MATLAB® 7
Desktop Tools and Development Environment

MATLAB®

The MathWorks™
Accelerating the pace of engineering and science
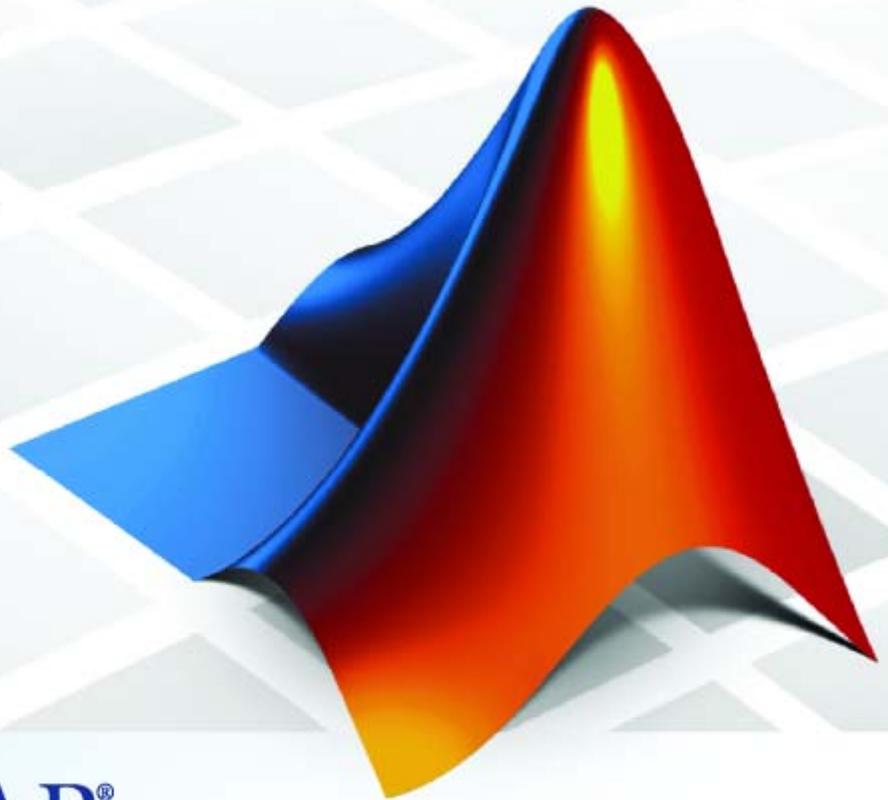
How to Contact The MathWorks

www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.com/contact_TS.html	Technical Support

suggest@mathworks.com	Product enhancement suggestions
bugs@mathworks.com	Bug reports
doc@mathworks.com	Documentation error reports
service@mathworks.com	Order status, license renewals, passcodes
info@mathworks.com	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

MATLAB Desktop Tools and Development Environment

Trademarks

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, SimBiology, SimHydraulics, SimEvents, and xPC TargetBox are registered trademarks and The MathWorks, the L-shaped membrane logo, Embedded MATLAB, and PolySpace are trademarks of The MathWorks, Inc.

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

Patents

The MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

Contents

Startup and Shutdown

1

Desktop

2

Running Functions — Command Window and History

3

Help for Using MATLAB

4

Workspace, Search Path, and File Operations

5

Editing and Debugging M-Files

6

Source Control Interface

9

<div align="right">

Index

</div>

1

Startup and Shutdown

This set of topics includes options for customizing the startup and shutdown.

Starting MATLAB on Windows Platforms (p. 1-2)	Ways to start MATLAB®, including from a desktop icon, or from a file in Windows Explorer. Associating file types with MATLAB.
Starting MATLAB on UNIX Platforms (p. 1-6)	MATLAB startup on UNIX.
Starting MATLAB on Macintosh Platforms (p. 1-7)	Ways to start MATLAB on a Macintosh, including from a MATLAB file type in the Finder.
Startup Directory for MATLAB (p. 1-8)	Ways to change the directory in which MATLAB starts.
Startup Options (p. 1-12)	Instruct MATLAB to perform operations upon startup via a `startup` file or the `matlab` function.
Toolbox Path Caching in MATLAB (p. 1-17)	About the cache file and updating the cache file.
Other Startup Topics (p. 1-21)	Error Log Reporter, passing Perl variables, and calling Java from MATLAB.
Quitting MATLAB (p. 1-23)	End a MATLAB session. Instruct MATLAB to perform specified operations upon shutdown.

Starting MATLAB on Windows Platforms

In this section...
"Starting MATLAB from the Windows Desktop or a DOS Window" on page 1-2
"Starting MATLAB from an M-File or Other File Type in Windows" on page 1-2
"Utility to Change Windows File Associations" on page 1-5
"Changing File Associations for MATLAB from Windows" on page 1-5

Starting MATLAB from the Windows Desktop or a DOS Window

To start MATLAB on a Microsoft Windows platform, select the **Start > Programs > MATLAB > R2007b > MATLAB R2007b**, or double-click the MATLAB R2007b shortcut icon 🔺 on your Windows desktop. The shortcut was automatically created when you installed MATLAB. If you have trouble starting MATLAB, see troubleshooting information in the Installation Guide for Windows.

To start MATLAB from a DOS window, cd to the directory in which you want to start MATLAB and type matlab at the DOS prompt.

After starting MATLAB, the desktop opens. Desktop components that were open when you last shut down MATLAB will be opened on startup. For more information, see Chapter 2, "Desktop".

Starting MATLAB from an M-File or Other File Type in Windows

On Windows platforms, you can start MATLAB from a Windows Explorer window by double-clicking a file with one of these extensions: .fig, .m, .mat, and .mdl. MATLAB starts and opens in an appropriate tool. If MATLAB is already running, the file opens in an appropriate tool in the existing session.

This startup feature is based on your Windows file type associations. When you installed MATLAB for Windows, you specified the file types to

associate with MATLAB. For example, if you accepted the default options, double-clicking an M-file in Windows Explorer opens the file in the MATLAB Editor/Debugger.

Other default options associate MEX-files and P-files with MATLAB in Windows Explorer, which assigns the file types a MATLAB icon. However, double-clicking a file with a `.mex` (`.mexw32` or `.mexw64`), or `.p` extension does not run or open the file in MATLAB.

File Type and Resulting Action

File Type	Result
FIG-file	Opens file in figure window
M-file	Opens file in Editor/Debugger
MAT-file	Opens Import Wizard to load the data into the MATLAB workspace
MDL-file	Opens file in Simulink® model window
MEX-file	Displays MATLAB icon in Windows Explorer
P-file	Displays MATLAB icon in Windows Explorer

Other applications you use can change Windows file associations. For example, Microsoft Access might associate files having a `.mat` extension. Then, double-clicking a MAT-file opens Access rather than MATLAB.

If you double-click a FIG-file, M-file, MAT-file, or MDL-file and it does not open in MATLAB, try this instead:

1 In Windows Explorer, right-click a file with one of the extensions listed in the preceding table, for example, `myfile.mat`.

2 From the context menu, select **Open With**. If MATLAB is one of the choices, select it to open `myfile.mat` in MATLAB. If MATLAB is not one of the choices, you will need to associate the file type with MATLAB using one of these techniques:

- "Utility to Change Windows File Associations" on page 1-5
- "Changing File Associations for MATLAB from Windows" on page 1-5

After associating a file type with MATLAB, you can open other applications using that file type via the context menu. For example, right-click `myfile.mat`, and from the context menu, select **Open With**. Microsoft Access will be one of the options.
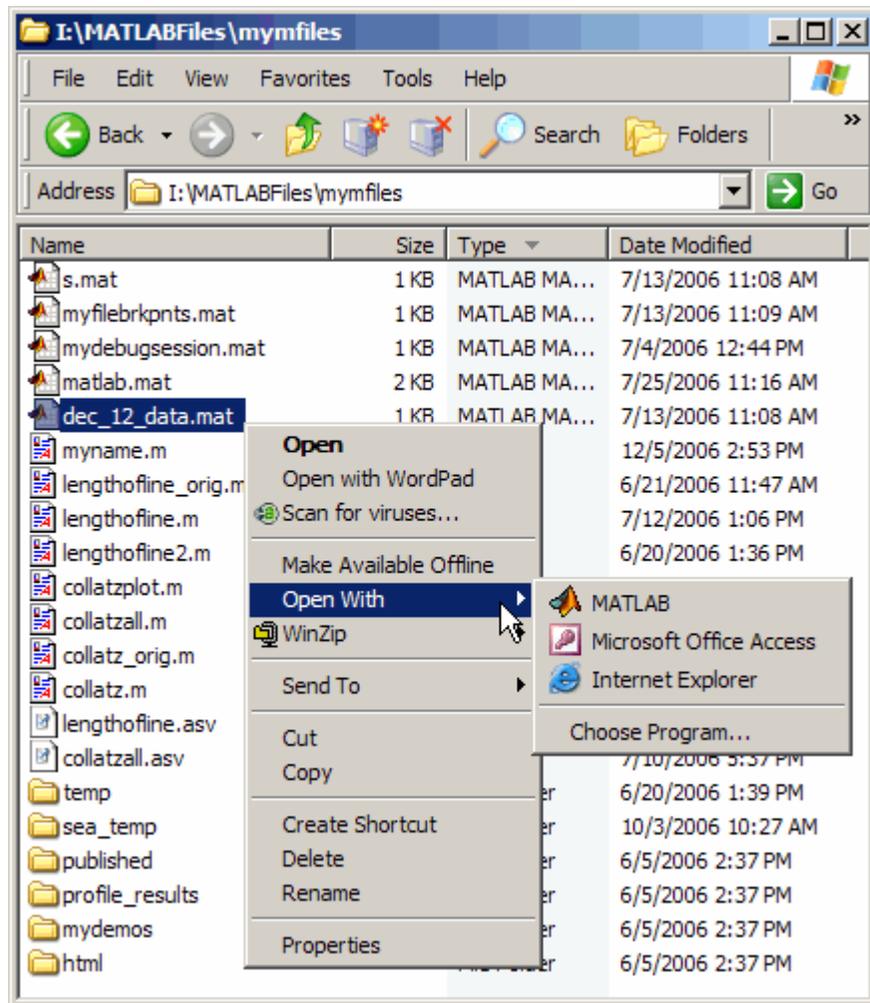
File associations for Windows Explorer do not affect what happens when you open one of these file types from *within* MATLAB. MATLAB acts on the file

using the MATLAB tool associated with that file type. For example, even if you associate .mat files with Microsoft Access, when you open a MAT-file from within MATLAB, it opens the Import Wizard to load the data.

Utility to Change Windows File Associations

If you are viewing this topic in the MATLAB Help browser, you can run one of the utilities provided here to create Windows associations for common MATLAB file types. This requires you to have permission to write to the HKEY_CLASSES_ROOT registry key, which typically requires power user or administrator privileges.

- Run utility to associate MATLAB with FIG-files

- Run utility to associate MATLAB with M-files

- Run utility to associate MATLAB with MAT-files

- Run utility to associate MATLAB with MDL-files

- Run utility to associate MATLAB with MEX-files

- Run utility to associate MATLAB with P-files

- Run utility to associate MATLAB with all of these file types: FIG, M, MAT, MDL. MEX, and P

The file type icon in Windows Explorer might not reflect the change immediately.

Changing File Associations for MATLAB from Windows

You can associate file types with MATLAB from Windows Explorer. This is useful if you want associate file types other than those offered by the above utilities. For example, you can double-click .xml files to open them in the MATLAB Editor/Debugger.

The following examples show one way to change file associations in Windows. Note that these instructions might not exactly apply to your version of Windows. If you encounter differences or problems, try to delete the association before using these instructions, or see your Windows documentation.

Assume that when you double-click a .mat file in Windows Explorer, it opens in Microsoft Access, but you want the file to open in MATLAB.

1 In Windows Explorer, select **Tools > Folder Options**.

2 In the resulting Folder Options dialog box, select the **File Types** tab. From the **Registered file types** list, select the MAT extension. (If you do not see MAT in the list, click **New** to add it.)

   Under **Details for 'MAT' extension**, click **Change**.

3 In the resulting Open With dialog box, select MATLAB from the list.

   If MATLAB is not listed, click **Browse**. Then look for and select matlab.exe, and click **Open**. The file is located in the folder in which you installed MATLAB. An example of the default location is C:\Program Files\MATLAB\R2007b\bin.

4 In the Open With dialog box, click **OK**. In the Folder Options dialog box, click **Close**.

Starting MATLAB on UNIX Platforms

To start MATLAB on a UNIX platform, type MATLAB at the operating system prompt.

If you did not set up symbolic links in the installation procedure, you must enter the full pathname to start MATLAB, *matlabroot*/bin/matlab, where *matlabroot* is the name of your MATLAB installation directory. If you have trouble starting MATLAB, see troubleshooting information in the Installation Guide for UNIX.

After starting MATLAB, the desktop opens. Desktop components that were open when you last shut down MATLAB will be opened on startup. For more information, see Chapter 2, "Desktop". If the DISPLAY environment variable is not set or is invalid, the desktop will not display.

Starting MATLAB on Macintosh Platforms

In this section...

Starting MATLAB from the Macintosh Desktop

To start MATLAB on Macintosh platforms, double-click the MATLAB icon on the desktop. If there isn't a MATLAB icon on the desktop, you can find it in the MATLAB folder within the Applications folder. MATLAB starts, assuming you accepted the defaults for the license manager to automatically start at system startup.

Note On Macintosh systems, you cannot perform a remote login, that is, you cannot run MATLAB remotely. For example, you cannot `rlogin`.

Starting MATLAB from an M-File or Other File Type on Macintosh Platforms

On Macintosh platforms, you can start MATLAB from the Finder by double-clicking a file with one of these extensions: `.fig`, `.m`, `.mat`, and `.mdl`. MATLAB starts and opens in an appropriate tool. If MATLAB is already running, the file opens in an appropriate tool in the existing session—see File Type and Resulting Action on page 1-3.

Startup Directory for MATLAB

In this section...
"What Is the Startup Directory?" on page 1-8
"Startup Directory (Folder) on Windows Platforms" on page 1-8
"Startup Directory on UNIX Platforms" on page 1-9
"Changing the Startup Directory" on page 1-9

What Is the Startup Directory?

The *startup directory* is the current directory in MATLAB when it starts. The default startup directory depends on your platform and installation. You can specify a different startup directory.

Startup Directory (Folder) on Windows Platforms

When you start MATLAB from a Windows shortcut icon, MATLAB sets the default startup directory to `My Documents\MATLAB` or `Documents\MATLAB` on Vista. This is a location where you can conveniently work with your personal MATLAB files. It offers these features:

- Because the `MATLAB` subfolder is the default startup directory, the current directory is automatically the one containing the files your work with.

- So that the files you work with are on the search path, MATLAB automatically adds the `MATLAB` subfolder to the top of the search path upon startup. If the `MATLAB` subfolder does not exist, MATLAB creates it.

- This location also utilizes the benefits provided by the Windows (or Vista) standard location for storing personal files, `My Documents` (or `Documents`). Files in the `MATLAB` subfolder are available to you when you use other machines, if your profile is set to roam. Because each user has their own `My Documents` or `Documents` folder, other users, even those using your machine, cannot access files in your `MATLAB` subfolder.

- When you upgrade to a newer version of MATLAB, MATLAB automatically continues to use the same `MATLAB` subfolder and your existing files, with all of the above benefits.

