-Time Workshop User's Guide* (online version).

RTWlib

The Real-Time Workshop now has a graphical user interface (GUI), called RTWlib, for quick access to:

- VxWorks Tornado — blocks for Matrix, Xycom, and VME Microsystems I/O support; and asynchronous blocks for ISR support
- DOS — blocks for Keithley-Metrabyte I/O support
- Custom Code — blocks for inserting your custom code into the code that the Real-Time Workshop generates from your model
- Create Your Own Asynchronous Library — templates that use the VxWorks asynchronous blocks as a starting point for developing your own asynchronous blocks
- Real-Time Workshop extras — contains a Function-call Configuration block

The GUI is located in the "Blocksets and Toolboxes" Library in the Simulink window. For more information about RTWlib, see the *Real-Time Workshop User's Guide*.

Merge Block Added

The new Merge block merges multiple signals into one for reduced memory utilization and increased model flexibility.

Level 2 S-Functions

Real-Time Workshop 2.2 supports Level 2 S-functions. In particular, these Level 2 S-functions support:

- Multiple input and output ports
- More simulation S-function routines:
  - `mdlProcessParameters`, which is called during simulation after parameters have been changed and verified by `mdlCheckParameters`
  - `mdlStart`, which performs actions such as allocating memory and attaching to pwork elements. Can only be in a C MEX S-function
- `mdlRTW`, a method for code generation in which your S-function influences the code generation process.

  In `mdlRTW`, you can write additional subrecords into the `model.rtw` file for the S-function block record. The Target Language Compiler (TLC) file that inlines your S-function can use this information. For more details about Level 2 S-functions, see *Writing S-Functions*.

Target Language Compiler (TLC) Enhancements

This section describes enhancements to the Target Language Compiler that is included as part of the Real-Time Workshop.

Passing Parameters: mdlRTW and RTWData

The Real-Time Workshop generates a `model.rtw` file that is a description of the model. There are two additional methods of passing user-specified information into the `model.rtw` file:

- `mdlRTW` — Used with Level 2 S-functions
- `RTWData` — Used with any nonvirtual Simulink block and with empty subsystems

mdlRTW. Level 2 S-functions can use the `mdlRTW` function to pass information from a C-MEX S-function into the `model.rtw` file for use during code generation.

The information that the `mdlRTW` function writes to `model.rtw` is used by the block target file for that block type. The writer of the block target file can use the additional identifier/value pairs as desired. For all the possible functions that you can use inside `mdlRTW` to generate information in the `model.rtw` file, see the file *`matlabroot`*`/simulink/src/sfuntmpl.doc`. See Chapter 8 of *Using Simulink* for a discussion of how to write an `mdlRTW` function.

Below is an example of how to use `mdlRTW` in a Level 2 S-function.

```
static void mdlRTW(SimStruct *S)
{
    int_T numElements = mxGetNumberOfElements(TASK_NAME);
    char *buf = NULL;

    if ((buf = malloc(numElements +1)) == NULL) {
        ssSetErrorStatus(S,"memory allocation error in mdlRTW");
        return;
    }
    if (mxGetString(TASK_NAME,buf,numElements+1) != 0) {
        ssSetErrorStatus(S,"mxGetString error in mdlRTW");
        free(buf);
        return;
    }
    /* Write out the parameters for this block.*/
    if (!ssWriteRTWParamSettings(S, 3,
         SSWRITE_VALUE_QSTR,"TaskName", buf,
         SSWRITE_VALUE_NUM,"Priority",
         (real_T) (*(mxGetPr(PRIORITY))),
         SSWRITE_VALUE_NUM,"StackSize",
         (real_T) (*(mxGetPr(STACK_SIZE))))) {
      return; /* An error occurred which will be reported by SL */
    }

    /* Write out names for the IWork vectors.*/
    if (!ssWriteRTWWorkVect(S, "IWork", 1, "TaskID", 1)) {
      return; /* An error occurred which will be reported by SL */
    }

    /* Write out names for the PWork vectors.*/
    if (!ssWriteRTWWorkVect(S, "PWork", 1, "SemID", 1)) {
      return; /* An error occurred which will be reported by SL */
    }

    free(buf);
}
```

This code contains the resulting `model.rtw` information:

```
Block {
  . . .
  SFcnParamSettings {
                 TaskName""
                 Priority20
                 StackSize1024
  }
  NumIWorkDefines      1
  IWorkDefine {
                 Name        "TaskID"
                 Width       1
  }
  NumPWorkDefines      1
  PWorkDefine {
                 Name        "SemID"
                 Width       1
  }
}
```

RTWData. `RTWData` is a parameter that you can set on Simulink blocks using the `set_param()` command and view with the `get_param()` command. The parameter/value pair is saved along with the model.

The command syntax is

```
set_param(gcb,'rtwdata',userdata)
```

where `gcb` is the current block pathname. The variable `userdata` must be a MATLAB data structure where each element is a string. For example:

```
userdata.a = 'rpm'
userdata.b = '1.25'
```

When attached to a nonvirtual block, the associated `model.rtw` information for the block is:

```
Block {
. . .
  RTWdata {
    a    "rpm"
    b    "1.25"
  }
}
```

The block target file for that block type can process the information as desired. For example, if `RTWData` is attached to a S-function, the TLC inlining file for the S-function could process the information in the `BlockInstanceSetup` function.

Besides nonvirtual blocks, `RTWData` can be attached to one special case of a virtual block, an empty subsystem. This allows information the be passed into the `model.rtw` without it being associated with a specific nonvirtual block. This is useful when some block-independent information needs to be passed into `model.rtw` for use during code generation. For empty subsystems, the `RTWData` parameter is placed in the `System` record for the nonvirtual system in which the empty subsystem is contained.

```
System {
. . .
  EmptySubsysInfo {
    NumRTWdatas    1
    RTWdata {
      a    "rpm"
      b    "1.25"
    }
  }
}
```

Because the empty subsystem technique is used by the Custom Code block of the `RTWLib`, there is support built into the system target files to handle `RTWData` attached to empty subsystems. Specifically, if an `EmptySubsysInfo` record exists, all `RTWdata` subrecords are checked for the existence of an identifier named `TLCFile`. If the identifier exists, the value of `TLCFile` is used as a block target filename and the TLC function `ProcessRTWdata` in that file is called using the `TLC GENERATE` directive. This functionality can also be used by other (user-written) blocks if desired.

Stateflow 1.0.6

Version 1.0.6 of Stateflow and Stateflow Coder was shipped with MATLAB 5.2. Version 1.0.6 is essentially the same as the Patch Release 1.0.5 that was made available to Stateflow customers via FTP. However, 1.0.6 fixes some software problems that still existed in the patch release.

Toolboxes and Blocksets

Almost all of the toolboxes and blocksets were updated for release with MATLAB 5.2. For many of these toolboxes and blocksets, the updates simply involved fixing software problems and taking more advantage of MATLAB 5 features.

These toolboxes and blocksets were updated for 5.2. The toolboxes and blocksets with significant updates are highlighted with an asterisk and are discussed in more detail in the rest of this chapter (in alphabetical order).

- Communications Toolbox 1.3*
- Control System Toolbox 4.1*
- DSP Blockset 2.2*
- Extended Symbolic Math Toolbox 2.0.1
- Financial Toolbox 1.1*
- Frequency Domain System Identification 2.0.3
- Fuzzy Logic Toolbox 2.0*
- Higher-Order Spectral Analysis Toolbox 2.0.2
- Image Processing Toolbox 2.1*
- LMI Control Toolbox 1.0.4
- Mapping Toolbox 1.0.1
- Model Predictive Control Toolbox 1.0.3
- Mu-Analysis and Synthesis Toolbox 3.0.3
- NAG Foundation Blockset 1.0.3 (for Sun4, Sol2, Alpha, SGI, and SGI64)
- Neural Network Toolbox 3.0*
- Optimization Toolbox 1.5.2
- Partial Differential Equation Toolbox 1.0.3
- QFT Control Design Toolbox 1.0.3
- Robust Control Toolbox 2.0.5
- Signal Processing Toolbox 4.1*
- Spline Toolbox 2.0*
- Statistics Toolbox 2.1.1

- System Identification Toolbox 4.0.4
- Wavelet Toolbox 1.1

Power System Blockset 1.0

The Power System Blockset is a new blockset introduced with MATLAB 5.2.

The Power System Blockset is a modern design tool that allows scientists and engineers to build models rapidly and easily that simulate power systems. The blockset uses the Simulink environment, allowing a model to be built using simple *click-and-drag* procedures. Not only can you draw the circuit topology rapidly, but the analysis of the circuit can include its interactions with mechanical, thermal, control, and other disciplines. This is possible because the electrical portions of the simulation interact with Simulink's extensive modeling library. Because Simulink uses MATLAB as the computational engine, MATLAB's toolboxes can also be used by the designer.

Power System Blockset libraries contain models of typical power equipment such as transformers, lines, machines, and power electronics. Their validity is based on the experience of the Power Systems Testing Laboratory of Hydro-Quebec, a large North American utility located in Canada.

See the *Power System Blockset User's Guide* for information about using this blockset.

Communications Toolbox 1.3

Note Much of the new functionality of the Communications Toolbox 1.3 requires Simulink 2.2. However, even if you use the Communications Toolbox without Simulink, upgrading to Version 1.3 will let you take advantage of a number of other software quality improvements in the toolbox.

The Communications Toolbox 1.3 added 22 new Simulink function blocks and 12 new example block diagrams.

The new function blocks are:

- Passband digital modulation/demodulation blocks
- Interleave and scrambler blocks

These new blocks expand the functionality of the Communications Toolbox so that it now provides:

- Five new phase-shift keying modulation/demodulation methods
- Three new phase-shift keying mapping/demapping techniques
- Differential encoding/decoding
- Block interleaving and deinterleaving
- Scrambling/descrambling
- Pseudorandom sequence generation

The Communications Toolbox 1.3 also builds on recent MATLAB and Simulink enhancements. These minor changes to the Communications Toolbox are primarily in the area of graphical scopes such as the Error Rate Meter, Eye-Pattern and Scatter plots, and the Trellis plot in the Convolutional Decode block.