Startup Directory on UNIX Platforms

On UNIX platforms, the default startup directory is the directory you are in on your UNIX file system when you start MATLAB, for example, `/home/$user/matlab`. MATLAB automatically adds this `matlab` directory to the top of the MATLAB search path upon startup.

Changing the Startup Directory

You can start MATLAB in a directory other than the default.

For Windows Platforms Only

To change the startup directory on Windows platforms,

1 Right-click the MATLAB shortcut icon and select **Properties** from the context menu.

The Properties dialog box for MATLAB opens to the **Shortcut** pane.

2 The **Target** field contains the full path to start MATLAB.

By default, the startup directory is `My Documents\MATLAB` or `Documents\MATLAB` on Vista; for more information, see "Startup Directory (Folder) on Windows Platforms" on page 1-8.

In the **Start in** field, specify the full path to the directory in which you want MATLAB to start, and click **OK**.

In the **Start in** field, enter the full path to the directory in which you want MATLAB to the start. For example, `I:\my_matlab_files`.
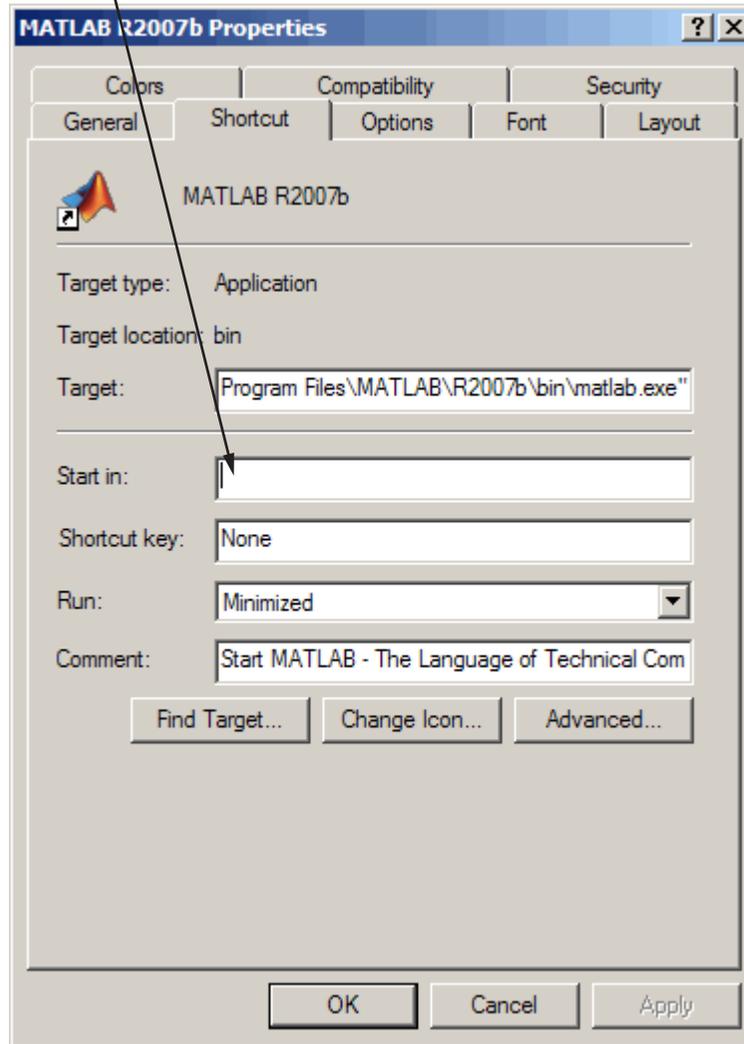
The next time you start MATLAB using that shortcut icon, the current directory will be the one you specified in step 2.

You can make multiple shortcuts to start MATLAB, each with its own startup directory, and with each startup directory having different startup options.

For All Platforms

To change the startup directory,

1 Create a `startup.m` file — see "Using the Startup File for MATLAB, startup.m" on page 1-12.

2 In the `startup.m` file, include the `cd` function to change to the new directory.

3 Put the `startup.m` file in the default startup directory for your platform, as noted in "Startup Directory (Folder) on Windows Platforms" on page 1-8 or "Startup Directory on UNIX Platforms" on page 1-9. Alternatively, put `startup.m` in the current startup directory.

Startup Options

About Startup Options

You can define startup options that instruct MATLAB to perform certain operations when you start it. You specify these options using a startup file (startup.m) or as options to the matlab startup command.

Using the Startup File for MATLAB, startup.m

At startup, MATLAB automatically executes the master M-file matlabrc.m and, if it exists, startup.m. The file matlabrc.m, which is in the *matlabroot*/toolbox/local directory, is reserved for use by The MathWorks and by the system manager on multiuser systems.

The file startup.m is for you to specify startup options. For example, you can modify the default search path, predefine variables in your workspace, or define Handle Graphics® defaults. Creating a startup.m file with the lines

```
addpath /home/$user/mytools
cd /home/$user/mytools
```

adds /home/*$user*/mytools to your default search path and makes mytools the current directory upon startup.

Location of startup.m

Place the startup.m file in the default or current startup directory, which is where MATLAB first looks for it. For more information, see "Startup Directory for MATLAB" on page 1-8.

Adding Startup Options for Windows Platforms

You can add selected startup options (also called command flags or switches for the command line) to the target path for your Windows shortcut for MATLAB. Or you can add them to the command line when you start MATLAB in a DOS window. On Windows systems a startup option is preceded by either a hyphen (-) or a slash (/). For example, `-nosplash` and `/nosplash` are equivalent ways of specifying the `nosplash` option for Windows users.

Startup Options in Windows Shortcut

To use startup options in the Windows shortcut for MATLAB, follow these steps:

1 Right-click the MATLAB shortcut icon and select **Properties** from the context menu. The Properties dialog box for MATLAB opens to the **Shortcut** pane.

2 In the **Target** field, after the target path for `matlab.exe`, add the startup option, and click **OK**. For example, adding `-r "filename"` runs the M-file `filename` after startup.

This example instructs MATLAB to automatically run the file `results` after startup, where `results.m` is in the startup directory or on the MATLAB search path. The statement in the **Target** field might appear as

```
C:\Program Files\MATLAB\bin\matlab.exe -r "results"
```

Include the statement in double quotation marks (`"statement"`). Use only the filename, not the file extension or pathname. For example, MATLAB produces an error when you run

```
... matlab.exe -r "D:\results.m"
```

Use semicolons or commas to separate multiple statements. This example changes the format to `short`, and then runs the M-file `results`:

```
... matlab.exe -r "format('short');results"
```

Separate multiple options with spaces. This example starts MATLAB without displaying the splash screen, and then runs the M-file `results`:

```
... matlab.exe -nosplash -r "results"
```

Startup Options in DOS Window

When you start MATLAB in a DOS window, include startup options after the matlab command.

This example uses the nosplash startup option to start MATLAB without the splash screen, and adds the -r option to run the results function located in the startup directory, after starting MATLAB in a DOS window:

```
matlab -nosplash -r "results"
```

Adding Startup Options for UNIX Platforms

Include startup options (also called command flags or command line switches) after the matlab command. On UNIX systems a startup option is preceded by a hyphen (-). For example, to start MATLAB without the splash screen, type

```
matlab -nosplash
```

Commonly Used Startup Options

The following table provides a list of some commonly used startup options for both Windows and UNIX platforms. For more information, including a complete list of startup options, see the matlab (Windows) reference page or the matlab (UNIX) reference page.

Platform	Option	Description
All	-c licensefile	Set LM_LICENSE_FILE to licensefile. It can have the form port@host.
All	-h or -help	Display startup options (without starting MATLAB).
All	-logfile "logfilename"	Automatically write output from MATLAB to the specified log file.

Platform	Option	Description
UNIX	-nodesktop	Start MATLAB without bringing up the MATLAB desktop. Use this option to run without an X-window, for example, in VT100 mode, or in batch processing mode. Note that if you pipe to MATLAB using the > constructor, the nodesktop option is used automatically.
		With nodesktop, you can still use most development environment tools by starting them via a function. For example, use preferences to open the Preferences dialog box and helpbrowser to open the Help browser.
		Do not use nodesktop to provide a command-line interface. If you prefer a command-line interface, select **Desktop > Desktop Layout > Command Window > Only**.
Windows	-minimize	Start MATLAB with the desktop minimized. Any desktop tools or documents that were undocked when MATLAB was last closed will not be minimized upon startup.
UNIX	-nojvm	Start MATLAB without loading the Java VM. This minimizes memory usage and improves initial startup speed, but restricts functionality. With nojvm, you cannot use the desktop, or any tools that require Java.
		For example, you cannot set preferences if you start MATLAB with the -nojvm option. However, you can start MATLAB once *without* the -nojvm option, set the preference, and quit MATLAB. MATLAB remembers that preference when you start it again, even if you use the -nojvm option.

Platform	Option	Description
All	`-nosplash`	Start MATLAB without displaying the MATLAB splash screen.
All	`-r "statement"`	Automatically run the specified MATLAB statement immediately after MATLAB starts. This is sometimes referred to as calling MATLAB in batch mode. Files you run must be in the MATLAB startup directory or on the MATLAB search path. Do not include pathnames or file extensions. Enclose the statement in double quotation marks (`"statement"`).

Toolbox Path Caching in MATLAB

In this section...

About Toolbox Path Caching

For performance reasons, MATLAB caches toolbox directory information
across sessions. The caching features are mostly transparent to you. However,
if MATLAB does not see the latest versions of your M-files or if you receive
warnings about the toolbox path cache, you might need to update the cache.

Using the Cache File Upon Startup

Upon startup, MATLAB gets information from a cache file to build the toolbox
directory cache. Because of the cache file, startup is faster, especially if you
run MATLAB from a network server or if you have many toolbox directories.
When you end a session, MATLAB updates the cache file.

MATLAB does not use the cache file at startup if you clear the **Enable
toolbox path cache** check box in **File > Preferences > General**. Instead,
it creates the cache by reading from the operating system directories, which
is slower than using the cache file.

Updating the Cache and Cache File

How the Toolbox Path Cache Works

MATLAB caches (essentially, stores in a known files list) the names and
locations of files in *matlabroot*/toolbox directories. These directories are
for MathWorks supplied files that should not change except for product
installations and updates. Caching those directories provides better
performance during a session because MATLAB does not actively monitor
those directories.

1-17

We strongly recommend that you save any M-files you create and any MathWorks supplied M-files that you edit in a directory that is *not* in the *matlabroot*/toolbox directory tree. If you keep your files in *matlabroot*/toolbox directories, they may be overwritten when you install a new version of MATLAB.

When to Update the Cache

When you add files to *matlabroot*/toolbox directories, the cache and the cache file need to be updated. MATLAB updates the cache and cache file automatically when you install toolboxes or toolbox updates using the MATLAB installer. MATLAB also updates the cache and cache file automatically when you use MATLAB tools, such as when you save files from the MATLAB Editor/Debugger to *matlabroot*/toolbox directories.

When you add or remove files in *matlabroot*/toolbox directories by some other means, MATLAB might not recognize those changes. For example, when you

- Save new files in *matlabroot*/toolbox directories using an external editor

- Use operating system features and commands to add or remove files in *matlabroot*/toolbox directories

MATLAB displays this message:

```
Undefined function or variable
```

You need to update the cache so MATLAB will recognize the changes you made in *matlabroot*/toolbox directories.

Steps to Update the Cache

To update the cache and the cache file,

1 Select **File > Preferences > General**.

The **General Preferences** pane is displayed.

2 Click **Update Toolbox Path Cache** and click **OK**.
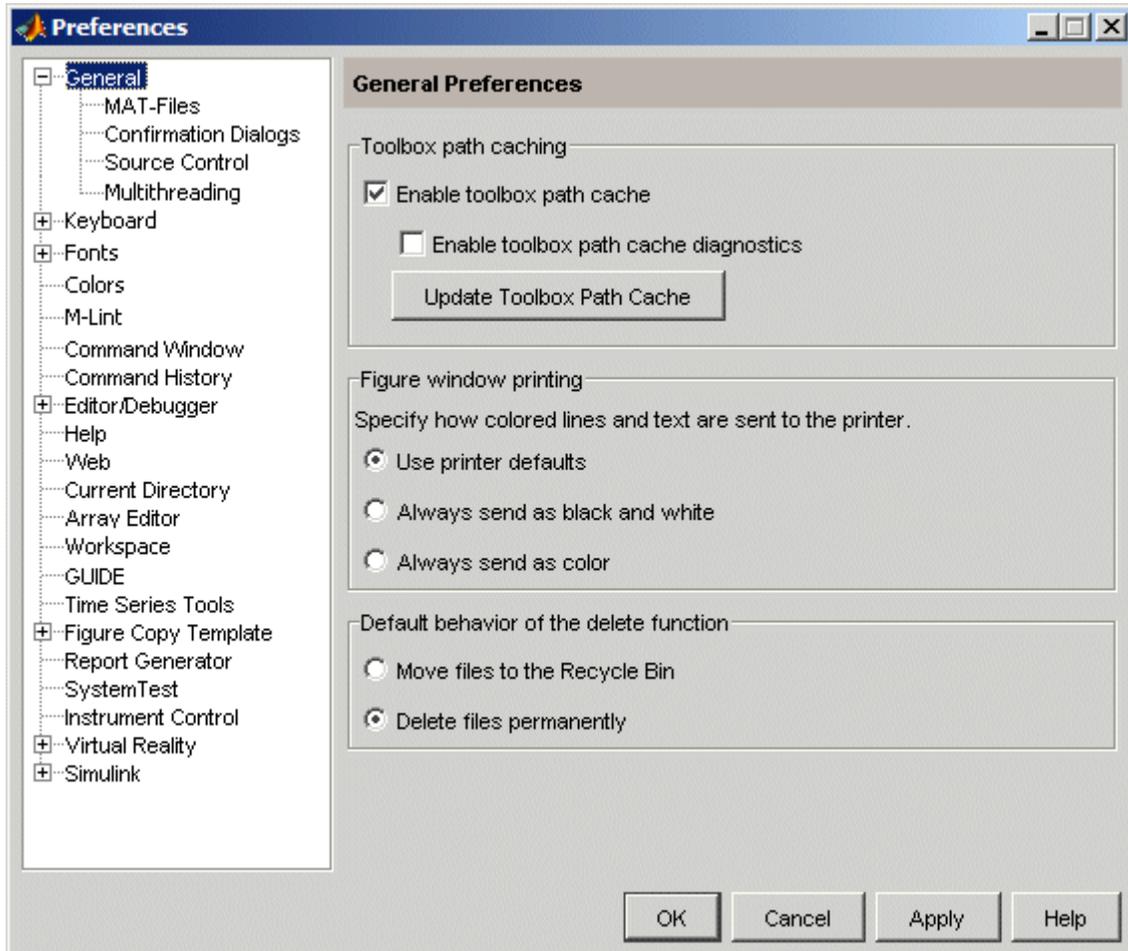
Function Alternative

To update the cache, use `rehash toolbox`. To also update the cache file, use `rehash toolboxcache`. For more information, see `rehash`.

Additional Diagnostics with Toolbox Path Caching

To display information about startup time when you start MATLAB, select the **Enable toolbox path cache diagnostics** check box in **General Preferences**.

Other Startup Topics

In this section...
"Error Log Reporter" on page 1-21
"Passing Perl Variables on Startup" on page 1-21
"Startup and Calling Java from MATLAB" on page 1-22

Error Log Reporter

Upon startup, if MATLAB detects an error log generated by a serious problem encountered during the *previous* session, an Error Log Reporter prompts you to e-mail the log to The MathWorks for analysis. Click **Send Report** to e-mail the log, or click **Help** for more information. After sending the log, a confirmation message appears in the Command Window. For more information, see "Abnormal Termination" on page 1-24.

Passing Perl Variables on Startup

You can pass Perl variables to MATLAB on startup by using the `-r` option of the `matlab` function. For example, assume a MATLAB function `test` that takes one input variable:

```
function test(x)
```

To start MATLAB with the function `test`, use the command

```
matlab -r "test(10)"
```

On some platforms, you might need to use double quotation marks:

```
matlab -r "test(10)"
```

This command starts MATLAB and runs `test` with the input argument 10.

To pass a Perl variable instead of a constant as the input parameter, follow these steps.

1 Create a Perl script such as

```
#!/usr/local/bin/perl
$val = 10;
system('matlab -r "test(' . ${val} . ')"');
```

2 Invoke the Perl script at the prompt using a Perl interpreter.

For more information, see the `matlab (Windows)` or `matlab (UNIX)` reference page.

Startup and Calling Java from MATLAB

When MATLAB starts, it constructs the Java class path using `librarypath.txt` as well as `classpath.txt`. If you call Java from MATLAB, see more about this in "The Java Class Path" and "Locating Native Method Libraries" in the MATLAB External Interfaces documentation.

Quitting MATLAB

In this section...
"Ways to Quit MATLAB" on page 1-23
"Confirm Quitting MATLAB" on page 1-23
"Running a Script When Quitting MATLAB" on page 1-24
"Abnormal Termination" on page 1-24

Ways to Quit MATLAB

To quit MATLAB at any time, do one of the following:

- Click the Close box ⊠ in the MATLAB desktop.
- Select **Exit MATLAB** from the desktop **File** menu.
- Type quit at the Command Window prompt.

MATLAB closes after

- Prompting you to confirm quitting, if that preference is specified (see "Confirm Quitting MATLAB" on page 1-23)
- Prompting you to save any unsaved files
- Running the finish.m script, if it exists in the current directory or on the MATLAB path (see "Running a Script When Quitting MATLAB" on page 1-24)

Confirm Quitting MATLAB

To set a preference that displays a confirmation dialog box when you quit MATLAB, select **File > Preferences > General > Confirmation Dialogs**, select the **Confirm before quitting** check box, and click **OK**. MATLAB then displays the following dialog box when you quit.

1-23

For more information, see "Confirmation Dialogs Preferences" on page 2-81.

You can also display your own quitting confirmation dialog box using a `finish.m` script, as described in the following section.

Running a Script When Quitting MATLAB

When MATLAB quits, it runs the script `finish.m`, if `finish.m` exists in the current directory or anywhere on the MATLAB search path. You create the file `finish.m`. It contains statements to run when MATLAB terminates, such as saving the workspace or displaying a confirmation dialog box. There are two sample files in *matlabroot*/`toolbox`/`local` that you can use as the basis for your own `finish.m` file:

- `finishsav.m` — Includes a `save` function so the workspace is saved to a MAT-file when MATLAB quits.

- `finishdlg.m` — Displays a confirmation dialog box that allows you to cancel quitting.

For more information, see the `finish` reference page.

Abnormal Termination

In the event MATLAB experiences a segmentation violation (segv) or another serious problem, save your files and workspace if possible, exit MATLAB, and restart.

Upon startup, if MATLAB detects an error log generated by a serious problem during the *previous* session, an Error Log Reporter prompts you to e-mail the log to The MathWorks for analysis. If the problem occurs repeatedly,

make note of what seems to cause it and look for information about it in the MathWorks Bug Reports database.

There are some situations where the Error Log Reporter will not open, for example, when you start MATLAB with a `-r` option or run in deployed mode. If you experience segmentation violations but do not see the Error Log Reporter on subsequent startups, you can instead e-mail the reports by following the instructions at the end of the segmentation violation message in the Command Window.

Crash Recovery and Multithreading

When multithreaded computation is enabled, if MATLAB experiences a segmentation violation or other serious problem, it cannot try to return control to the Command Window. You do not have an opportunity to view a segmentation violation message in the Command Window as you might when multithreaded computation is *not* enabled. Instead, your platform's vendor, for example, Microsoft or Apple, provides an error dialog box. MATLAB then terminates.

Upon the next MATLAB startup after a fatal problem, the "Error Log Reporter" on page 1-21 prompts you to e-mail the log to The MathWorks.

2

Desktop

If you have an active Internet connection, you can watch the Working in the Development Environment video demo for an overview of the major functionality. The easiest way to learn to use the desktop is just by working with it. If you have problems or questions, refer to the following sections.

Overview of the Desktop

In this section...
"About the Desktop" on page 2-3
"Summary of Desktop Tools" on page 2-5

About the Desktop

In general, when you start MATLAB, it displays the MATLAB desktop, a set of tools (graphical user interfaces or GUIs) for managing files, variables, and applications associated with MATLAB.

The first time you start MATLAB, the desktop appears with the default layout, as shown in the following illustration. You can change the desktop arrangement to meet your needs, including resizing, moving, and closing tools. For details, see "Arranging the Desktop" on page 2-6.

Some tools, such as the Editor/Debugger and Array Editor, support multiple document windows within them. Similarly, you can group multiple figure windows together. For information about working with documents in the desktop, see "Opening and Arranging Documents" on page 2-8.

Menus change, depending on the tool you are using.

Select the title bar for a tool to use that tool.

Get help.

View or change the current directory.

Move, maximize, minimize or close a window.

Click the **Start** button for quick access to tools and more.

View or execute previously run statements.

Drag the separator bar to resize windows.

Enter MATLAB statements at the prompt.
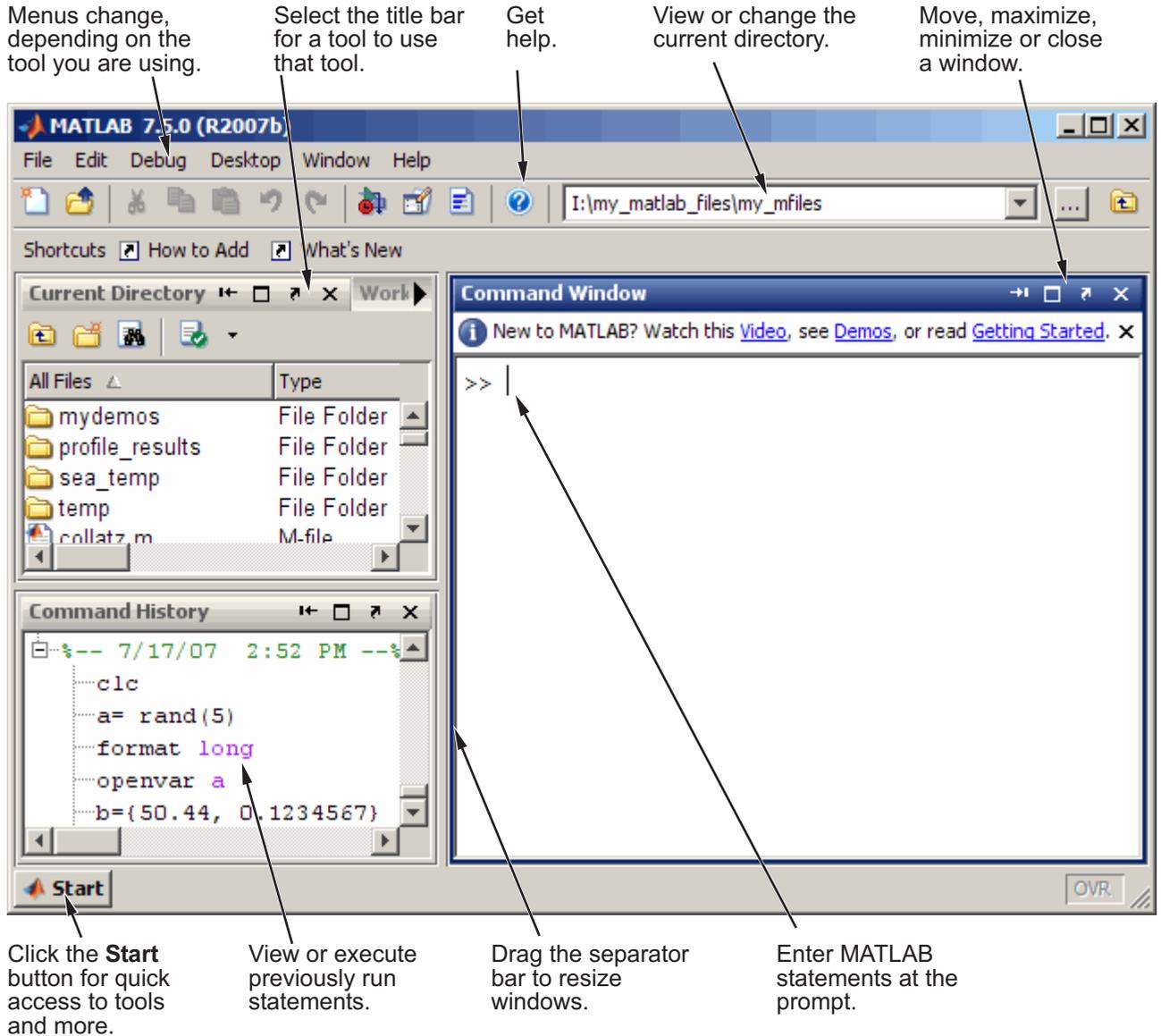
Summary of Desktop Tools

The following tools are managed by the MATLAB desktop, although not all of them appear by default when you first start. If you prefer a command-line interface, you can often use equivalent functions to accomplish the same result. To perform the equivalent of the GUI tasks in M-files, you must use the equivalent function. Instructions for using equivalent functions to perform the task are provided with the documentation for each tool and are typically labeled as Function Alternatives.

Desktop Tool	Description
Array Editor	View array contents in a table format and edit the values.
Command History	View a log of or search for the statements you entered in the Command Window, copy them, execute them, and more.
Command Window	Run MATLAB statements.
Current Directory Browser	View files, perform file operations such as open, find files and file content, and manage and tune your files.
Editor/Debugger	Create, edit, debug, and analyze M-files (files containing MATLAB statements).
Figures	Create, modify, view, and print MATLAB figures.
File Comparisons	View line-by-line differences between two files.
Help Browser	View and search the documentation and demos for all your MathWorks products.
Profiler	Improve the performance of your M-files.
Start Button	Run tools and access documentation for all your MathWorks products, and create and use MATLAB shortcuts.
"Web Browser" on page 2-55	View HTML and related files produced by MATLAB.
Workspace Browser	View and make changes to the contents of the workspace.

Arranging the Desktop

In this section...
"Modifying the Desktop Configuration" on page 2-6
"Opening and Arranging Tools" on page 2-6
"Opening and Arranging Documents" on page 2-8
"Saving Desktop Layouts" on page 2-13

See also "Examples of Desktop Arrangements" on page 2-15.

Modifying the Desktop Configuration

You can modify the desktop configuration to best meet your needs. Because the desktop uses many standard graphical user interface (GUI) conventions, it is easy to learn about arranging the desktop just by using it.

The desktop manages tools differently from documents. The Command History and Editor/Debugger are examples of tools, and an M-file is an example of a document, which appears in the Editor/Debugger tool.

Opening and Arranging Tools

This table summarizes actions for arranging desktop tools. For further information, click the "see more details online" links.

Tool Action	Steps to Perform
Opening desktop tools	To maximize your work area, keep open only those tools you use. To open a tool, select the tool name from the **Desktop** menu. Opened tools have a check mark before them in the menu. The tool appears in the location it occupied the last time it was open. The sizes of other tools adjust to accommodate the newly opened tool. See more details online.

Tool Action	Steps to Perform
Navigating among desktop tools	The **Window** menu displays all open desktop tools and documents, as well as opened tools for other MathWorks products. Select an entry in the **Window** menu to go directly to that tool or document. Another way to access an undocked desktop tool is by selecting its entry in the Windows task bar, or the equivalent for your platform. See also "Keyboard Shortcuts" on page 2-40 and more details online.
Closing desktop tools	To close a desktop tool, select the item in the **Desktop** menu, which clears the check mark in the menu and closes the tool. Or click the Close box (X) in the title bar for the tool, or select **File > Close** for the tool. See more details online.
Resizing tools	To resize tools in the MATLAB desktop, drag the separator bar, which is the bar between tools. You can hide the title bars for tools in the desktop so the tools use less space—select **Desktop > Titles**, and then hover over a title bar to see a tooltip containing the name of the tool. See more details online.
Moving tools within the desktop	To move a tool in the MATLAB desktop, drag the title bar of the tool toward where you want the tool to be located. As you drag the tool, an outline of it appears. When the outline nears a position where you can keep it, the outline snaps to that location. Release the mouse button. The tool stays at the new location. Other tools in the desktop resize to accommodate the new configuration. The inside edges of the desktop container and tools all act as if they are "sticky," so you can position a tool along any inside edge. See more details online.
Moving tools out of the desktop (undocking)	Move a tool out of the desktop to make it larger or easier to work with. To move a tool outside the MATLAB desktop (called *undocking*), select the tool to make it active, and then select **Desktop > Undock > Toolname**. The tool appears outside the MATLAB desktop and an entry for it appears in the Windows task bar or the equivalent for your platform. Tools within the desktop resize accordingly. Another way to undock is by using the Undock button ⊿ in the tool's title bar. See more details online.
Moving tools into the desktop (docking)	To move a tool that is outside the MATLAB desktop into the desktop, click the Dock button ⊾ in the tool's menu bar, or select **Desktop > Dock Toolname**. See more details online.

Tool Action	Steps to Perform
Grouping tools together	You can group tools so that they overlay each other in the MATLAB desktop. To group tools together, drag the title bar of one tool in the desktop on top of the title bar of another tool in the desktop. To make a tool active, click its name in the title bar. See more details online.
Maximizing tools in the desktop	To resize the active tool so it occupies the entire MATLAB desktop, double-click the tool's title bar; to return to the layout prior to maximizing, double-click the title bar of the maximized tool. Alternatively, use the menus: select **Desktop > Maximize Toolname**. To return to the layout prior to maximizing, select **Desktop > Restore Toolname**. You can also use the Maximize button ■ and Restore button ▣ in the tool's title bar. This feature is not supported on Macintosh platforms.
Minimizing tools in the desktop	You can minimize any tool in the desktop, which creates a button along an edge of the desktop that represents the tool. Select **Desktop > Minimize Toolname**. You can also use the Minimize button ▯ in the tool's title bar. The tool's button appears along the edge indicated by the minimize arrow in the menu item or on the button. To move the tool's button to a different edge, right-click the button, and from the context menu, select an edge. To view or use a minimized tool, hover over or click the button—this temporarily opens the tool in the desktop. Once you are finished using the tool, click the button or another tool and the tool is again shown only as a button along the edge. To return the tool to the desktop layout position it occupied before being minimized, double-click the button. Alternatively, restore it by right-clicking the button and selecting **Restore > Toolname**, or use the Restore button ▣ in the tool's title bar. This feature is not supported on Macintosh platforms.

Opening and Arranging Documents

Open a document, such as an M-file or a variable, and it opens in its tool, for example, the Editor/Debugger or Array Editor. The following example illustration shows a desktop arrangement that includes Editor/Debugger and Array Editor documents. See instructions in "Summary of Actions for Arranging Documents" on page 2-11.