This release of the Communications Toolbox also includes changes made to ensure integration with the Real-Time Workshop 2.2. If you are using Real-Time Workshop with the Communications Toolbox 1.3, you need Real-Time Workshop 2.2. Specifically, a few parameter definitions in the Communications Toolbox have been changed for use with C-coded S-functions in Real-Time Workshop.

See the *Communications Toolbox 1.3 New Features Guide*, available in printed form and online (PDF), for more details on these new features.

Control System Toolbox 4.1

The Control System Toolbox 4.1 provided two main enhancements:

- The Root Locus Design GUI (graphical user interface)
- The Simulink LTI Viewer

The Root Locus Design GUI is an interactive design tool that you can use to:

- Implement root locus methods on single input-single output (SISO) LTI models defined using `zpk`, `tf`, or `ss`
- Specify the parameters of a feedback compensator: poles, zeros, and gain
- Examine how changing the compensator parameters affects the root locus, as well as various closed-loop system responses (step response, Bode plot, Nyquist plot, or Nichols chart)

The Root Locus Design GUI is documented in Chapter 6 of the *Control System Toolbox User's Guide*.

The Simulink LTI Viewer is similar to the Control Systems Toolbox LTI Viewer. The Simulink LTI Viewer is used to analyze portions of a Simulink model. Its features include:

- Drag-and-drop blocks that identify the location for the inputs and outputs of the portion of a continuous-time Simulink model you want to analyze
- The ability to specify the operating conditions about which the Simulink model is linearized
- Access to all time and frequency response tools featured in the regular Control System Toolbox LTI Viewer
- The ability to compare a set of linearized models obtained from the same Simulink diagram by varying either the operating conditions or some model parameter values

The Simulink LTI Viewer is documented in Chapter 4 of the *Control System Toolbox User's Guide*.

Two additional enhancements are:

- Sharper Root Locus plots
- An Export option for the LTI Viewer

DSP Blockset 2.2

DSP Blockset 2.2 introduced a number of new features and improvements. There are over 30 new and enhanced blocks, a filter design wizard, support for data frames, and expanded support of vector and matrix inputs. This section outlines the new additions and provides pointers to the complete feature descriptions in the *DSP Blockset User's Guide*. See Chapter 1 of the online *User's Guide* for an overview of the blockset's contents.

Also see the DSP Blockset `readme` file for a summary of the new additions. To view the `readme` file, type

```
info dspblks
```

at the MATLAB command line.

Data Frames

The DSP Blockset added support for data *frames*, vectors whose elements represent consecutive time samples from a single signal. Framed data is a common format in real-time systems, where the data acquisition hardware often operates most efficiently by accumulating a large number of signal samples at a high rate, and then propagating these samples to the real-time system as a block, or frame, of data. Data frames can also be constructed through the usual DSP Blockset buffering operations (using the Buffer and Complex Buffer blocks, for example).

Version 2.2 includes two new blocks designed to operate specifically on framed data. They are frame-oriented counterparts to the FIR Rate Conversion and Multichannel IIR Filter blocks and are distinguished by the word "Frame" in the block name:

- FIR Rate Conversion (Frame)
- Multichannel IIR Filter (Frame)

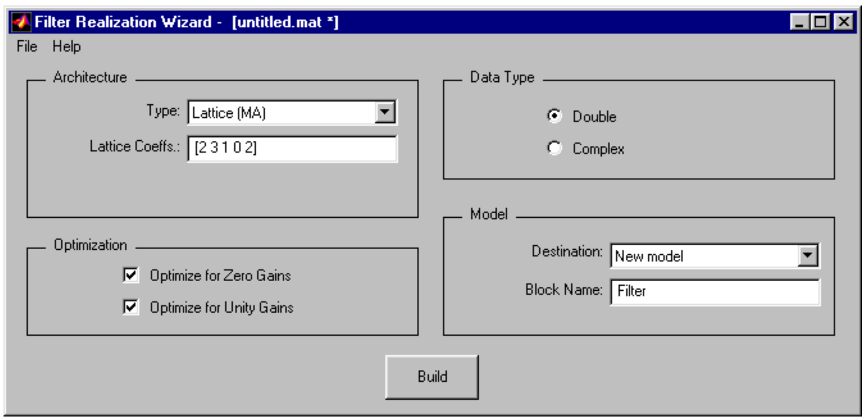
Use these blocks to directly filter or resample framed data in its native format without the computational expense of unbuffering. Other blocks that operate on framed data include the FFT, DCT, and cepstrum blocks in the Transforms library.

In addition to these frame-based blocks, the data frame format is accepted by all blocks in the blockset that accept vector inputs. Be aware, however, that many blocks implicitly expect the elements of vector inputs to represent *independent channels* and not consecutive samples. Besides the FIR Rate Conversion and Multichannel IIR Filter blocks, others that expect *non-frame* data include the "running" blocks in the Statistics library, the variable delay blocks, and the filter design blocks. In general, if a block uses past inputs in generating the current output (and is not specifically designated as a frame-based block), then it considers the elements of a vector (or matrix) input to represent distinct channels, and *not* a frame of consecutive samples.

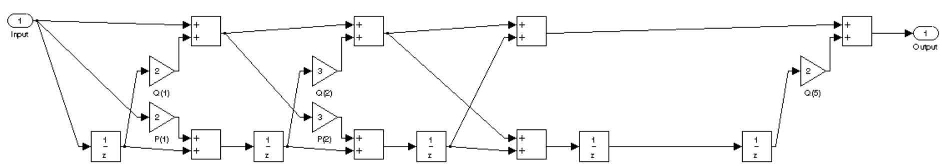
See “Working with Frames” in Chapter 3 of the *User’s Guide* for a complete discussion of this data format.

Filter Realization Wizard

Another new element of the blockset is the Filter Realization Wizard, a GUI that allows you to construct filters easily with a a variety of different architectures. The GUI is shown below.



When you click the GUI’s **Build** button with the particular settings shown above, the wizard constructs the specified moving average (MA) lattice architecture as a subsystem within a new model window.



You can then alter or optimize the filter to suit your own needs. Additional information about the Filter Realization Wizard can be found in the online Reference.

New and Enhanced Blocks

The table below lists the blocks added in Version 2.2. Among the most significant additions were variable delay blocks, discrete cosine transform and cepstrum blocks, linear prediction blocks (LPC, Levinson-Durbin), and new spectral estimation blocks.

Block Library	Block Name	Purpose
DSP Sources	Complex Diagonal Matrix	Generate a square, constant-diagonal complex matrix
DSP Sinks	Triggered Complex Matrix To Workspace	Send a time sequence of complex matrices to the MATLAB workspace
	Triggered Complex To Workspace	Write the time sequence of a complex input to the MATLAB workspace
	Triggered Matrix To Workspace	Send a time sequence of matrices to the MATLAB workspace
	Triggered To Workspace	Write the time sequence of an input to the MATLAB workspace
Signal Operations	Complex Delay	Delay a complex input by an integer number of sample periods
	Complex Levinson-Durbin	Apply Levinson-Durbin recursion to design an IIR filter with a prescribed autocorrelation sequence
	Complex LPC	Determine the coefficients of an FIR filter that predicts the next sequence value from past and present inputs
	Levinson-Durbin	Apply Levinson-Durbin recursion to design an IIR filter with a prescribed autocorrelation sequence
	LPC	Determine the coefficients of an FIR filter that predicts the next sequence value from past and present inputs
	Variable Fractional Delay	Delay an input by a fractional number of sample periods
	Variable Integer Delay	Delay an input by an integer number of sample periods
Transforms	Complex Cepstrum	Compute the complex cepstrum of an input
	DCT	Compute the DCT of a complex vector input
	IDCT	Compute the complex-valued IDCT of a complex input
	Real Cepstrum	Compute the real cepstrum of an input
	Real DCT	Compute the DCT of a real vector input
	Real IDCT	Compute the IDCT of a real input

Block Library	Block Name	Purpose
Buffers	Shift Register	Convert a scalar time series into a vector time series with the same sample period (serial-to-parallel conversion)
	Triggered Shift Register	Convert a scalar time series into a vector time series with the same sample period (serial-to-parallel conversion)
Switches and Counters	N-Sample Enable w/Reset	Output 1s for a specified number of sample times
	Sample and Hold	Sample and hold an input signal
Vector Math	Autocorrelation	Compute the autocorrelation of a real vector
	Complex Autocorrelation	Compute the autocorrelation of a complex vector
Complex	Complex Gain	Multiply an input by a complex constant
	Real to Complex	Construct a complex output from a real input
Statistics	Histogram	Compute the histogram (frequency distribution) of values in a vector input
	Median	Find the median value of a vector input
	Running Histogram	Track frequency distribution of values in a vector input over time
	Sort	Sort the elements in a vector by value
Filter Realizations	Filter Realization Wizard	Build an IIR or FIR filter with a particular architecture
	Multichannel IIR Filter (Frame)	Apply an IIR filter to a multichannel input signal
	Time Varying FIR Filter	Apply a variable FIR filter to a multichannel input signal
	Time Varying IIR Filter	Apply a variable IIR filter to a multichannel input signal
Multirate Filters	FIR Rate Conversion (Frame)	Upsample, filter, and downsample a real input
Spectrum Analysis	Burg Method	Compute a parametric estimate of the spectrum using the Burg method
	Yule-Walker AR	Compute a parametric estimate of the spectrum using the Yule-Walker AR method

In addition to the new blocks, several blocks were enhanced for Version 2.2, and are highlighted in the table below. The most important area of growth among the existing blocks is in the expanded support of vector and matrix inputs for buffering and unbuffering operations.

Block Library	Block Name	Enhancement
DSP Sources	Diagonal Matrix	Allows specification of a nonconstant diagonal
DSP Sinks	Frequency Vector Scope	Offers new menus, and window position memory
	Time Vector Scope	Offers new menus, and window position memory
Signal Operations	Complex Zero Pad	Offers the option of truncating the input to the specified output vector length
	Delay	Accepts an initial condition
	Zero Pad	Offers the option of truncating the input to the specified output vector length
Buffers	Buffer	Supports vector inputs, and accepts an initial condition
	Complex Buffer	Supports vector inputs, and accepts an initial condition
	Complex Partial Unbuffer	Supports matrix inputs
	Complex Unbuffer	Supports matrix inputs
	Partial Unbuffer	Supports matrix inputs
	Unbuffer	Supports matrix inputs
Switches and Counters	Commutator	Supports matrix inputs
	Distributor	Supports vector inputs, and accepts an initial condition
Multirate Filters	FIR Rate Conversion	Supports matrix inputs

For Users Upgrading from Version 1.0a

The DSP Blockset 2.2 is completely compatible with Version 1.0a, but there are some limitations on mixing buffer blocks from the two versions, and you will need to recompile any custom blocks that use C-MEX S-functions so that they work with Simulink 2.2.

See "Upgrading to DSP Blockset 3.0 and Communications Toolbox 1.4" in Chapter 4 for more details about upgrading from Version 1.0a.

Financial Toolbox 1.1

The Financial Toolbox 1.1 supports detailed term structure analysis. In addition, this version provided new date functions, coupon date functions, portfolio allocation tools, and a new derivative pricing function. These new functions are summarized below.

For information about these functions, refer to the *Financial Toolbox User's Guide*.

Term Structure Functions

Function	Description
`disc2zero`	Zero rate curve from a discount curve.
`fwd2zero`	Forward rate curve from a zero curve.
`pyld2zero`	Par yield curve from a zero curve.
`tbl2bond`	Conversion of TBills to TBond market convention.
`termfit`	Demo function for smoothing rates with splines.
`tr2bonds`	Conversion of Treasury data to bond input format.
`zbtprice`	Bootstrap a zero curve from market bond prices.
`zbtyield`	Bootstrap a zero curve from market bond yields.
`zero2disc`	Discount factors from a zero curve.
`zero2fwd`	Zero curve from a forward curve.
`zero2pyld`	Zero curve from a par curve.

Derivatives Function

Function	Description
`blkprice`	Black's pricing model.

Portfolio Analysis Function

Function	Description
ewcov	Asset covariance estimation with exponential weighting.

Date Functions

Function	Description
accrfrac	Accrued interest coupon period fraction.
busdate	Next or previous business day.
cfdates	Cash flow dates of a security.
datefind	Indices of date numbers in a matrix.
eomdate	Last date of month.
fbusdate	First business date of month.
holidays	Holidays and nontrading days.
ibusday	True for dates that are business days.
lbusday	Last business date of the month.
lweekdate	Date of last occurrence of weekday in month.
m2xdate	MATLAB serial date number to Excel date number.
months	Number of whole months between dates.
nweekdate	Date of specific occurrence of weekday in month.
yeardays	Number of days in year.
x2mdate	Excel serial date number to MATLAB date number.

Demo of an Excel Link Portfolio Optimizer Tool

The following files provide a demo of an Excel Link portfolio optimizer tool:

- `excelportopt.m`
- `excelportopt.xls`

Fuzzy Logic Toolbox 2.0

The Fuzzy Logic Toolbox 2.0 featured several improvements, including:

- Additional and enhanced GUIs for performing a number of tasks
- Several enhanced Fuzzy Logic algorithms
- Fuzzy Inference Systems (FIS) are represented as MATLAB structures
- More dimensions are allowed in user-defined membership functions

Graphical User Interface Enhancements

Fuzzy Logic Toolbox 2.0 added or enhanced several GUIs:

- GUI for Adaptive Neuro-Fuzzy Inference System (ANFIS) learning.

  With this GUI, you can implement an ANFIS and use automatic membership function adaptation without resorting to the command line. The learning process can also be viewed graphically and in real time, so any necessary adjustment can be made efficiently. The ANFIS Editor is also fully integrated with the other GUI tools: the Fuzzy System Editor, Membership Function Editor, Rule Editor, Rule Viewer, and Surface Viewer. This GUI is described in Chapter 2 of the *Fuzzy Logic Toolbox User's Guide*.
- Membership Function Editor.

  You can click and drag both the shape and the location of your membership functions.
- Rule Editor.

  You can point and click to build your rules easily, rather than typing in long rules.
- GUI for fuzzy clustering.

  This GUI lets you view both fuzzy c-means clustering and subtractive clustering while they are in progress.

- Rule Viewer for the fuzzy Simulink block.

  When a fuzzy inference system is used in Simulink, the Rule Viewer lets you see when each rule is triggered and how each membership function is applied during a simulation.

Fuzzy Algorithm Improvements

The following Fuzzy Logic algorithms have been added or enhanced:

- Backpropagation learning algorithm for ANFIS.

  Backpropagation is now available as an ANFIS learning algorithm.
- Constant output membership functions for ANFIS.

  You can now use constant output membership functions with ANFIS in addition to linear output membership functions.
- Fuzzy arithmetic.

  Basic fuzzy arithmetic functions are now provided for addition, subtraction, multiplication, and division operations among different membership functions.
- Customizable membership function discretization.

  Now you can adjust the sampling rate used to discretize the output membership functions of your rules. This gives you control of the accuracy and efficiency of the defuzzification calculations.

FIS Represented As MATLAB Structures

The Fuzzy Inference System (FIS) is now represented as a MATLAB structure. A structure (instead of a flat matrix) is now the basic element in constructing a fuzzy logic system. This fundamental change in the way of representing the fuzzy logic system makes many details of working with the constructed system easier.

A Fuzzy Inference System that you created with a pre-2.0 version of the Fuzzy Logic Toolbox is still usable in 2.0, if you run the `convertfis` function on it. The convertfis function automatically converts pre-2.0 Fuzzy Inference Systems to work with Version 2.0.

More Dimensions Allowed for User-Defined Membership Functions

You can now use up to 16 parameters when you define your own customized membership functions.

Image Processing Toolbox 2.1

Interactive Pixel Value Display

The new function `pixval` installs in a figure an interactive display of the data values for whatever image pixel the cursor is currently over. You can also click and drag to display the Euclidean distance between two pixels.

Feature Measurement

The new function `imfeature` computes feature measurements, such as the center of mass and the bounding box, for regions in an image.

Inverse Radon Transform

The new function `iradon` uses the inverse Radon transform to reconstruct images from projection data. In addition, the toolbox has a new function, `phantom`, that generates test images for use with the Radon and inverse Radon transforms.

Canny Edge Detector

The `edge` function now supports the Canny edge detection method. This method is better at detecting weak edges and is less sensitive to noise than the other supported edge-detection methods.

Other Enhancements

- The `bwfill` function can now automatically detect and fill holes in objects.
- The toolbox now supports the YCbCr color space with two new functions, `ycbcr2rgb` and `rgb2ycbcr`.
- You can now easily convert images between double precision and `uint8` using two new functions, `im2double` and `im2uint8`.
- You can control whether `imshow` automatically calls `truesize` by setting the new toolbox preference `'ImshowTruesize'`.

Neural Network Toolbox 3.0

The Neural Network Toolbox 3.0 provided several important new features, including:

- Modular network representation.

  All network properties are collected in a single "network object." Networks can have any number of sets of inputs and layers, any input or layer can be connected to any layer with a weight, and any weight can have a tapped delay.

- Reduced memory Levenberg-Marquardt (LM) algorithm.

  The fast LM algorithm (by a factor of 10 to 100 over other methods) can be used in much larger problems than in Version 2.0.
- New algorithms, including:
  - Conjugate gradient
  - R-Prop
  - Two quasi-newton methods
- New network types, including:
  - Probabilistic
  - Generalized Regression
- Automatic regularization and new training options, including:
  - Training on variations of mean square error for better generalization
  - Training against a validation set
  - Training until the gradient of the error reaches a minimum
- Pre- and post-processing functions, such as Principal Component Analysis.
- Better Simulink support: the Neural Network Toolbox now generates network simulation blocks.

These features are summarized in more detail in the "What's New in 3.0" section of the updated *Neural Network Toolbox User's Guide*.

Signal Processing Toolbox 4.1

The Signal Processing Toolbox 4.1 introduces a number of improvements, including a new GUI for the Filter Designer. This section outlines the new additions and provides pointers to the complete feature descriptions in the online (PDF) *Signal Processing Toolbox User's Guide*. The Signal Processing Toolbox `readme` file also contains a short summary of this information.

To view the `readme` file, type at the MATLAB command line

```
info signal
```

Spectral Estimation

The *MEM* spectral estimation method (previously implemented by the `pmem` function) has been more accurately renamed the *Yule-Walker AR* method, and is now implemented by the `pyulear` function. The `pmem` function continues to work, but generates the following warning message:

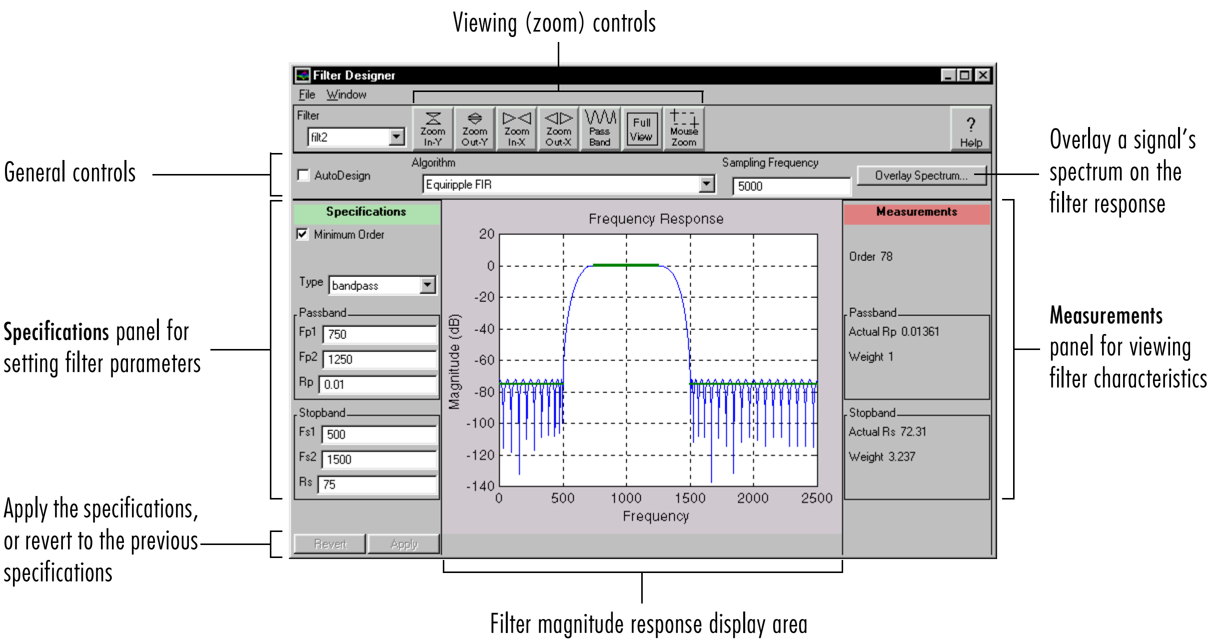
```
Warning: pmem is obsolete and will be discontinued.
Use pyulear instead.
```

In addition to this name change, the *Burg* method of spectral estimation has been added to the toolbox via the `pburg` function.