Example of Documents in the Desktop

Some common actions for working with documents in the desktop are

- Use the document bar to go to open documents.

- Use the **Window** menu or equivalent toolbar buttons to position documents.

- Close or undock a tool, including all documents in the tool.

- Undock a document from its tool.

- Use the document Close box with the **Ctrl** key to close the document without saving it and without displaying the unsaved document dialog box.

See also "Examples of Desktop Arrangements" on page 2-15.

Close, undock, or resize tool (Editor/Debugger),
including all open documents.

Position documents within the
tool using these options.

Undock document
from tool, or close
document.

Use
document
bar to go to
open
documents.

Click name
in title bar to
go to open
tool.

```
MATLAB 7.4.0 (R2007a)                                          _ □ ×
File   Edit   Text   Go   Cell   Tools   Debug   Desktop   Window   Help

  □ ☞   ✂   🖹 📋 ◊ ◊   🔲 📄 🖹   ?   I:\MATLABFiles\mymfiles   ▼   ...   📁

  🖹 Editor - I:\MATLABFiles\mymfiles\collatz.m              ↥ □ ↗ ×

  □ ☞ 🖫   ✂   🖹 📋 ◊ ◊   🖨 🔍 ← → f,   🖹 B...▼   »  □ ▼  ↗ ×

  1      function sequence=collatz(n)
  2      % Collatz problem. Generate a sequence of integers resolv
  3      % For any positive integer, n:
  4      %    Divide n by 2 if n is even
  5      %    Multiply n by 3 and add 1 if n is odd
  6      %    Repeat for the result
  7      %    Continue until the result is 1%
  8

 collatz.m  ×  lengthofline.m  ×

Command Window              ± □ ↗ ×  Workspace
>>




 🔺 Start                        collatz           Ln 1    Col 1    OVR
```

Summary of Actions for Arranging Documents

This table summarizes actions for arranging documents in their tool. For further information, click the click the "see more details online" links.

Document Action	Overview
Opening documents	When you open a MATLAB document, it opens in the associated tool. If the tool is not already open, it opens when you open the document and appears in the position it occupied when last used. Figures open undocked, regardless of the last position occupied.
	How to open a document depends on the document type:
	• M-file: Select **File > Open** and select the M-file. It opens in the Editor/Debugger.
	• Workspace variable: In the Workspace browser, double-click the variable. It opens in the Array Editor.
	• HTML file: In the Current Directory browser, double-click the file. It opens in the MATLAB Web Browser.
	• Figure: Type `plot` or use another graphics function. The plot appears in a figure window.
	There are many additional ways to open documents. See more details online.

Document Action	Overview
Navigating among documents — the document bar	When more than one document is open within a tool, each document is either maximized (the default), or arranged so that multiple documents are visible at once. Click a document that is in view to make it the active document. See also "Keyboard Shortcuts" on page 2-40.
	Use the document bar to go to a document that is open but not in view. The names of all open documents appear in the document bar. Select a document name in the document bar to make that document active. To show the document bar if it is not open, select **Desktop > Document Bar > Bar Position** and select the position for it, for example, **Right**. See more details online.
	Entries for undocked documents appear in the Windows task bar, or the equivalent for your platform. Click the task bar entry for a document to make that document active.
Positioning, moving, and resizing documents	To position open documents within their tool, select an arrangement from the **Window** menu when the tool is active, or by using the equivalent toolbar button for **Maximize**, **Float**, **Left/Right Tile**, **Top/Bottom Tile**, and **Tile**. You can also define a specific grid arrangement using **Window > Tile...**. On the Macintosh platform, the tile option might not be available in the **Window** menu so use the Tile button ⊞ instead.
	With the tile arrangements, you refine the document position by moving the pointer over the handle (▪) on the separator bar. A Close box then appears. When you click the Close box between two open documents, both documents stay open, but one moves on top of the other. When you click the Close box between a document and an empty tile, the empty tile closes.
	To move a document in a tiled arrangement, drag the title bar of a document to another tile. To resize tiled documents, drag the separator bar between the documents. See also the Editor/Debugger's "Split Screen Display" on page 6-38, which allows you to view two different parts of the same file simultaneously.
	To move or resize maximized documents, you move or resize their tool.
	See more details online.

Document Action	Overview
Closing documents	To close a document, click the Close box in the document's title bar. After closing all the documents in a tool, the tool remains open with no documents in it. If you select the Close box for the tool, all documents in that tool close. In the Editor/Debugger, when you close a file that has unsaved changes, a prompt appears asking if you want to save the document. To close a file without saving changes and without seeing the save prompt, use **Ctrl** when you click the document's Close box. See more details online.
Moving documents and tools out of the desktop (undocking)	To undock all documents in a tool from the desktop, click the Undock button in the tool's title bar. The tool and its documents move outside of the desktop. See more details online. To undock a document from its tool, click the Undock button for the document. The Undock button is either in the document's title bar, menu bar, or toolbar, depending on the document type and whether or not the document is within the desktop or is in its tool outside of the desktop. Undocked documents have entries in the Windows task bar (or the equivalent for your platform).
Docking documents and tools	When you dock a document, it moves to the position in the tool that it occupied before you undocked the document. To dock a document, click the Dock button in the document's menu bar. See more details online.
Grouping documents in a tool outside the desktop	To group all of the documents for a tool together outside of the desktop, undock the tool from the desktop, not just the documents. If you have already undocked all of the documents and closed the empty tool that had contained them, select **Desktop > Dock All in Editor**, for example. This moves all the documents into the tool in the desktop. Then undock the tool.

Saving Desktop Layouts

When you end a session, MATLAB saves the desktop layout. The next time you start MATLAB, the desktop is restored to the way you last had it.

To use a predefined layout, select **Desktop > Desktop Layout**, and choose a configuration. See more details in the online documentation.

To save your own layouts for later reuse, select **Desktop > Save Layout** and provide a name. To reuse a saved layout, select the name from **Desktop > Desktop Layout**. See more details in the online documentation.

Examples of Desktop Arrangements

About These Examples

Scan the illustrations in the following examples for a desktop arrangement similar to what you want, and then follow the brief instructions to achieve the arrangement. There are many different ways to accomplish the result; these instructions present just one way. The instructions might not apply exactly, depending on how your desktop looks before you start.

Tool Outside of Desktop and Other Tools Grouped Inside Desktop Example

This example shows two ways you can increase the size of a tool. One way is to move a tool outside of the desktop to increase its size. Here, the Help browser was moved outside of the desktop and made larger. To move a tool outside of the desktop, click the Undock button ⬈ in the tool's title bar when the tool is in the desktop.

Another way to increase the size of a tool is by grouping tools together inside the desktop, and then accessing a tool via the tool's name in the title bar. Here the Command Window, Command History, Workspace browser, and Current Directory browser are grouped together. To achieve this, drag the title bar of one tool on top of the title bar of the tool(s) you want to group it with.

Help browser is undocked from desktop to provide a large area for viewing documentation when needed.

Four tools in the desktop are grouped together, providing a large area for working with a given tool.
Click a tool's name in the title bar to make that tool active.

Help

File   Edit   View   Go   Favorites   Desktop   Window   Help

Help Navigator

Search for: info.xml   Go

Example: "plot tools" OR plot* tools

Contents | Index | Search Results | Demos

MATLAB
  Getting Started
  Examples
  Desktop Tools and Develop
    Sta
    Des
    Run

Title: Controlling Input :: Running Functions - Command Wi...

Tab Completion in the Command Window

MATLAB helps you automatically complete the names of these items as you type them in the Command Window:

MATLAB 7.4.0 (R2007a)

File   Edit   Debug   Desktop   Window   Help

I:\MATLABFiles\mymfiles

Command Window   **Command History**   **Workspace**   **Current Directory**

```
>> magic(5)

ans =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

>>
```

Start   OVR

Maximized Tool in Desktop Example

This example shows a way you can temporarily increase the size of a tool so that it occupies the entire area of the desktop. In this example, the Command Window in the default desktop layout is temporarily maximized by clicking the Maximize button ▣ in the tool's title bar.

Default desktop layout.
Maximize a tool, for example, the Command Window so it occupies the full MATLAB desktop area.
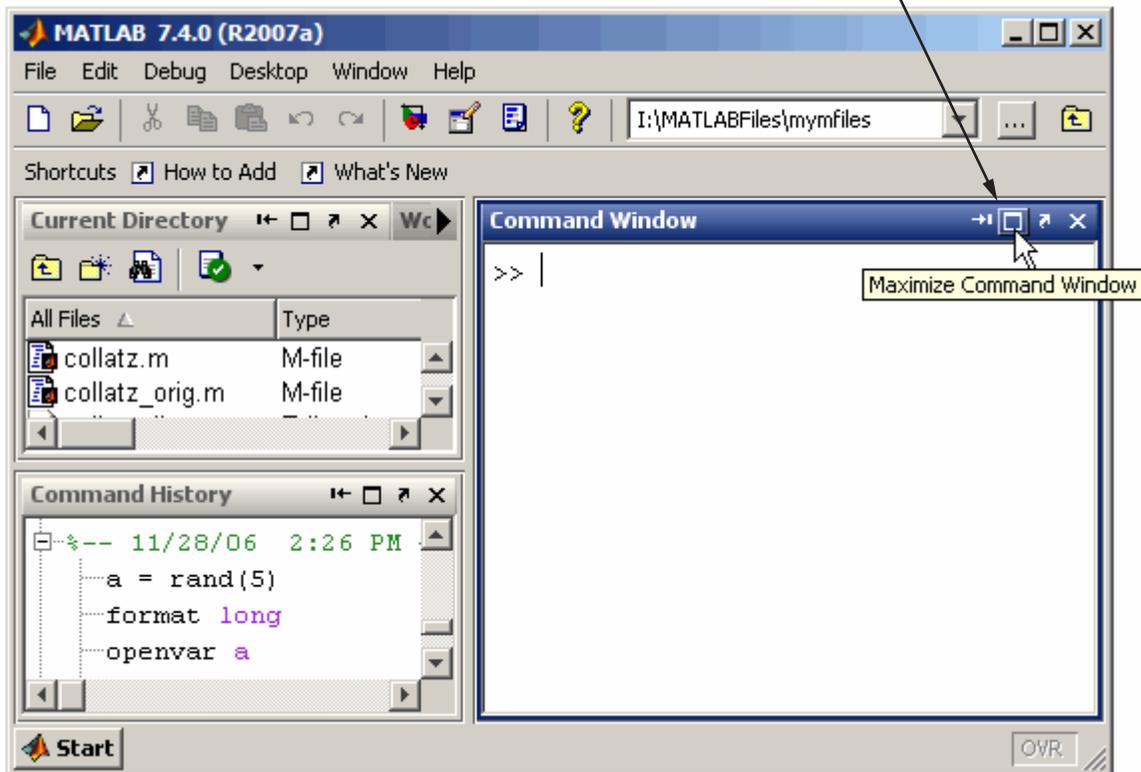
In this example, the maximized Command Window is being returned to its size and position in the default desktop layout by clicking the Restore button ▣ in the title bar.

Maximized, the Command Window now occupies the full desktop area.
Restoring the Command Window returns it to its original size and
location in the desktop.
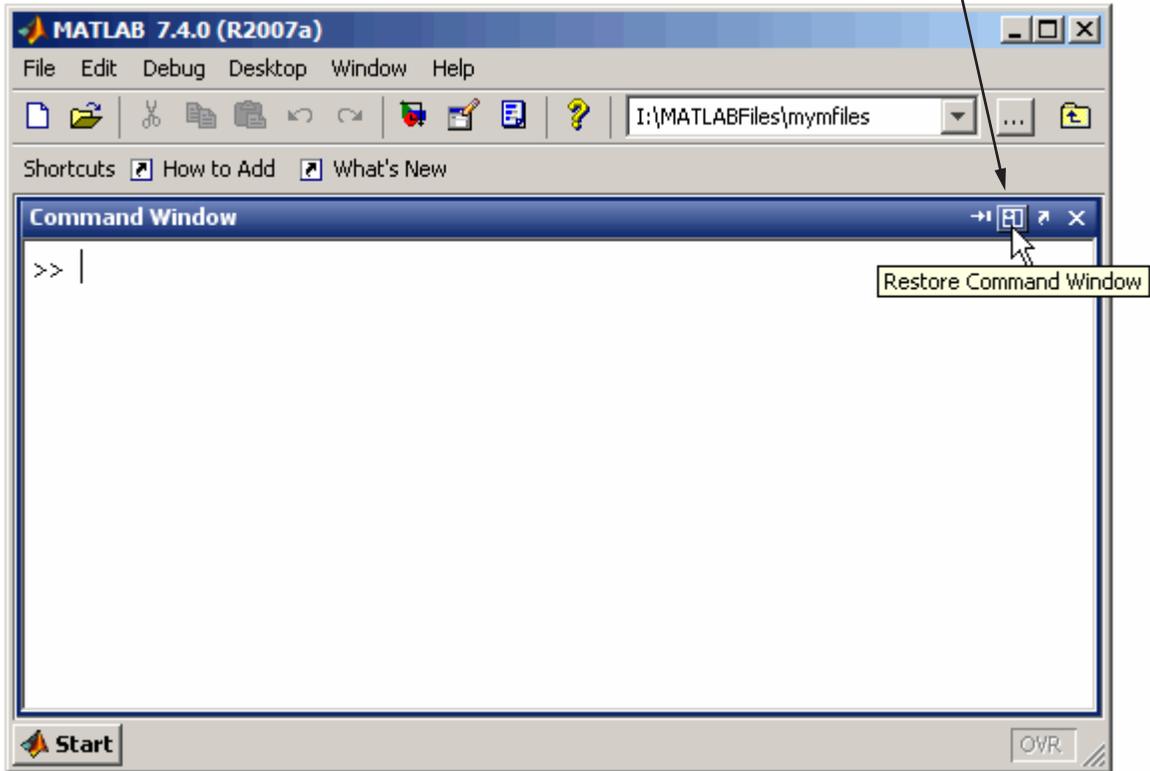
Minimized Tools in Desktop Example

Minimize a tool in the desktop to give the remaining desktop tools more space in the desktop. Minimizing is available on Windows and UNIX platforms. In this example, the Command History in the default layout is being minimized to the left edge of the desktop.