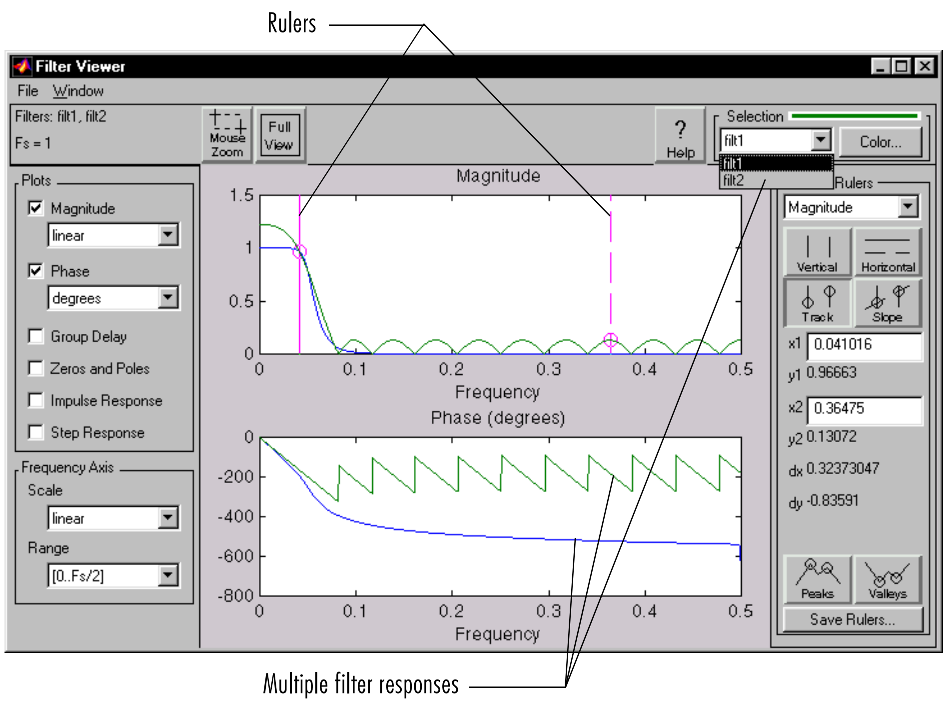
SPTool Graphical User Interface

Several areas of the SPTool interactive signal processing environment have been enhanced for Version 4.1. See Chapter 5 in the PDF version of the *User's Guide* for complete instructions on using the new features.

The Filter Designer interface has been revised for improved usability. A signal's spectrum can now be superimposed on any filter response, and a new **Measurements** panel displays the filter's characteristics as it is being designed.


Viewing (zoom) controls
General controls
Overlay a signal's spectrum on the filter response
Specifications panel for setting filter parameters
Measurements panel for viewing filter characteristics
Apply the specifications, or revert to the previous specifications
Filter magnitude response display area
Filter Designer
File Window
Filter
filt2
Zoom In-Y
Zoom Out-Y
Zoom In-X
Zoom Out-X
Pass Band
Full View
Mouse Zoom
Help
AutoDesign
Algorithm
Equiripple FIR
Sampling Frequency
5000
Overlay Spectrum...
Specifications
Minimum Order
Type bandpass
Passband
Fp1 750
Fp2 1250
Rp 0.01
Stopband
Fs1 500
Fs2 1500
Rs 75
Revert
Apply
Frequency Response
Magnitude (dB)
Frequency
Measurements
Order 78
Passband
Actual Rp 0.01361
Weight 1
Stopband
Actual Rs 72.31
Weight 3.237

The Filter Viewer is now capable of displaying multiple filter responses simultaneously, and also benefits from new rulers that can be used for fine measurement on all of the plot types.



The Spectrum Viewer offers two new spectral estimation methods, the fundamental *FFT* method, and the *Burg* method. Additionally, the *MEM* method has been renamed the *Yule-Walker AR* method. The **MEM** option has been retained in the **Method** pop-up menu for backwards compatibility, but will be removed in a future release. Please use **Yule AR** instead.

General Enhancements

The following enhancements and bug-fixes are also included in the 4.1 release.

- The generalized cosine window functions (`hamming`, `hanning`, and `blackman`) can now generate both periodic and symmetric windows. Formerly, they generated only symmetric windows.
- A bug in `cremez` that produced a complex filter instead of the appropriate real filter has been fixed. Additionally, the `opt.fgrid` and `opt.fextr` outputs are now normalized to the Nyquist frequency.
- The `invfreqz` and `invfreqs` functions now work for complex as well as real filters.

Spline Toolbox 2.0

Multivariate Spline Support

All M-files for the construction of splines (in B-form or ppform) have been expanded to handle tensor-product splines in any number of variables. The same is true for most of the M-files that make use of splines. This means that it is now possible to interpolate, approximate, or smooth gridded data in any number of variables and then evaluate, plot, differentiate, or integrate the resulting multivariate spline.

User Interface Enhancements

In the same spirit of keeping the number of commands small (and of object-oriented programming), most of the form-specific commands (such as `spval` or `ppbrk`) have been replaced by generic commands (such as `fnval` or `fnbrk`). The forms themselves are now structures, but that should be irrelevant to the casual user.

Vector-Valued Spline Enhancements

Since splines in the toolbox can be vector-valued, it is now possible to handle certain surfaces as 3-vector-valued bivariate tensor-product splines.

Additional Enhancements

Further new features include:

- Use of sparse matrices wherever appropriate
- Construction, in `spaps`, of the smoothing spline (linear, cubic, or quintic) that fits given data within a given tolerance
- Conversion, in `fnrfn`, of a given form to one on a refined knot or break sequence
- More flexibility in `fncmb` for arithmetic functions
- More freedom and ease with the input arguments to various M-files
- Optional use of weights in the construction of least-squares and of smoothing splines
- New M-files for helping with the conversion between forms and between breaks and knots and their multiplicities

3

MATLAB 5.1 Enhancements

What Was New in MATLAB 5.1?

Note All the features introduced in MATLAB 5.1 are also in Release 11 (MATLAB 5.3).

The main purpose of the MATLAB 5.1 release was to complete upgrades to the 5.0 level of the entire set of toolboxes and blocksets, and to introduce the Stateflow product. In addition, a number of small but useful enhancements to MATLAB were provided with this release.

Enhancements to MATLAB

MATLAB 5.1 added several enhancements to the MATLAB language and development environment, Handle Graphics®, printing, and the Application Program Interface (API).

The language and development environment enhancements included:

- `find` returns an empty matrix
- Multibyte character support
- Several PC enhancements:
  - Removal of Microsoft Windows TCP/IP requirement
  - Notebook support for Office 97
  - PC Editor/Debugger icons changed

The Handle Graphics and printing enhancements included:

- Scatter plot functions
- X-Windows support for `uisetcolor` (UNIX)
- Patch and surface printing enhancements
- TIFF and JPEG device drivers
- TIFF preview images for Encapsulated Postscript

The Application Program Interface enhancements included setting up the compiler location for Windows NT.

MATLAB 5.1 also fixed bugs from earlier releases either reported by customers or found through additional internal testing.

Upgrades to Simulink, Real-Time Workshop, Toolboxes, and Blocksets

MATLAB 5.1 completed the upgrades to the entire set of toolboxes. The Fixed-Point Blockset 1.0.2 was introduced with MATLAB 5.1.

New releases of the following products were also produced with MATLAB 5.1; however, these products were upgraded again in Release 10 (MATLAB 5.2) and in Release 11. The enhancements for these products are reflected in the Release 11 online documentation for each product.

- Simulink
- Real-Time Workshop
- DSP Blockset
- Image Processing Toolbox
- Symbolic Math Toolboxes
- Communications Toolbox

New Products

In addition to these toolboxes, two new products were introduced with MATLAB 5.1:

- Stateflow 1.0
- Mapping Toolbox 1.0

Stateflow and the Mapping Toolbox are described in more detail starting on page 3-12.

Language and Development Environment Enhancements

find Returns Empty Matrix

The `find` function returns an empty matrix if nothing is found. Previously it returned `[0,1]`.

Multibyte Character Support

On the PC, MATLAB 5.1 added support for multibyte characters (including Kanji) for data.

This feature allows you to use multibyte characters in MATLAB strings. You can also use multibyte characters in Handle Graphics property values and Simulink blocks.

Note that you *cannot* use multibyte characters in variable, file, or function names. Also, multibyte text may not be machine independent.

Removal of Microsoft Windows TCP/IP Issues

MATLAB 5.0 for Microsoft Windows 95 required the use of TCP/IP networking software even for non-networked installations. For MATLAB 5.1 this requirement was removed. The portions of the MATLAB user interface that depended upon TCP/IP were recoded to use ActiveX.

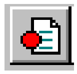
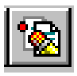
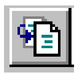
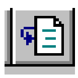
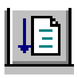
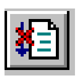
Notebook Support for Office 97

MATLAB 5.1 provided Notebook support for Microsoft Office 97.

Note The MATLAB 5.1 Notebook worked with Windows NT with Microsoft Office 97. However, for Windows 95, due to an Office 97 problem, you may experience problems printing a Notebook document that includes an imported graphic. See “OFF97: Imported EMF Files Are Not Printed Correctly” in the online Microsoft Knowledge Base for details.

PC Editor/Debugger

For MATLAB 5.1 the PC Editor/Debugger provided new debugging icons on the toolbar. The debugging operations are the same as for MATLAB 5.0. The new debugging icons on the toolbar were:

Toolbar Button	Description	Equivalent Command
	**Set/Clear Breakpoint:** set or clear a breakpoint at the line containing the cursor.	`dbstop/` `dbclear`
	**Clear All Breakpoints:** clear all breakpoints that are currently set.	`dbclear all`
	**Step In:** execute the current line of the M-file and if the line is a call to another function, step into that function.	`dbstep in`
	**Single Step:** execute the current line of the M-file.	`dbstep`
	**Continue:** continue execution of M-file until completion or until another breakpoint is encountered.	`dbcont`
	**Quit Debugging:** exit the debugging state.	`dbquit`

Handle Graphics Enhancements

MATLAB 5.1 provided some new Handle Graphics functions.

Scatter Plot Functions Added

MATLAB added two new functions, `scatter` and `scatter3`, which enable you to create two-dimensional and three-dimensional scatter plots. Each function allows you to specify the style, size, and color of the marks used to create the scatter diagrams.

See the online *MATLAB Function Reference* for more information about these functions.

X-Windows Support for uisetcolor

The `uisetcolor` function is supported on X-Windows systems (UNIX).

Previously Undocumented Functions

These two functions existed in MATLAB 5.0, with command line help, but were not documented in the MATLAB 5.0 online *Function Reference*; they are documented in the MATLAB 5.2 online *Function Reference*:

- `pagedlg` – Dialog box to set page layout properties for printing Figures.
- `printdlg` – Dialog box to manage printing of Figures.

Printing Patches and Surfaces

MATLAB 5.1 added support for printing texture-mapped patches and surfaces (this did not work in Version 5.0).

Also, printing interpolated patches and surfaces is more efficient than in Version 5.0.

TIFF and JPEG Device Drivers

MATLAB 5.1 added new built-in device drivers for producing Tagged Image File Format (TIFF) and Joint Photographic Experts Group (JPEG) graphics files from MATLAB figures. These drivers are available on all platforms.

This table summarizes the command-line switches for these drivers.

Device	Description
`–dtiff`	TIFF with packbit compression
`–dtiffnocompression`	TIFF with no compression
`–djpeg`	Baseline JPEG, quality level 75
`–djpeg`*`number`*	Baseline JPEG, quality level specified by *`number`*

Note These drivers work with MATLAB figures only. You cannot use these drivers to print Simulink models.

This section summarizes how to use these drivers with the `print` command.

TIFF

To produce a TIFF file from a MATLAB figure, use the `–dtiff` switch. For example, this command produces a TIFF file named `newplot.tif` from the current figure

```
print –dtiff newplot.tif
```

You can use the `–r` option in conjunction with the `–dtiff` switch to specify the resolution of the output. For example,

```
print –dtiff –r100 newplot.tif
```

If you do not specify the resolution, MATLAB uses the default resolution of 150 dots per inch.

Note that you must specify a filename because TIFF files cannot be sent directly to a printer. If you omit the filename, MATLAB assigns the file a name, such as `figure1.tif`. If you specify a filename that does not include the `.tif` extension, MATLAB appends the extension automatically.

The TIFF files that MATLAB produces are 24-bit truecolor bitmaps. MATLAB renders these graphics using the Z-buffer renderer, regardless of the setting of the figure `Renderer` property. If you use the `–painters` switch with the `print` command, the switch is ignored.

Compression

The TIFF output produced by `–dtiff` uses packbit compression, a lossless compression scheme that is supported by virtually all applications that can import TIFF graphics. If you need to import a TIFF file into an application that does not read packbit-compressed TIFF, use the `–dtiffnocompression` switch to produce an uncompressed TIFF file. (You can abbreviate this switch to `–dtiffn`.) For example,

```
print –dtiffn –r100 newplot.tif
```

An uncompressed TIFF file is often much larger than the same file compressed. For certain plots, the uncompressed file may be more than 10 times the size of the compressed file. (The actual ratio will vary. The size of an uncompressed file depends only on the resolution and the width and height values in the `PaperPosition` figure property; the size of the compressed file also depends on the content of the figure.)

JPEG

To produce a JPEG file from a MATLAB figure, use the `–djpeg` switch. For example, this command produces a JPEG file named `newplot.jpg` from the current figure

```
print –djpeg newplot.jpg
```

You can you use the `–r` option in conjunction with the `–djpeg` switch to specify the resolution of the output. For example,

```
print –djpeg –r100 newplot.jpg
```

If you do not specify the resolution, MATLAB uses the default resolution of 150 dots per inch.

Note that you must specify a filename because JPEG files cannot be sent directly to a printer. If you omit the filename, MATLAB assigns the file a name such as `figure1.jpg`. If you specify a filename that does not include the `.jpg` extension, MATLAB appends the extension automatically.

The JPEG files that MATLAB produces are 24-bit truecolor bitmaps. MATLAB renders these graphics using the Z-buffer renderer, regardless of the setting of the figure `Renderer` property. If you use the `–painters` switch with the `print` command, the switch is ignored.

Compression

JPEG files use a lossy compression scheme that compresses files dramatically with relatively little loss of information. This scheme enables you to make tradeoffs between file size and quality, by specifying a quality level between 0 (minimum quality, maximum compression) and 100 (maximum quality, minimum compression). By default, `–djpeg` uses a quality level of 75; however, you can use a different level by appending the value to the device name. For example, this command produces a JPEG file with a quality level of 50

```
print –djpeg50 –r100 newplot.jpg
```

Even at the highest quality level, JPEG files are often highly compressed. In fact, depending on the figure, a JPEG file with a quality level of 100 may be considerably smaller than a packbit-compressed TIFF file of the same figure.

TIFF Preview Images for Encapsulated PostScript

MATLAB 5.1 introduced support for TIFF preview images for Encapsulated PostScript (EPS) files. To produce a TIFF preview, use the `–tiff` switch. For example, this command creates an EPS file called `newplot.eps` that contains a TIFF preview

```
print –deps –tiff newplot.eps
```

The preview image has a resolution of 72 dots per inch, and the colors in the preview match the colors in the EPS file. MATLAB creates the EPS with a loose bounding box (i.e., white space around the figure), so that the size and position of the preview image match the EPS. There may be some differences between the EPS and the TIFF preview because the preview is always rendered using Z-buffer, while the EPS may be rendered with painter's algorithm.

The `–tiff` switch works on all platforms; you can view the resulting preview image within any application that can display TIFF graphics.

API Enhancements for Windows NT

Setting Up the Compiler Location

MATLAB 5.1 provided a new switch for the `mex` script. The switch, `setup`, allows you to configure the default options file, `mexopts.bat`, for your system C compiler. This eliminates the need to reinstall MATLAB if you change compilers for your environment.

You can run the `setup` option from either the MATLAB or DOS command prompt, and it can be called anytime to configure the options file.

Executing the `setup` option presents a list of compilers whose options files are currently shipped in the `bin` subdirectory of MATLAB. This example shows how to select the Microsoft Visual C++ compiler.

```
C:\mex –setup
C compilers
[1] Microsoft Visual C++
[2] Borland C/C++
[3] Watcom C/C++

Fortran compilers
[4] Microsoft Powerstation

[0] None

compiler: 1
```

If the selected compiler has more than one options file (because of more than one version of the compiler), you are asked for a specific version. For example,

```
Which version
[1] 4.x
[2] 5.x
version: 1
```

Finally, you are asked to enter the location of your compiler.

```
Please enter the location of your C compiler c:\msdev
```

Stateflow

Addition to the Simulink Modeling Environment

Stateflow was introduced with MATLAB 5.1, adding to the Simulink modeling and simulation environment. A graphical tool for designing complex control and supervisory logic systems, Stateflow allows you to model and simulate the behavior of complex reactive, event-based systems based on finite state machine theory. Stateflow lets you add event-driven elements of a system to Simulink continuous or discrete modeling for a single, closed-loop simulation.

Stateflow represents an evolution from finite state machine theory by adding several major improvements, including hierarchy, parallelism, junctions, and history. These changes enable Stateflow to make practical use of finite state machine theory with realistic application to control systems.

A major benefit of Stateflow is its seamless point-and-click interface to Simulink. The control behavior that Stateflow models provides an ideal complement to the algorithmic behavior modeled in Simulink. In Simulink, you develop your model of continuous- and discrete-time dynamic systems using its graphical, block diagram environment. Then you drag and drop the blocks that represent Stateflow diagrams directly into your Simulink model to add event-driven behavior to Simulink simulations.

Applications for Stateflow include developing the control logic in embedded systems for electronic and mechanical systems found in automobiles, aircraft, telecommunications systems, computer peripherals, office automation equipment, and medical instrumentation.

Stateflow works with the latest versions of Simulink and Real-Time Workshop to offer an integrated environment for modeling, simulating, and prototyping real-time embedded systems applications.

Stateflow Coder

Stateflow provides automatic C-code generation through the optional Stateflow Coder. C code generated by Stateflow Coder can be used independently or integrated with code from Real-Time Workshop. Thus, Simulink, Stateflow, and Real-Time Workshop are integrated from the design and modeling phase through the code generation stage. The generated code can be executed for rapid prototyping, hardware-in-the-loop testing, or for stand-alone simulations.

Mapping Toolbox

The Mapping Toolbox was introduced with MATLAB 5.2, providing a toolbox for geographic display and cartographic analysis. The toolbox gives you the ability to plot geographically based information as easily as any other type of data that you can plot in MATLAB. Both vector and matrix map data can be displayed, manipulated, and analyzed. The toolbox manages the projection, clipping, and trimming of the data automatically for you, even if you change the projection.