Minimize a tool, for example, the Command History window.
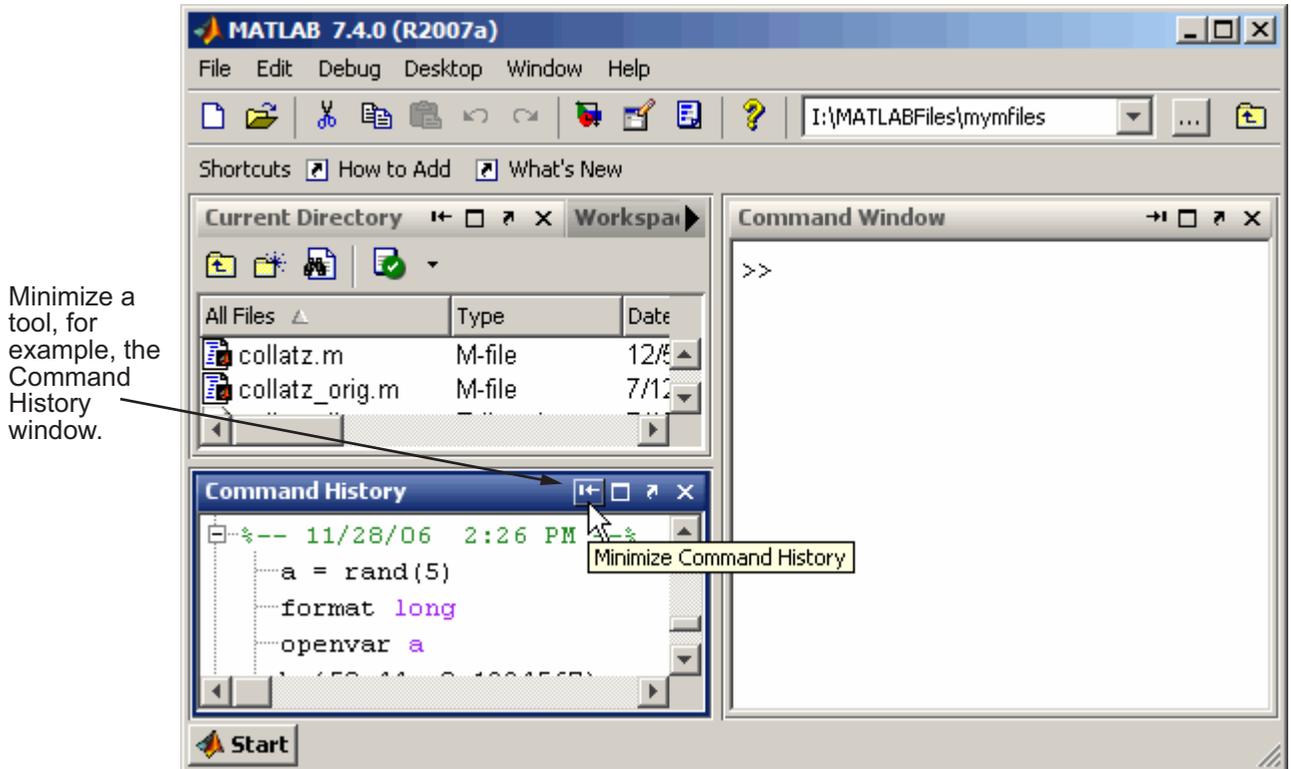
In this illustration, the Command History has been minimized and appears as a button along the left edge.

When minimized, a tool, such as the Command Window in this example, is represented by a button on the desktop border.
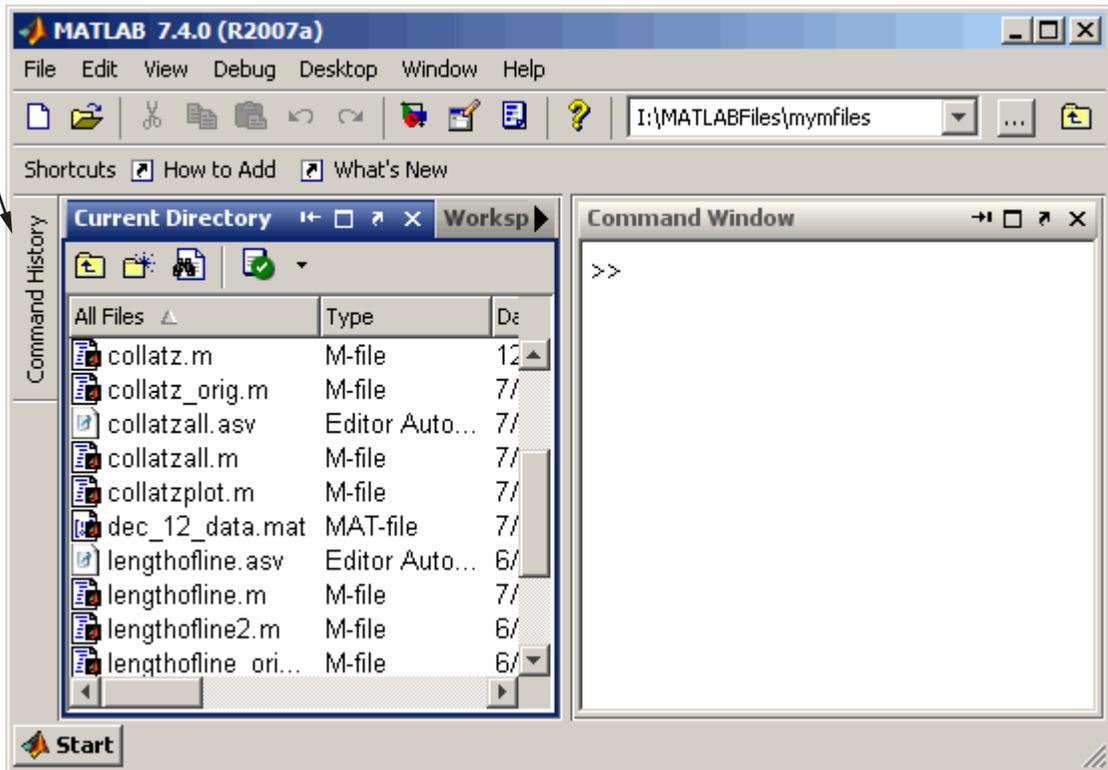
This illustration shows the minimized Command History being temporarily opened, as a result of clicking or hovering over the button.

Hover over or click the button for a minimized tool to temporarily view or use the tool.
The tool is temporarily displayed until you select another tool.
Then the tool becomes minimized again.
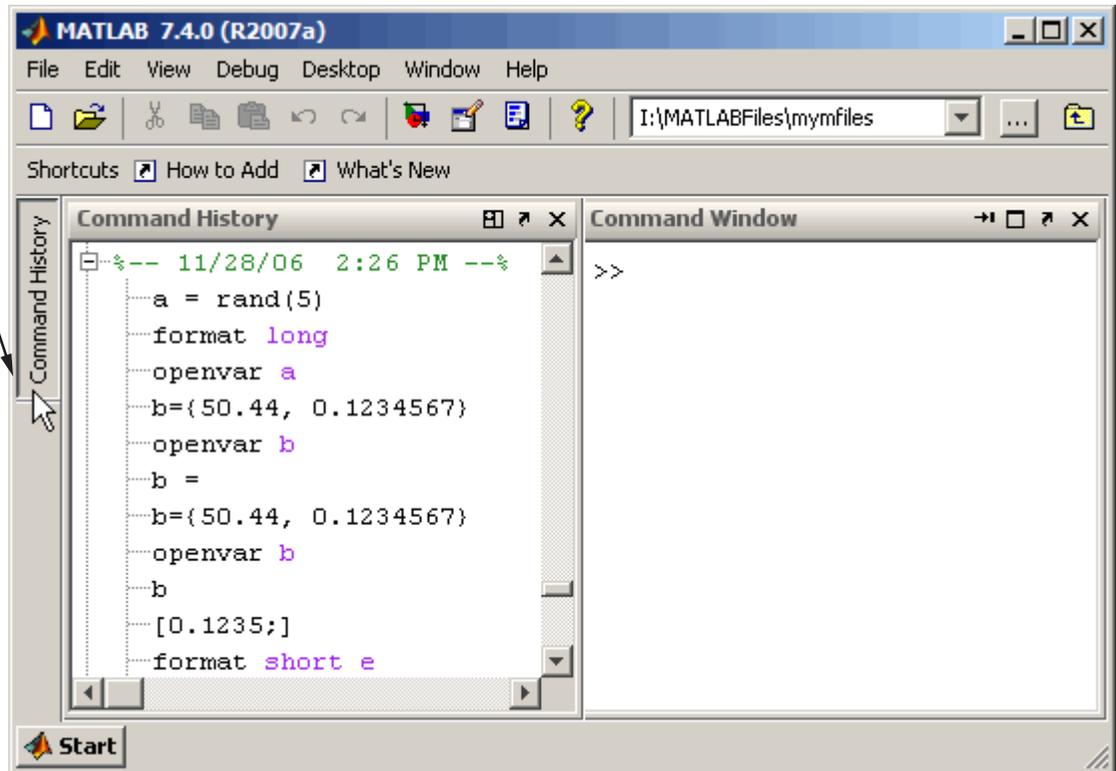
After using the Command History and clicking the button, or moving on to another tool, the Command History again becomes minimized as a button along the left edge.

This illustration shows the Command History being returned to the position and size it occupied in the desktop prior to being minimized by clicking the Restore button ⊞.

On the button for a minimized tool, right-click, and from the context menu,select **Restore**.
The tool resumes the size and position it had in the desktop before it was minimized.
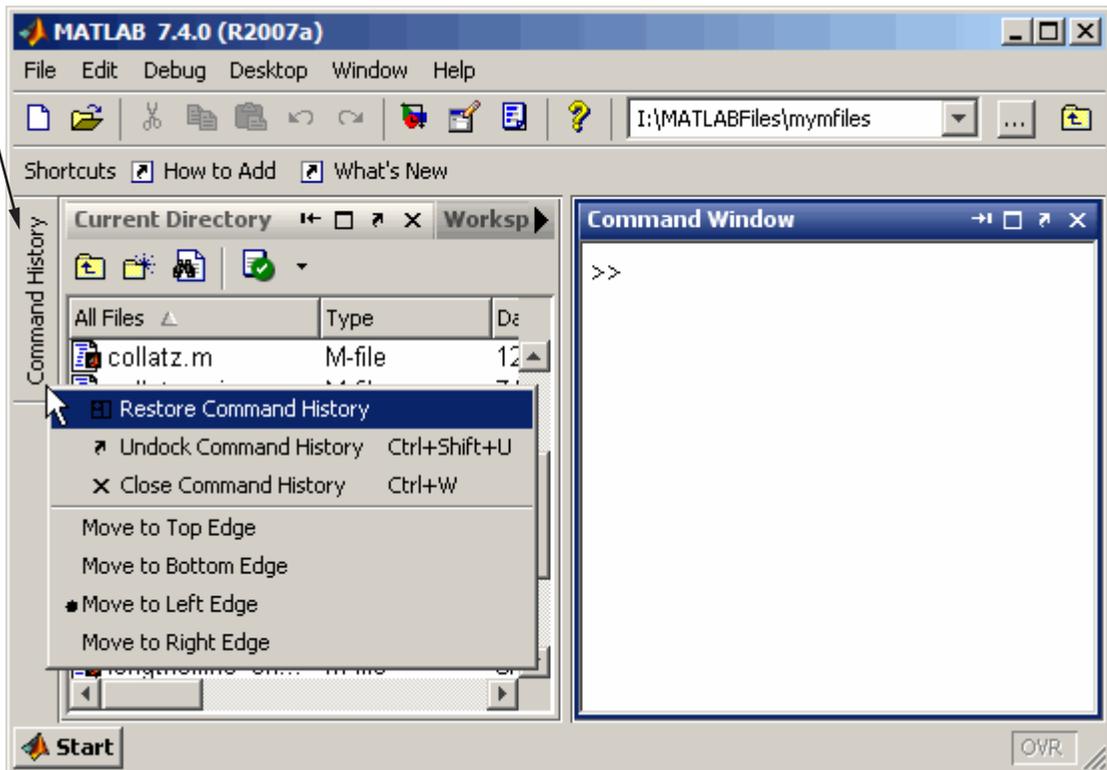
Tiled Documents in Desktop Example

When you open a document (for example, an M-file), it also opens the tool (for example, the Editor/Debugger) if the tool is not already opened. Subsequent documents of the same type open in the tool and you can then arrange the documents within the tool. You can move a document on top of another document, so that the one on top hides the one(s) beneath it, or you can show multiple documents at once. This example shows two M-files side-by-side, as a result of selecting **Window > Left/Right Tile** (or the ⬚ toolbar button).

When tools and documents are docked, you might want to save space by hiding toolbars and document bars. In this illustration, the desktop shortcuts toolbar is hidden. Select **Desktop > Toolbar name** to hide (or show) a toolbar. To see or move the document bar, select **Desktop > Document Bar > Bar Position**, and choose its location, for example, **Top**.

The shortcuts toolbar is hidden.
The document bar is at the top
edge of the Editor/Debugger.

Select a button from the list
to arrange the documents,
such as Left/Right Tile.
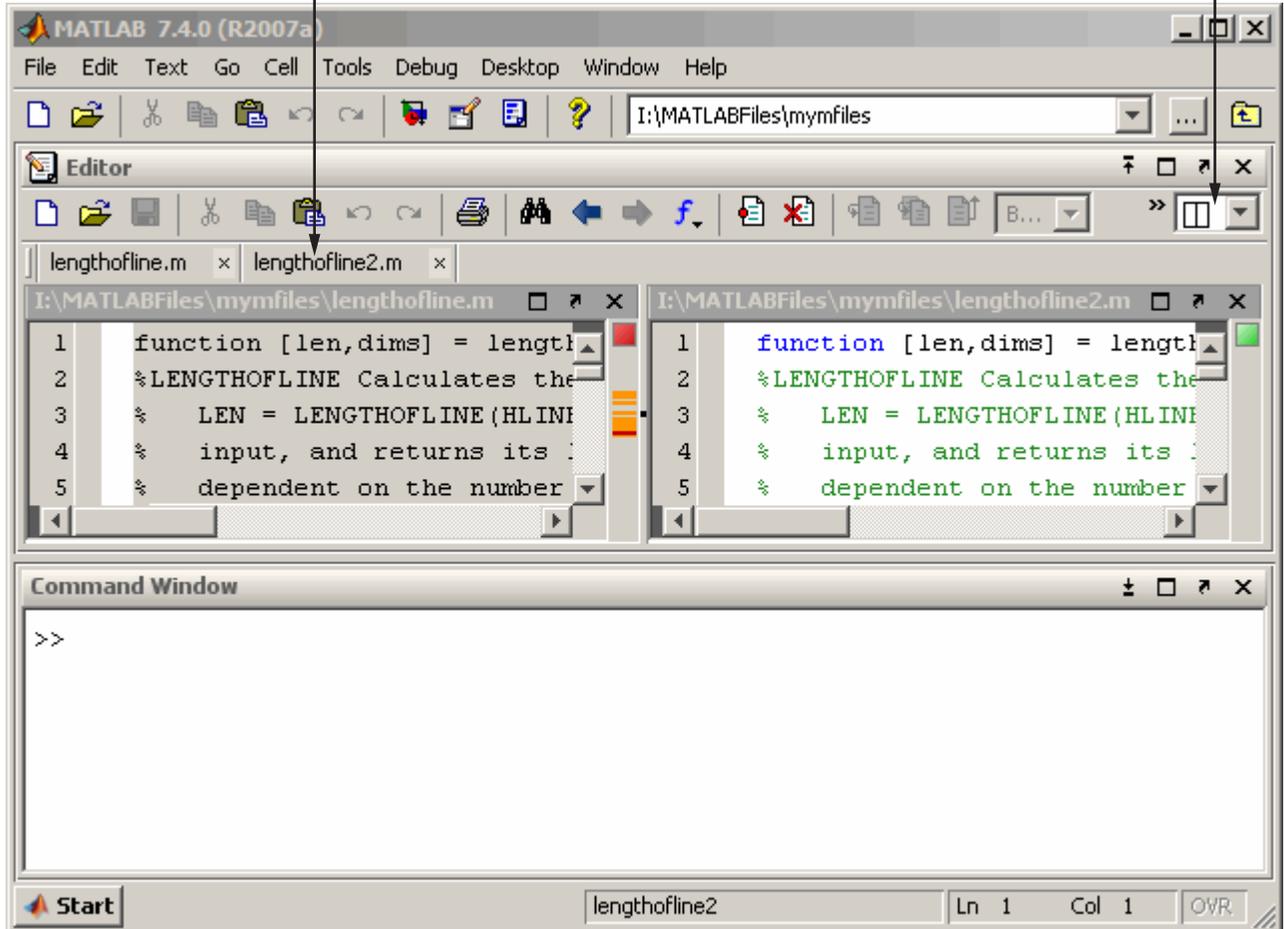
No Empty Document Tiles Example

To see more than two documents at once, select the Tile button and move the pointer across the gird that appears to select the number of tiles you want. The following "Before" illustration has four tiles, but only three documents are open. (The empty tile is gray.) You can move a document to any empty tile by dragging its title bar to the new location. To close an empty tile, position the pointer over the handle ▪ on the separator bar. It becomes a Close box, as shown here, which you click to close the empty tile. After clicking the Close box, the empty tile closes and the neighboring document expands as shown in the following "After" illustration. Similarly, click the Close box between two tiles containing documents, and one document becomes hidden. Note that preferences to show line numbers and M-Lint indicators have been cleared to provide more horizontal space.