The Mapping Toolbox provides more than 60 map projections. There are extensive geographic analysis functions, such as computations of distance, tracks, great and small circles, intersections, and navigation functions. These computations can be made for a spherical body or can make use of spheroidal models of the earth and other planets when more accuracy is required. Utility functions allow you to convert easily among different time, distance, and angle units.

The toolbox provides a number of global map data sets and allows you to import detailed data from government sources over the Internet and on CD-ROM. These data sets include the Digital Chart of the World, Tiger/Line files, and Digital Elevation Models of the world and the United States.

In addition to the command-line functions, the toolbox also provides an extensive suite of GUIs for accessing the toolbox functionality. These GUIs allow you to manage data interactively, plot it, modify the display, make measurements, and generate geographic data like tracks and circles. The GUIs are available as an integrated set and also are available individually.

4

Upgrading to Release 11

Migrating to Release 11 (MATLAB 5.3)

MATLAB Migration

It is useful to introduce two terms in discussing this migration. The first step in converting your code to MATLAB 5.3 is to make it MATLAB 5.3 *compatible*. This involves a rather short list of possible changes that let your M-files run under MATLAB 5.3. The second step is to make it MATLAB 5.3 *compliant*. This involves 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 5.3 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 must change to make your M-files MATLAB 5.3 compatible. Most of these are in the language area.

There are a somewhat larger number of things you can do (but don't have to) to make your M-files fully MATLAB 5.3 compliant. To help you gradually make your code compliant, MATLAB 5.3 displays warning messages when you use functions that are obsolete, even though they still work correctly.

Roadmap for Different Migration Routes

The sections of this chapter that you need to read depend on the version of MATLAB from which you are upgrading.

If You Are Upgrading to MATLAB 5.3 from...	Read These Sections or Document ...
MATLAB 5.2	"Upgrading From MATLAB 5.2 to MATLAB 5.3"
MATLAB 5.1	"Upgrading from MATLAB 5.1 to MATLAB 5.3" "Upgrading From MATLAB 5.2 to MATLAB 5.3"
MATLAB 5.0	"Upgrading from MATLAB 5.0 to MATLAB 5.3" "Upgrading from MATLAB 5.1 to MATLAB 5.3" "Upgrading From MATLAB 5.2 to MATLAB 5.3"
MATLAB 4	The separate online (PDF and HTML) document *Upgrading from MATLAB 4 to MATLAB 5.0*

Note The last section in this chapter discusses upgrade issues for Simulink, toolboxes, and blocksets.

Upgrading From MATLAB 5.2 to MATLAB 5.3

This section describes differences between Release 11 and Release 10 that may require code changes to your Release 10 code.

Language Issues

pcode

Prior to MATLAB 5.3, by default `pcode` put all its `.p` files, including methods and private functions, in the current directory.

That approach resulted in losing the scope directories (private and class directories) for the function and had the potential to have a method from one directory overwrite the method from another directory.

In MATLAB 5.3, by default `pcode` puts its `.p` files for methods and functions in a corresponding class or private directory (created in the current directory if those directories don't already exist).

If you have any pre-5.3 code that relies on the `.p` file being in the current directory, and you generate new P-code for the method, the `.p` file will not be where the pre-5.3 code expected it.

Date Functions Need pivotyear Parameter

With MATLAB 5.3, the date functions `datenum`, `datestr`, and `datevec` include a new calling sequence that allows a pivot year specification to override the default. For example, here's the new calling sequence for `datevec`:

```
[...] = datevec(t, pivotyear)
```

This new call uses the pivot year instead of the current year minus 50 years.

Whether you update your applications that use these date functions depends on the nature of your data and the timeframe when you plan to use the application:

- If the dates of all your data are always going to be within 50 years of the current year, then you do not need to specify the pivot year.
- If your data includes dates that are older than 50 years before the current year, you will have to specify a pivot year of 1900 to produce the same results as with 5.2.

Sparse scalar Expansion

`A(:,:) = scalar` was incorrectly producing the result `A = scalar` when it should have been changing all the elements of `A` to the scalar value (which is what `A(1:end,1:end) = scalar` does).

getfield Must Use a 1-by-1 Structure

In MATLAB 5.3, the `getfield` function produces an error message if you use other than a 1-by-1 structure. In MATLAB 5.2, the commands

```
a(1).b = 1;
a(2).b = 3;
a(2).b.c = 4;
a(1).b.c = 2;
d = [getfield([a.b], 'c')];
```

successfully returned the two element vector

```
d =
     2    4
```

but in MATLAB 5.3, the same code would lead to `getfield` returning an error message.

Syntax Change for dlmread

The `dlmread` syntax has changed. You should no longer use the `range` argument together with the row and column offsets as you could in previous versions of MATLAB (the row and column offsets actually were not used in pre-5.3 MATLAB, if you specified a `range` argument). Using this calling sequence now produces a warning message:

```
M = dlmread(filename,delimiter,row_offset,column_offset,range)
```

To use the `range` argument, use this new calling sequence:

```
M = dlmread (filename,delimiter,range)
```

Behavior of linspace and logspace Now the Same as with MATLAB 5.1

In MATLAB 5.3, `linspace` and `logspace` now handle `NAN`s, `Inf`s, and complex vectors in the same manner as they did for MATLAB 5.1 and earlier versions. This eliminates an inconsistency in how `linspace` and `logspace` handled `NAN`s, `Inf`s, and complex vectors in MATLAB 5.2, as compared to previous MATLAB releases.

Name Changes

These MATLAB functions have new names and calling sequences to support new functionality.

Old Function Name	New Function Name
`fmin`	`fminbnd`
`fmins`	`fminsearch`
`nnls`	`lsqnonneg`

Note that if you have older M-files that use the old names and calling sequences, these calls will generally continue to work. However, the older functions may be removed from MATLAB in future releases, so it is a good idea to revise your code now to use the new names and calling sequences.

Method Search Order Changed

MATLAB now calls converter methods on the path before calling a constructor. Prior to 5.3, MATLAB called constructors before calling methods on the path.

One visible effect of this is that in the case where

```
@double/ss.m
@ss/ss.m
```

exist, MATLAB calls `@double/ss.m` instead of `@ss/ss.m` (i.e., the previously shadowed converter functions become visible).

Change to Subscripting for Objects

In MATLAB 5.3, subscripting syntax is dispatched differently from how it was dispatched in previous releases. Within a method, the syntaxes

`a(i)`, `a(i,j)`, etc.
`a{i}`, `a{i,j}`, etc.
`a(i).name`, `a.name`, `a(i.j).name`, etc.

now use the built-in `subsref` or `subsasgn` method if the type of a doesn't match the class directory and no overloaded `subsref` is defined for the object `a`. This change only affects child objects within parent methods.

Prior to 5.3, use of the `subsref` or `subsasgn` syntax called the parent's `subsref` method recursively for a child object. Now the syntax uses the built-in method.

For example, for the following

```
@parent/get.m
function b = get(a,i)
b = a(i);

@parent/subsref.m
function r = subsref(a,s)
r = a(1).dat;
```

when no `subsref` existed for `@child`

```
c = child(5);
get(c,1)
```

used to call the parent's `subsref` method inside `@parent/get.m`. MATLAB now uses the built-in method.

Within parent methods where you want the parent's `subsref` to be used, call the parent's `subsref` directly (using the function syntax).

Use clear classes to Clear the Class Definition Table

To clear the class definition table so that MATLAB picks up changes in an object's field definition, use

```
clear classes
```

In previous releases, using

```
clear all
```

would often clear the class definition table, but not always in the proper manner. Replace `clear all` with `clear classes` when you want to clear the class definition table safely.

Changes to legend

MATLAB 5.3 enhances `legend` to:

- Support multiline labels, allowing you to wrap long labels
- Keep the legend the same size as it is displayed on screen when you print a figure
- Integrate with the Plot Editor

To support these enhancements, MATLAB 5.3 treats the legend text as one text object, grouping all the text together. So, if your pre-5.3 code manipulates individual handles within a legend, to specify properties such as font size or font color, that code will probably no longer work in MATLAB 5.3.

Upgrading from MATLAB 5.1 to MATLAB 5.3

This section describes compatibility issues involved in upgrading from MATLAB 5.1 to MATLAB 5.2.

Note If you are upgrading from MATLAB 5.1 to MATLAB 5.3, in addition to reading this section, you should read the previous section, called "Upgrading From MATLAB 5.2 to MATLAB 5.3."

Use of P-Code Between MATLAB Versions

You cannot use Version 5.2 P-code in a pre-5.2 P-code application. You can use pre-5.2 P-code in a Version 5.2 P-code application.

If you want to distribute an application to users who might be running a different version of MATLAB than the one in which you are writing the application, you should use M-files instead of P-code.

Colon Expressions with Floating-Point Numbers

Values produced in colon (:) expressions may vary between MATLAB 5.2 and pre-5.0 versions of MATLAB, if you are doing an exact comparison of floating-point numbers.

For floating-point numbers, you should use tolerance-based comparisons (`eps`), not exact comparisons. (Use exact comparisons only for integers.)

Warning When Using == with an Empty Matrix

The expression `A == []` produces `0` or `1` (as it did in MATLAB 4), and MATLAB issues the following warning message when this expression is used:

```
Warning: X == [] is technically incorrect. Use isempty(X) instead.
```

This warning is issued in anticipation of future versions of MATLAB, which will return an empty matrix, `[]`, for this expression.

Invoking the Path Editor from the Command Line

To invoke the MATLAB path editor from the command line, issue the `pathtool` command. In previous releases on various platforms the `pathedit` and `editpath` commands also invoked the path editor, but the command that works on all platforms for Version 5.2 is `pathtool`.

Frame Uicontrols and Stacking Order

Frames are opaque, not transparent, so the order you define Uicontrols is important in determining whether Uicontrols within a frame are covered by the frame or are visible. *Stacking order* determines the order objects are drawn: objects defined first are drawn first; objects defined later are drawn over existing objects. If you use frames to enclose objects, you must define the frames before you define the objects.

Before MATLAB 5.2, frames were always drawn below other Uicontrols on Microsoft Windows applications regardless of the order they were created.

If you use MATLAB on UNIX computers, this change does not affect you. If, however, you use MATLAB on Microsoft Windows, stacking order affects any applications that define frames *after* they define objects contained within the frames. To ensure that frames are drawn *below* other objects, either:

- Revise the M-files by altering the order in which these objects are defined. Create frames before creating the objects contained in the frames.
- Modify the stacking order to ensure that objects within the frames are visible. For example, these statements define a push button, a check box, and a frame, then alter the stacking order for the Figure (`position` vectors are defined by `pbpos`, `cbpos`, and `fpos` to simplify the code):

```
hpush  = uicontrol('Style','pushbutton','Position',pbpos);
hcheck = uicontrol('Style','checkbox','Position',cbpos);
hframe = uicontrol('Style','frame','Position',fpos);
% change stacking order to put frame on bottom
% gcf is the current figure
stackvec(1)=hpush; stackvec(2)=hcheck; stackvec(3)=hframe;
set(gcf,'Children',stackvec)
```

- Issue a `system_dependent` command to force frames to be drawn below other objects. The form of this command is:

  ```
  system_dependent('ForceFramesOnBottom','on')
  ```

  Note that the `ForceFramesOnBottom` string is case sensitive. Issue the command before running the application. When you issue the command, MATLAB issues a warning indicating that frames will be inserted below other objects. To suppress the warning message for just this command, include these statements in your M-file or your `startup.m` file:

  ```
  warning off
  system_dependent('ForceFramesOnBottom','on')
  warning on
  ```

  You should use these commands only until you have had a chance to correct the M-files. The first two solutions are preferable to this solution; this solution is provided to ease the transition for users who were not aware that the Microsoft Windows behavior was inconsistent with stacking order rules that applied to all other Handle Graphics objects.

  You can turn off this behavior using this statement:

  ```
  system_dependent('ForceFramesOnBottom','off')
  ```

Change to clear Behavior

The `clear` function does not remove an M-file from the MATLAB workspace if that M-file is locked with the `mlock` function, introduced in MATLAB 5.2.

Although pre-5.2 code does not use `mlock`, if that code is modified to use `mlock`, `clear` will not behave as it did in previous versions of MATLAB (i.e., it will not be guaranteed to clear all M-files in the workspace). To unlock an M-file, use `munlock`.

try, catch, and persistent Are Now Keywords

You can no longer use `try`, `catch`, and `persistent` as variable names in MATLAB. In previous releases MATLAB did not treat these as keywords.

Matrix Assignment

In pre-5.2, for

```
A(:) = b
```

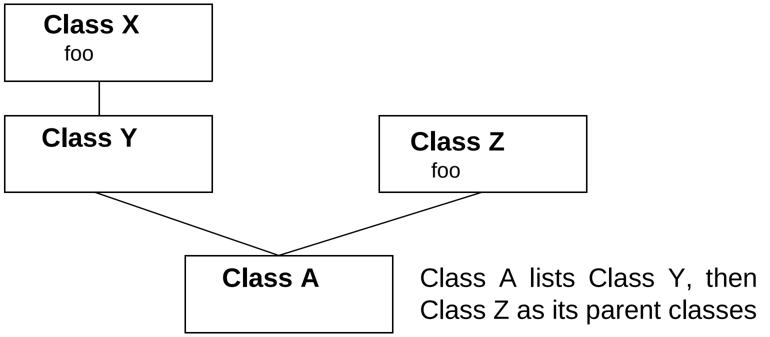
where `b` is a scalar or vector, the resulting type was the type of `b`. Starting with MATLAB 5.2, the resulting type is the type of `A`. So, if `A` is a `uint8` array to begin with, and `b` is a `double`, the result is that `A` is still a `uint8`.

This change to preserve `A`'s type was made to ensure consistent indexing behavior.

Change to Method Search Order

When an object inherits from two different classes, MATLAB performs a depth-first search when a method is called (this is how the method search was documented as working in pre-5.2 versions). MATLAB exhausts the search in one parent, then goes up the class hierarchy for each class from which the object inherits (from left to right, as appears in the class definition). In previous releases, MATLAB actually performed a modified breadth-first search.

In this hierarchy:



If function `foo` is called:

- In MATLAB 5.2 and later, Class X's `foo` would be invoked.
- In MATLAB 5.0 or 5.1, Class Z's `foo` would be invoked.

Changes to legend

The second output argument for `legend` is no longer `m`-by-`2`. Instead, it is a column containing line, patch, and text object handles, in no particular order.

PC-Specific Changes

Change to clc Command

In MATLAB 5.2, the `clc` command produces the same result as using the **Edit** menu item **Clear Sessions**. Thus, after you issue `clc`, you can no longer scroll back to see the previous contents of the Command Window (as you could in earlier versions of MATLAB).

However, you can use the up arrow to see the history of the commands, one at a time.

Change to cd Command

In MATLAB 5.2, if you `cd` from one drive to another for your working directory, the `cd` command does not retain any subdirectory part of the path if you `cd` back to the initial drive.

For example, if you first issue a `cd` command such as

```
cd C:\MyApps
```

and then issue

```
cd D:\MyMatlabDir
cd C:
pwd
```

you will see

```
C:\
```

In earlier versions of MATLAB, if you issued the same commands as shown above, you saw

```
C:\MyApps
```

API Memory Management Compatibility Issue

To address performance issues, the internal MATLAB memory management model has been changed somewhat. These changes support future enhancements to the MEX-file API.

With this release, MATLAB now implicitly calls `mxDestroyArray`, the `mxArray` destructor, at the end of a MEX-file's execution on any `mxArrays` that are not returned in the left-hand side list (`plhs[]`). You are now warned if MATLAB detects any misconstructed or improperly destroyed `mxArrays`.

We highly recommend that you fix code in your MEX-files that produces any of the warnings discussed in the following sections. For additional information, see the "Memory Management" section in Chapter 3 of the *Application Program Interface Guide*.

The rest of this discussion describes situations in which you would receive such warning messages. The discussion of each situation includes an example and a solution.

Improperly Destroying an mxArray

You cannot use `mxFree` to destroy an `mxArray`.

Warning

```
Warning:  You are attempting to call mxFree on a <class-id> array.
The destructor for mxArrays is mxDestroyArray; please call this
instead. MATLAB will attempt to fix the problem and continue, but
this will result in memory faults in future releases.
```

Note In MATLAB 5.2, these warnings are enabled by default for compatibility reasons. They can be disabled with the command

```
feature('MEXFileCompat',0)
```

Disabling the code that detects and fixes these error conditions may slightly improve performance, but will cause serious problems if your MEX-file actually causes any of these errors.

Example That Causes Warning

```
    mxArray *temp = mxCreateDoubleMatrix(1,1,mxREAL);
...
    mxFree(temp);  /* INCORRECT */
```

mxFree does not destroy the array object. This operation frees the structure header associated with the array, but MATLAB will still operate as if the array object needs to be destroyed. Thus MATLAB will try to destroy the array object, and in the process, attempt to free its structure header again.

Solution

Call mxDestroyArray instead:

```
mxDestroyArray(temp);  /* CORRECT */
```

Incorrectly Constructing a Cell or Structure mxArray

You cannot call mxSetCell or mxSetField variants with prhs[] as the member array.

Warning

```
Warning: You are attempting to use an array from another scope
(most likely an input argument) as a member of a cell array or
structure. You need to make a copy of the array first. MATLAB will
attempt to fix the problem and continue, but this will result in
memory faults in future releases.
```

Example That Causes Warning

```
myfunction('hello')
/* myfunction is the name of your MEX-file and your code */
/* contains the following:    */

    mxArray *temp = mxCreateCellMatrix(1,1);
...
    mxSetCell(temp, 0, prhs[0]);  /* INCORRECT */
```

When the MEX-file returns, MATLAB will destroy the entire cell array. Since this includes the members of the cell, this will implicitly destroy the MEX-file's input arguments. This can cause several strange results, generally having to do with the corruption of the caller's workspace, if the right-hand side argument used is a temporary array (i.e., a literal or the result of an expression).

Solution

Make a copy of the right-hand side argument with `mxDuplicateArray` and use that copy as the argument to `mxSetCell` (or `mxSetField` variants); for example:

```
mxSetCell(temp, 0, mxDuplicateArray(prhs[0]));  /* CORRECT */
```

Creating a Temporary mxArray with Improper Data

You cannot call `mxDestroyArray` on an `mxArray` whose data was not allocated by an API routine.

Warning

```
Warning: You have attempted to point the data of an array to a
block of memory not allocated through the MATLAB API. MATLAB will
attempt to fix the problem and continue, but this will result in
memory faults in future releases.
```

Example That Causes Warning

If you call `mxSetPr`, `mxSetPi`, `mxSetData`, or `mxSetImagData` with memory as the intended data block (second argument) and that memory was not allocated by `mxCalloc`, `mxMalloc`, or `mxRealloc`, as shown below

```
    mxArray *temp = mxCreateDoubleMatrix(0,0,mxREAL);
    double data[5] = {1,2,3,4,5};
...
    mxSetM(temp,1); mxSetN(temp,5); mxSetPr(temp, data);
    /* INCORRECT */
```

then when the MEX-file returns, MATLAB will attempt to free the pointer to real data and the pointer to imaginary data (if any). Thus MATLAB will attempt to free memory, in this example, from the program stack. This will cause the above warning when MATLAB attempts to reconcile its consistency checking information.

Solution

Rather than use mxSetPr to set the data pointer, instead create the mxArray with the right size and use memcpy to copy the stack data into the buffer returned by mxGetPr:

```
     mxArray *temp = mxCreateDoubleMatrix(1,5,mxREAL);
     double data[5] = {1,2,3,4,5};
...
     memcpy(mxGetPr(temp), data, 5*sizeof(double));  /* CORRECT */
```

Potential Memory Leaks

Prior to Version 5.2, if you created an mxArray using one of the API creation routines and then you overwrote the pointer to the data using mxSetPr, MATLAB would still free the original memory. This is no longer the case.