Before

Close the empty tile using the handle on the separator bar.

Tile more than two documents with the Tile button.
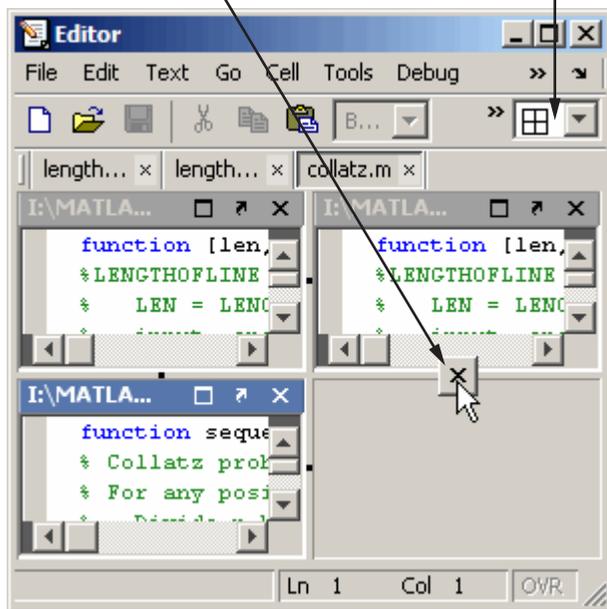
After

There are only three tiles.
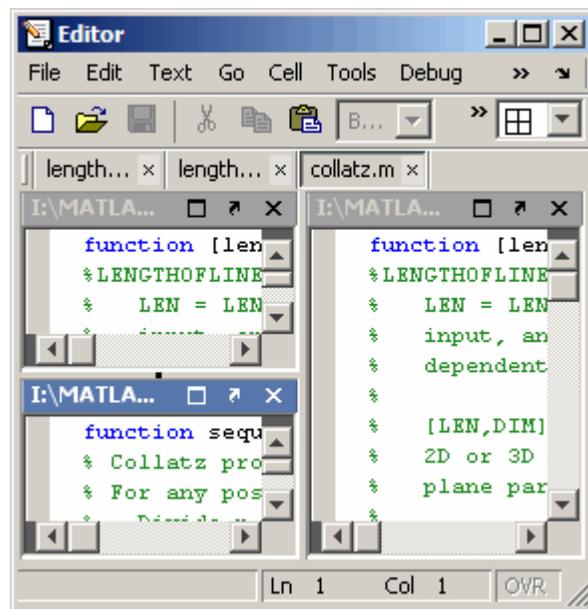
Maximized Documents Outside of the Desktop Example

This example illustrates a way to provide a large area for multiple documents, in this case, M-files maximized in the undocked Editor/Debugger.

Document bar on left. Cell toolbar for rapid code iteration and publishing. Maximize Document button.
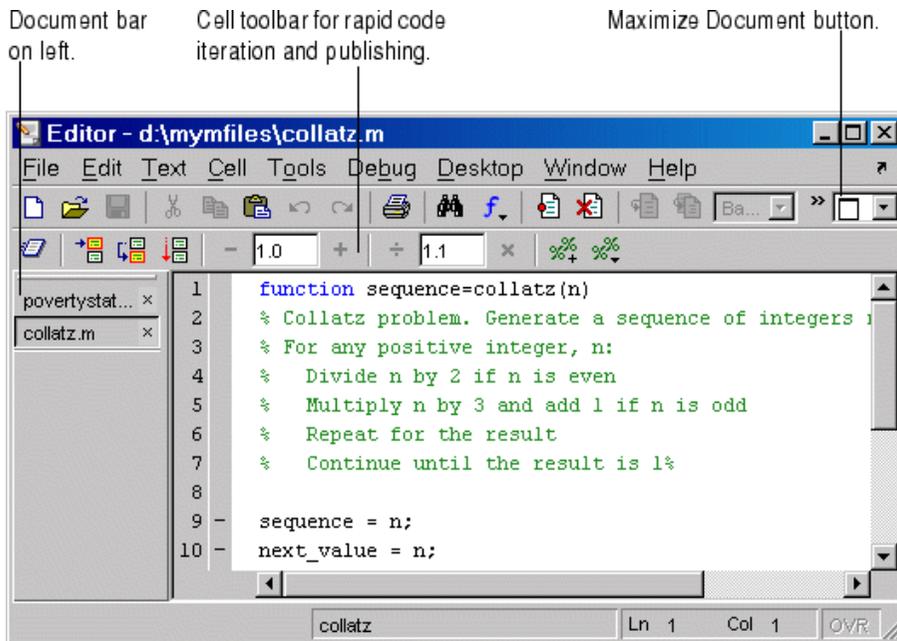
Some common actions for working with documents outside of the desktop are

• Group all Editor/Debugger documents together — select **Desktop > Dock All in Editor** from any Editor/Debugger document.

• Move all Editor/Debugger documents outside of the desktop — select **Desktop > Undock Editor** when the Editor/Debugger is the active window.

• Make a document occupy the full area in the Editor/Debugger — click the Maximize button in the Editor/Debugger toolbar, or select **Window > Maximize**.

• Display the cell toolbar — select **Desktop > Cell Toolbar**. This menu item is available only when the current document is an M-file.

- Access any document in the Editor/Debugger using the document bar. To show the document bar on the left side of the Editor/Debugger, select **Desktop > Bar Position > Document Bar > Left** from the Editor/Debugger.

Floating (Cascaded) Figures in Desktop Example

This example illustrates multiple figures in the desktop. By default, figures open outside the desktop. Click the Dock button in each figure's menu bar to move the figures into the desktop.

You can float (also called cascade) the figures by selecting **Window > Float**, or clicking the Float button 🗗. To get even more screen area for the figures, hide the document bar as shown in this example — select **Desktop > Document Bar > Bar Position > Hide**.

Dock figures in the desktop and use the float option to arrange them within a Figures group. The document bar is hidden.
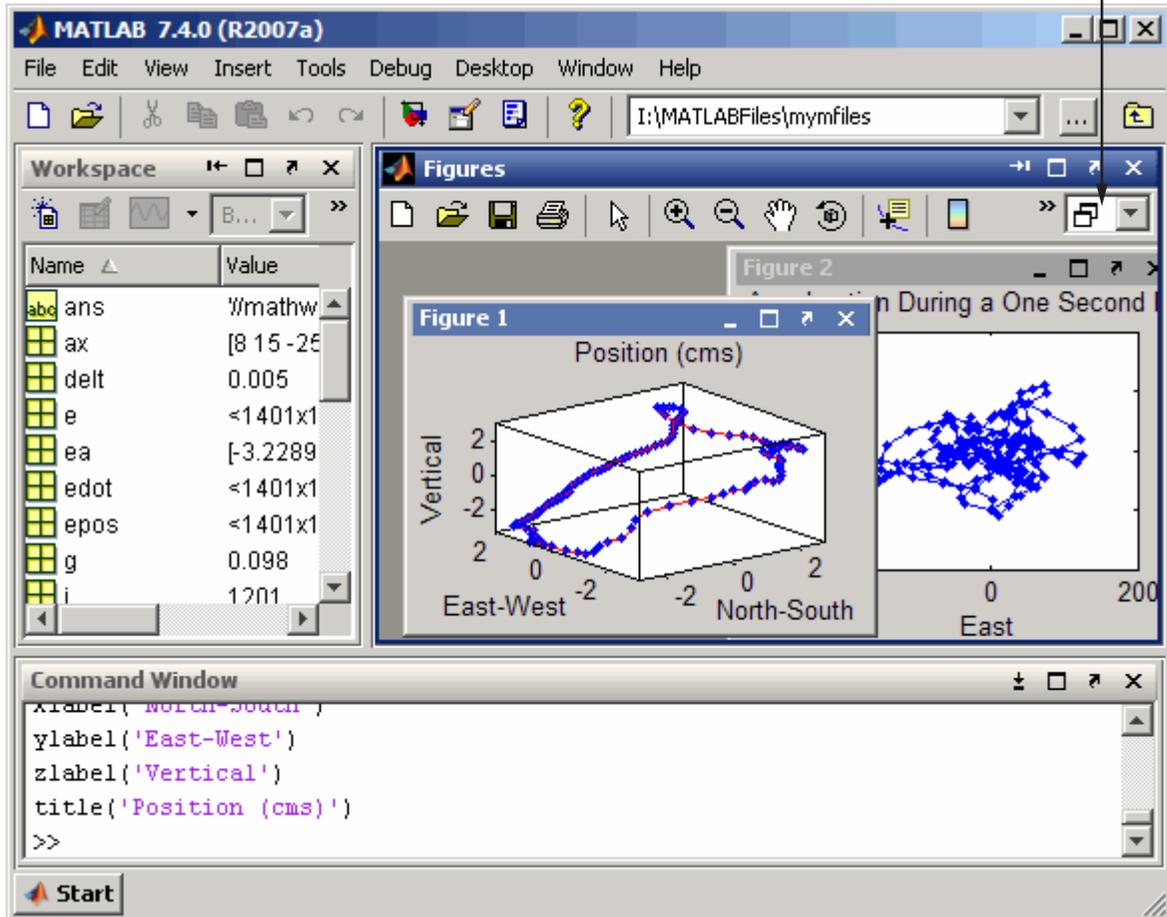
Undocked Tools and Documents Example

You can use tools and documents outside of the desktop. One way to achieve this is to first undock the tool from the desktop by selecting **Desktop > Undock Toolname**. Then undock documents from the undocked tool by selecting **Desktop > Undock Documentname** from the tool. If you undock all documents from a tool, an "empty" tool window remains.

In this example, one of the Editor/Debugger documents, collatz.m, includes the name of the tool with it; the other Editor/Debugger document, lengthofline.m, does not. Contrast this with the Array Editor documents, where neither document window includes the name of the tool. This is because the Array Editor was undocked from the desktop, the variables were undocked from the Array Editor, and the "empty" Array Editor window was closed. The tool's undocked documents remain open. If you closed the Editor/Debugger, the lengthofline.m document would remain open. To close all undocked documents and their tools at once, select **Window > Close All Documents** from an undocked document window.
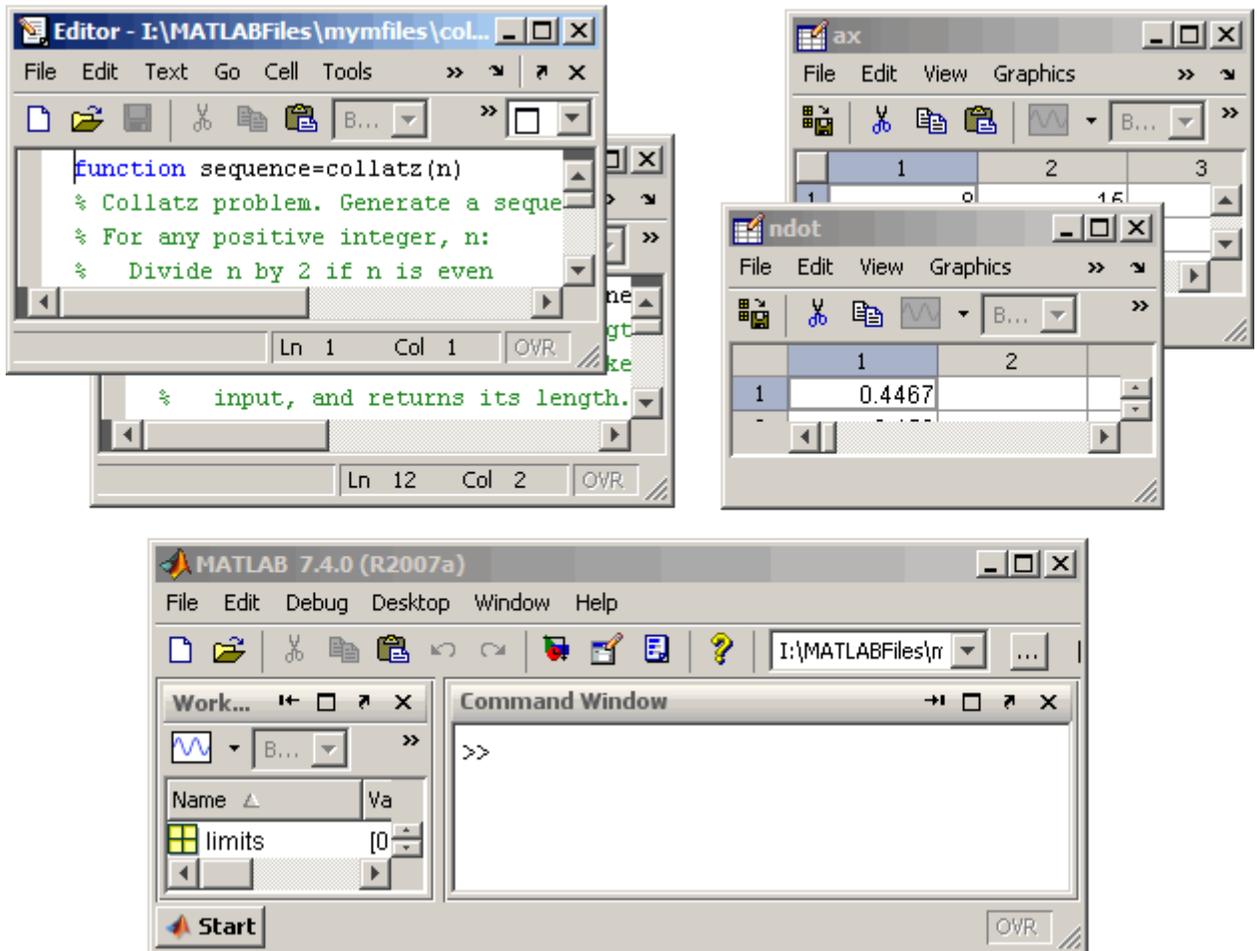
Shortcuts for MATLAB — Easily Run a Group of Statements

In this section...

What Is a Shortcut?

A MATLAB shortcut is an easy way to run a group of MATLAB statements that you use regularly. First you create a shortcut that contains all the statements. Then you select and run the shortcut to execute all the statements it contains. Create, run, and organize shortcuts from the **Start > Shortcuts** menu or the desktop **Shortcuts** toolbar.

Differences Between Shortcuts and M-Files

A shortcut is like an M-file script, but unlike an M-file, a shortcut does not have to be on the MATLAB search path or in the current directory when you run it. In addition, you can run the shortcut by selecting it from the **Start** button or desktop **Shortcuts** toolbar, which are readily accessible.

Although shortcuts run MATLAB statements, they are not M-files and are not stored as M-files.

Examples of Useful Shortcuts

These are some examples of useful types of shortcuts:

• If you frequently run the same group of functions, consider creating a shortcut for them. An example of this is setting up your environment when you start working if you do not use a startup file, or if there are statements you do not want to include in the startup file. Some users

create a shortcut for even a single function they use frequently, such as `clc` to clear the Command Window.

- Create a shortcut to set the same properties for figures you create, such as adding a legend and setting the background color.

- Create a shortcut for a long statement, such as changing the current directory (`cd`) when the pathnames are long.

- Create a shortcut for a statement you do not easily remember but need to use.

Creating Shortcuts

This is an example of a shortcut you might create for a project you work on, the Sea Temperature project. When you work on that project, you might want to set up your environment in a certain way by running a series of statements. You create a shortcut called `sea_temp_env`, which contains the statements. Then when you work on the project, you run the shortcut to execute all of the statements with a single click. The statements are

```
more on
format long e
cd d:/mymfiles/sea_temp_project
clear
workspace
filebrowser
clc
```

To create a shortcut, perform the following steps:

1 From the **Start** button, select **Shortcuts > New Shortcut**.

The Shortcut Editor dialog box appears.

2 Create the shortcut by completing the dialog box.

    a Provide a shortcut name in the **Label** field, for example, `sea_temp_environment`.

    b Put the MATLAB statements in the **Callback** field as shown in the following illustration. Either type them in, or copy and paste or drag them from a desktop tool. Edit the statements as needed. The field uses

2-33

the Editor/Debugger preferences for key bindings, colors, and fonts. Note that if you copy the statements from the Command Window, the prompt appears in the shortcut, but MATLAB removes the prompt when you save the shortcut.

c Assign a category, which is like a directory for organizing shortcuts. Specify `sea_temp_project`. To add the shortcut to the shortcuts toolbar, select the `Toolbar Shortcuts` category.

d Use the default shortcuts icon , or select your own.

e Click **Save**. MATLAB automatically removes any Command Window prompts (>>) in the **Callback** field upon saving the shortcuts.
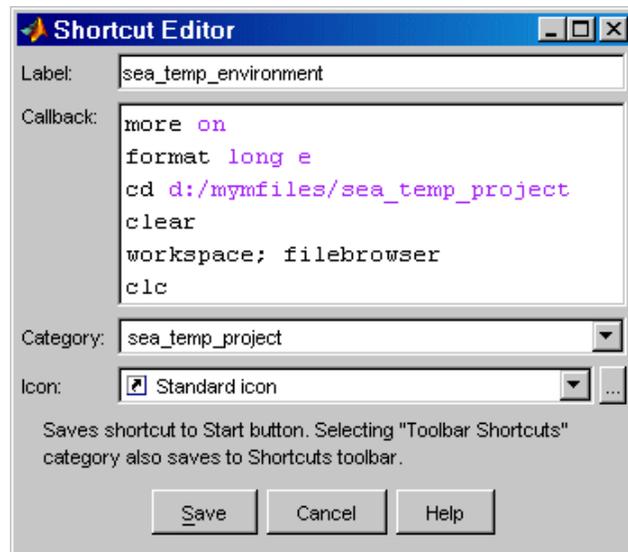
3 MATLAB adds the shortcut to the **Shortcuts** entry in the **Start** button, and to the **Shortcuts** toolbar, if you selected that **Category**.

After creating a shortcut, run it by selecting it from its category in the **Start** button. You can also run it from the **Shortcuts** toolbar if you selected the `Toolbar Shortcuts` category.

MATLAB maintains shortcut information in the file `shortcuts.xml`. Type `prefdir`, and MATLAB displays the location of the file. Most likely, you will not need to access this file, as MATLAB updates the file automatically.

For more information on the options in the Shortcut Editor dialog box, click the **Help** button.

Additional Ways to Create Shortcuts

You can also use these methods to create shortcuts:

* Add shortcuts to and run them from the desktop **Shortcuts** toolbar. See "Shortcuts Toolbar" on page 2-35.

* From the Command History window, create a shortcut by selecting statements, right-clicking, and selecting **Create Shortcut** from the context menu. By default, shortcuts created from the Command History window are assigned to the `Toolbar Shortcuts` category, meaning they will appear on the **Shortcuts** toolbar.

* From the Help browser, select **Favorites > Add to Favorites**, complete the Favorites Editor dialog box, and the shortcut appears in the shortcuts `Help Browser Favorites` category. You can also access `Help Browser Favorites` shortcuts from the Help browser **Favorites** menu.

* Drag statements from a desktop tool, such as the Command History, onto the **Start** button.

Running Shortcuts

To run a shortcut, select the shortcut name, for example, **sea_temp_environment**, from the **Start > Shortcuts** menu or from one of its category submenus. All of the statements in the shortcut **Callback** field execute. It is as if you ran those statements from the Command Window, although they are not reflected in the Command History window.

If you added a shortcut to the **Shortcuts** toolbar, you can run it by clicking its icon on the shortcuts toolbar.

Shortcuts Toolbar

The **Shortcuts** toolbar is an alternative to creating and running shortcuts via the **Start** button. To show or hide the shortcuts toolbar, use **Desktop > Shortcuts Toolbar**. To create and run shortcuts via the desktop **Shortcuts** toolbar, perform these steps:

1 Select statements from the Command History window, the Command Window, or an M-file.

2 Drag the selection to the desktop **Shortcuts** toolbar. The following illustration shows two statements being dragged from the Command Window.

Shortcuts toolbar—Drag statements to it to create a shortcut.
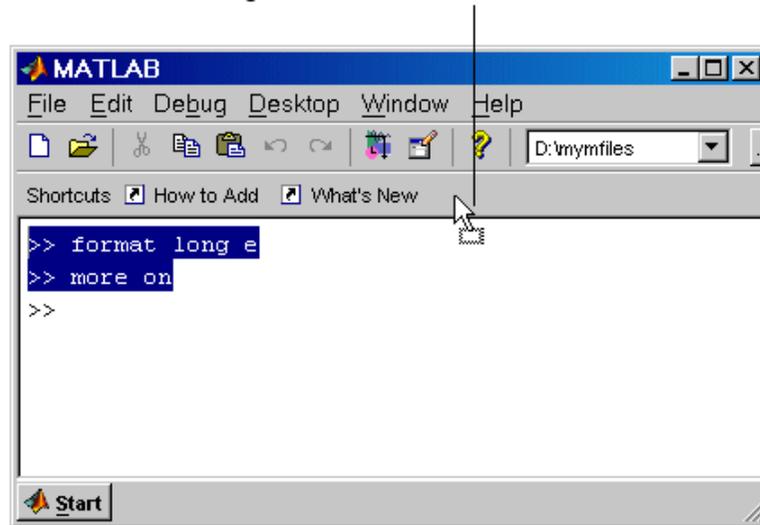
3 The Shortcut Editor dialog box appears. The **Callback** field contains the selected statements, which you can edit as needed. If prompts (>>) from the Command Window appear, note that MATLAB automatically removes them when you save the shortcut. The **Category** field is `Toolbar Shortcuts`, which you must retain in order for the shortcut to appear on the toolbar.

Provide the **Label**, select an **Icon**, and click **Save**.

The shortcut icon and label appear on the toolbar. If you have more shortcuts on the toolbar than can be displayed at once, use the drop-down list to access all of them. For more information, click the **Help** button in the Shortcut Editor dialog box.

4 Click the icon on the **Shortcuts** toolbar to run the shortcut. You can also run the shortcut from the **Start** button by selecting it in the `Toolbar Shortcuts` category.

Right-click a shortcut and then delete or edit it.

Right-click in the **Shortcuts** toolbar and then use the **Show Labels** item to hide shortcut labels and show only the icons.

Drag selected statements to the **Shortcuts** toolbar to create a new shortcut on the toolbar.

Click an item to run that shortcut.
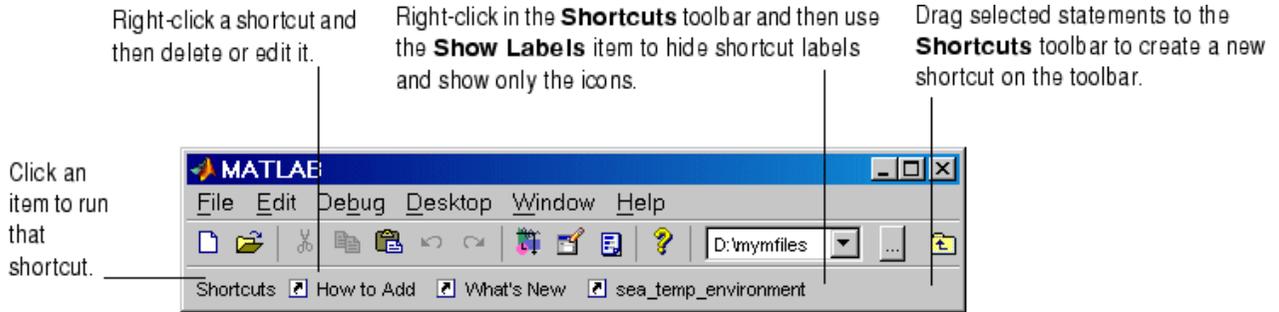
You can also add a shortcut to the desktop **Shortcuts** toolbar by right-clicking the toolbar and selecting **New Shortcut**. Complete the resulting Shortcut Editor dialog box. Assuming you maintain the `Toolbar Shortcuts` category, the shortcut appears on the toolbar. To change the order of the shortcuts on the toolbar, select **Start > Shortcuts > Organize Shortcuts** and move the shortcuts within the `Toolbar Shortcuts` category.

How to Add and What's New Shortcuts
The **Shortcuts** toolbar includes two shortcuts provided with MATLAB. The **How to Add** shortcut provides help about shortcuts and adding them to the **Shortcuts** toolbar. **What's New** displays the Release Notes documentation.

To remove the **How to Add** or **What's New** shortcut from the **Shortcuts** toolbar, choose a different category. For instructions, see "Organizing and Editing Shortcuts" on page 2-38.

If you do not want to keep these shortcuts, remove each one by right-clicking its toolbar shortcut button and selecting **Delete** from the context menu. Click **OK** in the confirmation dialog box to remove the shortcut.

Shortcut Labels on Toolbar
You can hide the shortcut labels on the toolbar. Right-click in the **Shortcuts** toolbar. From the context menu, select **Show Labels**, which clears the check mark next to the item. The shortcut icons appear on the toolbar without labels.

When you move the mouse over a shortcut icon, its label appears as a tooltip. To make labels display in the toolbar, right-click the toolbar and select **Show Labels**, which adds a check mark next to the item and displays the labels.

Organizing and Editing Shortcuts

To create categories for shortcuts, and to move, edit, and delete shortcuts, perform these steps:

1 Select **Shortcuts > Organize Shortcuts** from the **Start** button. Alternatively, access it via the shortcuts toolbar context menu.

The Shortcuts Organizer dialog box appears. When a shortcut category is selected in the dialog box, the **Edit Shortcut** button is replaced by the **Rename Category** button.

Move shortcuts and more. For example, drag a shortcut to another category.
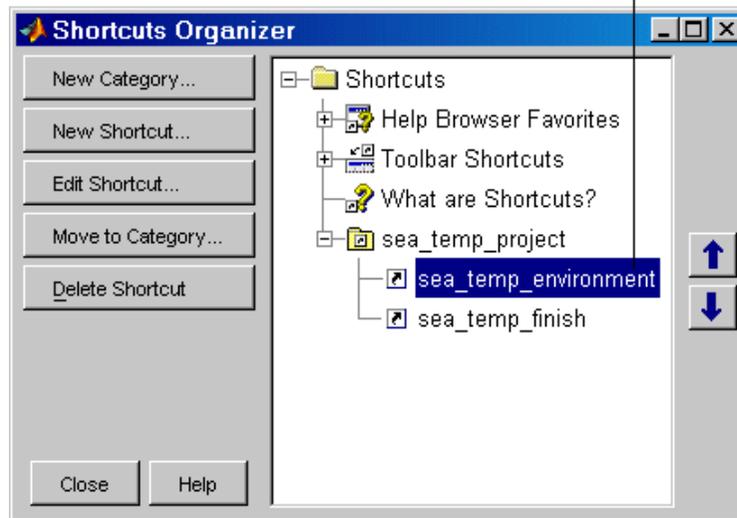
2 Use the buttons in the dialog box to edit and organize shortcuts and categories. You can also right-click an item and select an action from the context menu.

Changes take effect immediately.

3 Click **Close**.

For more information about using the Shortcuts Organizer dialog box, click the **Help** button.

Keyboard Shortcuts

Keyboard Shortcuts (Accelerators or Hot Keys) and Mnemonics

You can access many of the menu items using shortcut keys (sometimes called accelerators or hot keys) for your platform. For example, use the **Ctrl+X** shortcut to perform a cut on Windows platforms. Many of the menu items show the shortcuts. Additional standard shortcuts for your platform usually work but only one is listed with each menu item.

See additional shortcuts for the Command Window at "Keyboard Shortcuts in the Command Window" on page 3-26, and for the Editor/Debugger at "Keyboard Shortcuts in the Editor/Debugger" on page 6-59.

Instructions in the documentation specify shortcuts using the Windows **Ctrl+** key convention, but with Macintosh key bindings selected, you can use the Command key instead. On the Macintosh, to make full use of all keyboard shortcuts, you need to select the **Full Access** system preference for **Keyboard Shortcuts**.

You can also use mnemonics to access menu items and buttons, such as **Alt+F** to open the **File** menu. This is not supported on the Macintosh platform. Mnemonics are listed with the menu item or button. For example, on the **File** menu, the **F** in **File** is underlined, which indicates that **Alt+F** opens the menu. In the Profiler, the **R** in the **Run this code** toolbar field is underlined, indicating that **Alt+R** moves the cursor to this field.

Note that some versions of Windows do not automatically show the mnemonics on the menu. For example, you might need to hold down the **Alt** key while the tool is selected in order to see the mnemonics on the menus and buttons. In

Windows 2000, go to **Display Control Panel**, select **Effects**, and clear the item **Hide keyboard navigation indicators until I use the Alt key**. See your Windows documentation for details.

Following are some general shortcuts that are not listed on menu items.

Key	Result
Enter	The equivalent of double-clicking, **Enter** performs the default action for a selection. For example, press **Enter** while a statement in the Command History window is selected to run that statement in the Command Window. For buttons in tools and dialog boxes, **Enter** executes the default button (the button with a border around it). If there is no default button, press the **space bar** to execute the active button (the button with a dotted outline inside it). See "Default Button and Active Button (Button with Focus)" on page 2-42 for an illustration.
Esc (escape)	Cancels the current action. For example, if you select the **Edit** menu, the menu items display. Pressing **Esc** retracts the menu items. Pressing **Esc** in a dialog box is the same as selecting the **Cancel** button.
Tab	Advances to the next button or field in a tool or dialog box. In the Command Window, completes a statement if the tab completion preference is selected.
Space bar	For buttons in tools and dialog boxes, activates the active button. See "Default Button and Active Button (Button with Focus)" on page 2-42 for an illustration of selecting default and active buttons using keys.
+ or **-** or ***** on numeric keypad	Use these keys on the numeric keypad to expand and collapse items in tree views. The Help browser **Help Navigator** pane and the Command History window use tree views. Use **+** to expand the selected item, use **-** to collapse the selected item, and use ***** to recursively expand it, meaning open all items contained in the selected item.
Alt+S	Displays the **Start** button menu (except on Macintosh platforms).
Alt+Y	Provides access to the current directory field in the toolbar (except on Macintosh platforms).
Ctrl+Tab	Moves to the next open tool in the desktop, or to the next open group of tools tabbed together.