For example

```
pr = mxCalloc(5*5, sizeof(double));
... <load data into pr>
plhs[0] = mxCreateDoubleMatrix(5,5,mxREAL);
mxSetPr(plhs[0], pr);  /* INCORRECT */
```

will now leak 5*5*8 bytes of memory, where 8 bytes is the size of a double.

You can avoid that memory leak by changing the code

```
plhs[0] = mxCreateDoubleMatrix(5,5,mxREAL);
pr = mxGetPr(plhs[0]);
... <load data into pr>
```

or alternatively:

```
pr = mxCalloc(5*5, sizeof(double));
... <load data into pr>
plhs[0] = mxCreateDoubleMatrix(5,5,mxREAL);
mxFree(mxGetPr(plhs[0]));
mxSetPr(plhs[0], pr);
```

Note that the first solution is more efficient.

Similar memory leaks can also occur when using mxSetPi, mxSetData, mxSetImagData, mxSetIr, or mxSetJc. You can address this issue as shown above to avoid such memory leaks.

Recommendation: MEX-Files Should Destroy Their Own Temporary Arrays

In general, we recommend that MEX-files destroy their own temporary arrays and clean up their own temporary memory. All inconsistent `mxArrays` except those returned in the left-hand side list and the return from the `mxGetArrayPtr` may be safely destroyed. This approach is consistent with other MATLAB API applications (i.e., MAT-file applications, Engine applications, and MATLAB Compiler-generated applications).

Upgrading from MATLAB 5.0 to MATLAB 5.3

This table describes some changes you can make to your code to eliminate error messages and warnings due to incompatible and noncompliant statements in MATLAB 5.0 code that you are upgrading to MATLAB 5.1.

Note If you are upgrading from MATLAB 5.0 to MATLAB 5.3, in addition to this section, you should read the previous two sections, called “Upgrading From MATLAB 5.2 to MATLAB 5.3“and “Upgrading from MATLAB 5.1 to MATLAB 5.3.”

Function	Change	Action
`find`	`find` was modified for sparse row vectors. `find(sparse_row)` was a column in MATLAB 4 and MATLAB 5.0. In 5.1 it produces a row when the input is a row. All other cases still return columns.	Update code.
`lasterr`	In MATLAB 5.1, `lasterr` doesn’t contain the `???` that prints out when you get an error. Some types of errors in MATLAB 5.0 erroneously contained `???`.	None required.
`plot`	In MATLAB 5.0, `plot` incorrectly accepted arrays with more than two dimensions, but treated them as two-dimensional arrays. In 5.1, this causes an error.	Do not use arrays of more than two dimensions as arguments for `plot`.

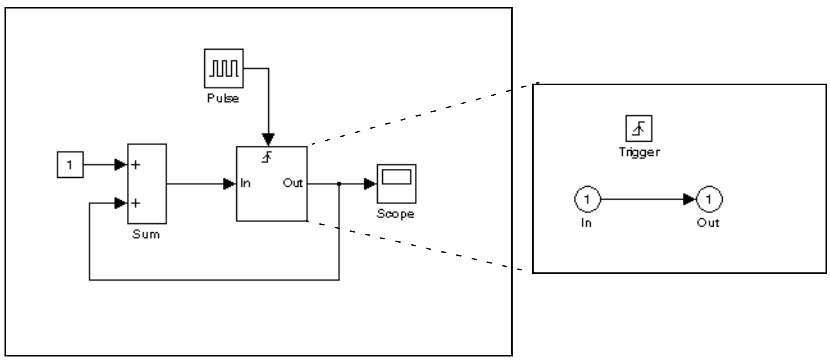
Function	Change	Action
Set functions: `intersect`, `setdiff`, `setxor`, `union`, `unique`	These functions now error out if the inputs aren't vectors, and you aren't using the `rows` flag.	Update code if required.
`size`	An empty string created within a MEX-file is now of size 0,0, consistent with the rest of MATLAB. (This change occurred with MATLAB 5.1.)	Update code accordingly.

Upgrading Simulink, Toolboxes, and Blocksets

Upgrading to Simulink 3.0 From Simulink 2.1

Starting in Version 2.2 (Release 10), Simulink changed the way it treats direct-feedthrough loops containing triggered subsystems. Before Version 2.2, Simulink treated such loops as algebraic and attempted to use an algebraic solver to solve them. Beginning in Version 2.2, Simulink treats direct feedthrough loops containing triggered subsystems as nonalgebraic, thereby allowing use of nonalgebraic solvers.

For example, Simulink 2.1 unsuccessfully attempts to simulate the following counter model, using an algebraic solver.



In particular, it stops the simulation after detecting discontinuities in the algebraic solution. Simulink 2.2 and subsequent versions successfully use a variable-step discrete solver to simulate the same model.

Upgrading to DSP Blockset 3.0 and Communications Toolbox 1.4

The New Complex Data Format

Versions 1.0 through 2.2 of the DSP Blockset and Versions 1.0 through 1.3 of the Communications Toolbox provided complex data capability by creating a double-length real vector whose first half contained the real components of the vector's elements, and whose second half contained the imaginary components of the vector's elements. This format was generally only recognized by other blocks in the DSP Blockset and Communications Toolbox that had the complex identifier (*) at the appropriate port.

Simulink 3.0 provides an *intrinsic* complex data type (see Chapter 8 of *Using Simulink*) that supplants the earlier DSP Blockset and Communications Toolbox implementation. Complex data in Simulink is now handled very much the same as complex data in MATLAB. Double-length vectors are no longer used to convey complex information.

Your existing complex-data models will continue to work in Simulink 3.0 since the complex capability in older models is implemented solely by the blocks themselves. The older blocks also continue to be available to you by typing `dsplib(2)` and `commlib(1)` at the command line. However, 3.0 blocks cannot be directly intermixed with the older blocks in complex-data models. The next section explains options for upgrading.

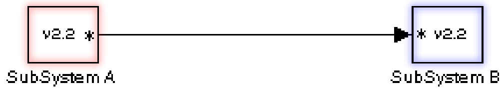
Why You Need to Update Your Models to Use the New Complex Data Format

The new complex data format introduced by Simulink 3.0 is not compatible with the old complex data format implemented in the DSP Blockset 2.2 and the Communications Toolbox 1.3. Mixing new (i.e., Simulink 3.0 and its associated toolboxes and blocksets) and old (2.2) blocks in a simulation using complex data can easily lead to errors and incorrect results.

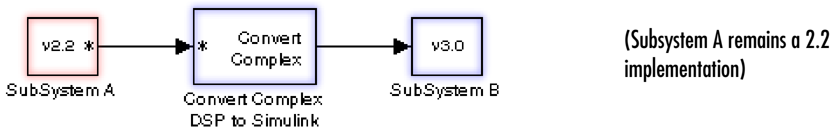
For small models, we recommend completely replacing *all* old blocks in the complex portion of the model with their new equivalents when you begin incorporating new blocks into the model.

For larger models, where the required block substitutions might be extensive, you can use the Convert Complex DSP to Simulink block to convert complex data in the old format to complex data in the new format. To convert back to the old format, use the Convert Complex Simulink to DSP block.

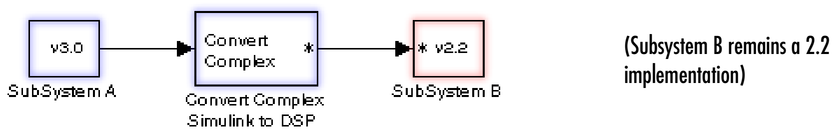
The figure below shows how you can use these two convertor blocks to migrate part of a complex-data model to the new complex format while letting other components continue to use the old complex-data format.



Existing 2.2 complex-data model



Subsystem B upgraded to 3.0 complex-data format



Subsystem A upgraded to 3.0 complex-data format

The convertor blocks are only needed for interfacing new blocks to the *complex-data* section of an older model. New blocks can be added to *real-data* sections of older models without any data format conversion.

The Convert Complex DSP to Simulink and Convert Complex Simulink to DSP blocks are provided to facilitate the transition from 2.2 to 3.0 by allowing you to upgrade your models incrementally, as convenient. Ultimately, of course, you should try to migrate all of your models to the 3.0 complex data format.

Note Within a section of model that uses the old complex format, you should continue to use the complex port identifier (∗) as a guide to wiring blocks. Output ports labeled with the ∗ symbol should only be connected to input ports labeled with the ∗ symbol.

Locating Old Blocks in a Model

In a DSP Blockset model that contains both old and new blocks, you can determine which blocks are linked to a particular version by using the `dsp_links` utility. At the command line, type

```
blks = dsp_links
```

to highlight (in red) blocks in the current model that are linked to old libraries. Blocks that are linked to new libraries are highlighted in blue. The `dsp_links` command analyzes all levels of the model, and also displays a summary in the command window of the number of blocks linked to each library version. The function returns a structure, `blks`, containing separate lists of the old and new blocks in the model.

Similarly, you can use the `comm_links` utility to perform the same analysis for the Communications Toolbox.

Upgrading Optimization Toolbox 2.0

New large-scale algorithms have been incorporated into the Optimization Toolbox 2.0 functions. The new functionality improves the ability of the toolbox to solve large sparse problems. To accommodate this new functionality, many of the function names and calling sequences have changed.

For more information on how to convert your old syntax to the new function calling sequences, see the *Optimization Toolbox User's Guide*.

Upgrading to Fuzzy Logic Toolbox 2.0

Note The Fuzzy Logic Toolbox has not been updated for Release 11. If you have already updated your FIS models from a pre-2.0 version of the Fuzzy Logic Toolbox to the 2.0 level (as part of a Release 10 upgrade), you can ignore this section.

In the Fuzzy Logic Toolbox 2.0, the Fuzzy Inference System (FIS) is represented as a MATLAB structure. A structure (instead of a flat matrix) is now the basic element in constructing a fuzzy logic system. This fundamental change in the way of representing fuzzy logic system makes many details of working with the constructed system easier.

A Fuzzy Inference System that you created with a pre-2.0 version of the Fuzzy Logic Toolbox is still usable in 2.0, if you run the `convertfis` function on it. The `convertfis` function automatically converts pre-2.0 Fuzzy Inference Systems to work with Version 2.0.

Index

G

H

V

W

Y

Z