Key	Result
Ctrl+Shift+Tab	Moves to the previous open tool or group of tabbed tools in the desktop.
Ctrl+Page Down	Moves to the next tool within a group of tools tabbed together. In a group of documents, moves to next document.
Ctrl+Page Up	Moves to the previous tool within a group of tools tabbed together. In a group of documents, moves to previous document.
Ctrl+F6	Moves to the next tool or document (only for Windows and Solaris platforms).
Ctrl+Shift+F6	Moves to the previous tool or document (only for Windows and Solaris platforms).
Alt+F4	Closes the desktop, thereby quitting MATLAB. Or outside the desktop, closes the active window (except on Macintosh platforms).

For additional shortcuts available in the various desktop tools, see the documentation for each tool. For example, see "Keyboard Shortcuts in the Command Window" on page 3-26 and "Keyboard Shortcuts in the Editor/Debugger" on page 6-59.

Go To First Letter (Type Ahead) Feature in Desktop Tool Lists

In the Current Directory browser and Command History window, you can type a letter to move directly to the next item in the list that starts with the letter you typed. This is sometimes referred to as *type ahead*.

Default Button and Active Button (Button with Focus)

These illustrations demonstrate the default versus active button in a dialog box.

The default button has a border around it. Here, **Save** is the default button. Press the **Enter** key to execute the default button.

The active button (the button with focus) has a dotted outline inside it. Here, **Cancel** is the active button. Press the space bar to execute the active button.
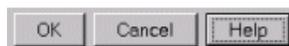
Here, the **Help** button is both the default button and the active button. In some cases, the default always changes to match the active button. You can press either **Enter** or the space bar to execute the **Help** button

Other Desktop Features

In this section...

Start Button for Accessing Tools

The MATLAB **Start** button provides easy access to tools, demos, and documentation for all your MathWorks products. From it, you can also create and run MATLAB shortcuts, which are groups of MATLAB statements.

Using the Start Button

1 Click the **Start** button to view a menu of product categories and desktop tools installed on your system. As an alternative, press **Alt+S** to view the **Start** button contents (except on Macintosh platforms). In the following illustration, MATLAB is selected.
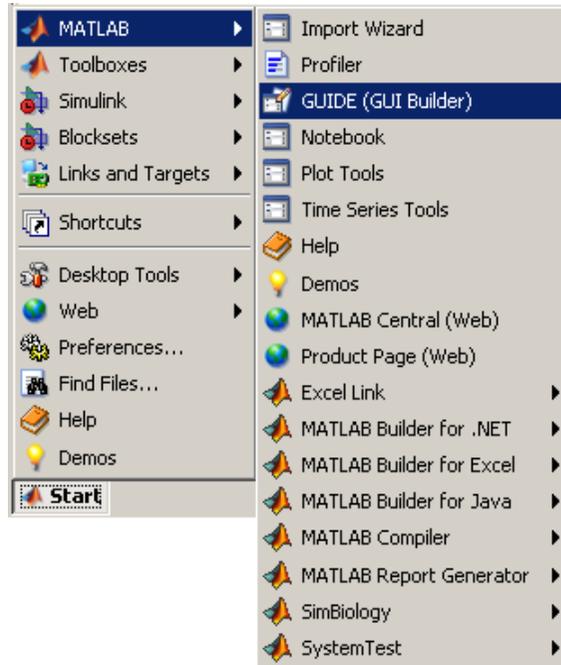
2 From the menu and submenu items, select an item to open it. Use the icons to quickly locate a type of product or tool — see the following description of icons.

For example, select **Start > MATLAB > GUIDE (GUI Builder)** to open that tool.

Icons in the Start Button. Icons help you quickly locate a particular type of product or tool. This table describes the action performed when you select an entry with one of these icons in the **Start** button.

Icon	Description of Action When Opened
	Documentation for that product opens in the Help browser.
	Demos for the product are listed in the Help browser **Demos** pane.
	Selected tool opens.

Icon	Description of Action When Opened
	Block library opens.
	Document opens in your system Web browser.

Customizing the Start Button

You can add your own toolboxes to the **Start** button. Select **Start > Desktop Tools > View Source Files** to open the Start Button Configuration Files dialog box. For more information, click the **Help** button in the dialog box.

Menus and Context Menus

Merged Menus

When you use a tool in the desktop, its menu appears at the top of the desktop. When you work in a different tool in the desktop, you still use the menu at the top of the desktop, but the menu content changes to support that tool. When you undock a tool from the desktop, access its menu at the top of the undocked tool.

Context Menus

Many of the features in MATLAB desktop tools are available from context menus, also known as pop-up or right-click menus. To access a context menu, right-click a selection or an area, or press **Ctrl+Shift+F10**. The context menu for the selection or tool appears, presenting the available actions. For example, following is the context menu for a selection in the Command History window.

If a context menu does not appear, try right-clicking in a different part of the tool. When a context menu item is gray, the item does not apply to the current selection or area.

Access context (pop-up) menus by right-clicking a selection or any area in a tool.
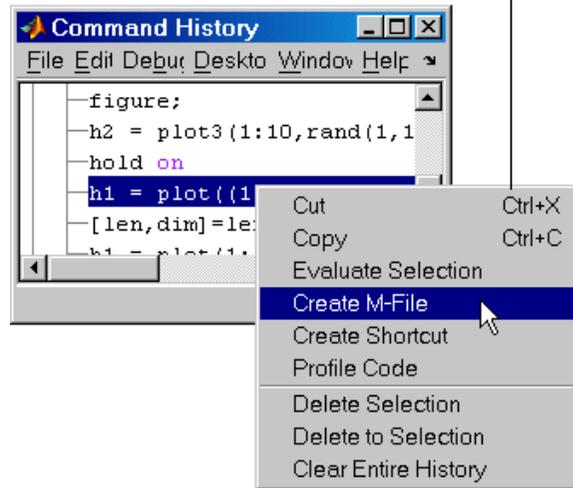
Toolbars

The toolbar in the desktop provides easy access to frequently used operations. Position the pointer over a button for a second or two and a tooltip appears that describes the item.
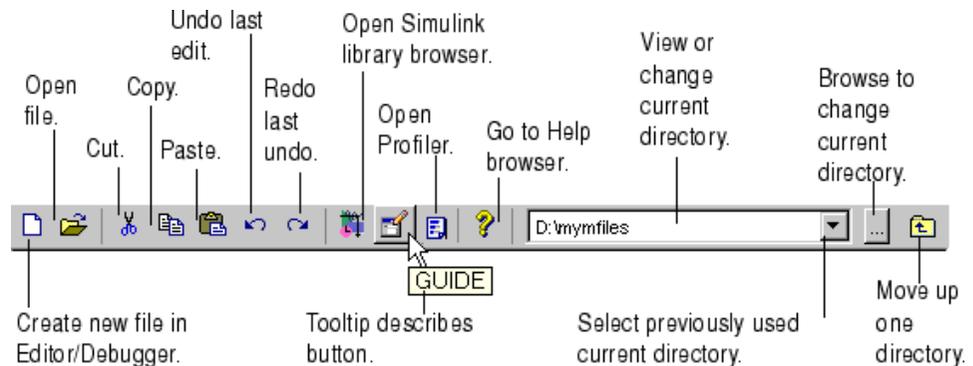
Some tools also have their own toolbars, which are located within the tool's own window. For example, the Current Directory browser has its own toolbar. When you undock one of these tools, the undocked tool includes the toolbar.

To hide a toolbar, or to show it again after hiding it, use the appropriate toolbar item in the **Desktop** menu. As an alternative, right-click a toolbar or menu bar and select a toolbar from the context menu to hide or show it.

For figure windows, use the toolbar item in its **View** menu.

Current Directory Field

The current directory field in the desktop toolbar shows the MATLAB current working directory. You can change the current directory using this field and any of these methods:

- Type the new current directory directly in the field.

- Use the drop-down list to change to a previously used current directory. To specify the number of entries maintained each session, use the **History** preference you access via **File > Preferences > Current Directory**.

- Use the Browse for folder button **...** to select a new current directory.

- Use the Go Up One Level button 🖿 to move the current directory up one level.

The same current directory field also appears in the Current Directory browser when the Current Directory browser is undocked from the desktop. Use the Current Directory browser to perform many additional file operations. For more information, see "File Management Operations" on page 5-35.
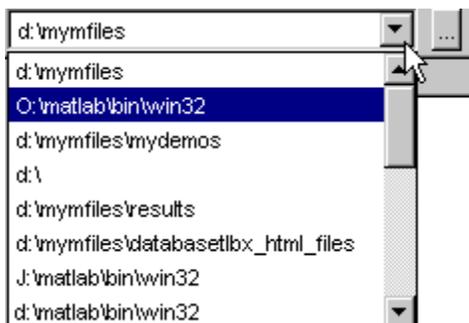
Status Bar

Along the bottom of the desktop is the status bar. It displays messages, such as when MATLAB is busy executing statements or when the Profiler is on. Some tools, such as the Editor/Debugger, display additional status information, such as the current line number. Not all status information appears on the status bar — many MATLAB functions and tools provide status information that is not reported to the status bar.

You can construct your own functions to provide status information. See the `timer` function, and search for other specific terms describing the status of interest.

Sizing, Arranging, and Sorting Columns in Tools

Some desktop tools present information in columns, such as the Current Directory browser.

To change the column width, drag the separator bar between two column headings in a tool. When a column is too narrow to show all the information in it, position the pointer over an item and the full value for that item displays like a tooltip.

To rearrange the columns in a tool, drag the column header to a different position. To sort the information by a particular column, click the column header. For example, in the Current Directory browser, click the **Last Modified** date to sort the items in date order. Some columns also allow you to reverse the sort order by clicking the column header again. A small gray arrow in the header indicates the current sort order — for example, an up arrow in the **Last Modified Date** column header indicates an ascending sort order, meaning the oldest files are at the top of the list.

Click a column heading to sort by that column. Click again to reverse the sort order.

To reorder columns, drag a column header, like **Description**, to another position.

Hold the pointer over a field to see the entire value for that column.
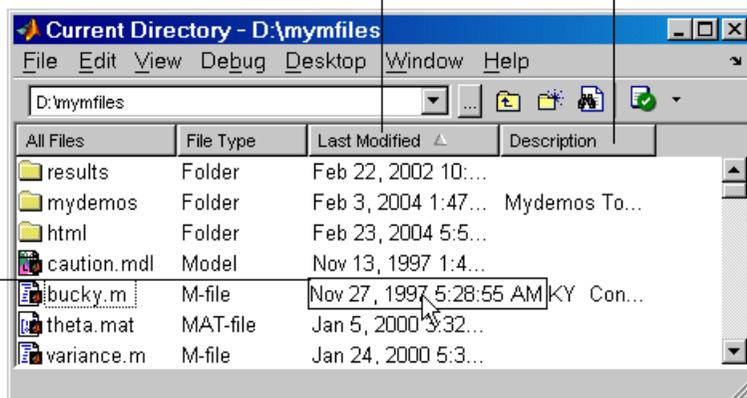
Selecting Multiple Items

In many desktop tools, you can select multiple items and then select an action to perform on all the selected items. Select multiple items using the standard practices for your platform.

For example, if your platform is Windows, do the following to select multiple items:

1 Click the first item you want to select.

2 Hold the **Ctrl** key and then click the next item you want to select. Repeat this step until you have selected all the items you want. To select contiguous items, select the first item, hold the **Shift** key, and then select the last item.

Now you can perform an action on the selected items, such as delete.

To clear one of multiple selected items, use **Ctrl**+click. To clear all selected items, click outside of the selection.

Cut, Copy, Paste, and Move

You can cut and copy a selection from a desktop tool to the clipboard and then paste it from the clipboard into another tool or application. Use the **Edit** menu, toolbar, context menus, or standard keyboard shortcuts. For example, you can copy a selection of statements from the Command History window and paste them into some MATLAB desktop tools.

Use **Paste** to move items copied to the clipboard from other applications. The **Paste to Workspace** item in the **Edit** menu opens the selection on the clipboard in the Import Wizard. You can use this to copy data from another application, such as Excel, into MATLAB. For details, see the "Using the Import Wizard".

When editing in the Command Window and Editor/Debugger, to move text to a new location, select the text and drag it. To copy text, press **Ctrl** and drag the selected text to the new location.

To undo the most recent cut, copy, or paste command, select **Undo** from the **Edit** menu. Use **Redo** to reverse the **Undo**. For some tools, you can undo multiple times in succession.

See also the `clipboard` function.

Drag and Drop

You can also move or copy a selection from one tool to another by dragging the selection. For example, make a selection in the Command History window and drag it to the Command Window, which pastes it there. Edit the lines in the Command Window, if needed, and then press the **Enter** key to run the lines from the Command Window.

Another example is to drag a filename from the Current Directory browser to the Editor/Debugger to open that file in the Editor/Debugger. If you drag editable text, for example, text in the Editor/Debugger, the text is cut rather than copied. Use **Ctrl** and drag to copy rather than cut editable text.

On Windows platforms, you can drag items from external applications into MATLAB. For example, dragging text from a Microsoft Word document into the Editor/Debugger cuts and pastes it into the open file. Dragging an M-file from Windows Explorer to the Command Window runs the file. Similarly, you

can drag selections from desktop tools to other applications. For example, you can drag text from the Editor/Debugger to Microsoft Word.

Macintosh Differences in the Desktop

MATLAB on the Macintosh platform sometimes uses Macintosh GUI conventions, which might be different from what is stated in the MATLAB documentation, but the intended action should be clear. For example, if you select **File > Save** on the Macintosh, the Save dialog box that appears presents the options **Don't Save** and **Save**. On Windows and UNIX platforms, the Save dialog box presents the options **Yes**, **No**, and **Cancel**.

The standard Macintosh mouse is a single-button device. Other platforms use a mouse with more than one button. MATLAB takes advantage of these buttons. The documentation does not usually present the equivalent Macintosh instruction. When the documentation instruction is right-click, use **Ctrl**+click on the Macintosh. When the documentation instruction is middle-click, use **Command**+click on the Macintosh.

Printing and Page Setup Options for Desktop Tools

You can print from all desktop tools except the Current Directory browser, but there are some differences in usage.

To print, select **File > Print** from the tool. A Print dialog box opens. The **Properties** button in the Print dialog box is enabled for the Web and Help browsers and the Profiler, but is not enabled for the other desktop tools.

To specify standard page setup options for your platform when you print from the Command History, Workspace browser, and Array Editor, select **File > Page Setup**. A standard page setup dialog box for your platform opens.

MATLAB provides special page setup options for printing from the Command Window and Editor/Debugger. The setup options are essentially the same for both tools, with minor variations. This section covers their use:

- "Specifying Page Setup Options" on page 2-53
- "Layout Options for Page Setup" on page 2-54
- "Header Options for Page Setup" on page 2-54

• "Fonts Options for Page Setup" on page 2-54

Specifying Page Setup Options

To specify page setup options, perform these steps:

1 In the tool you want to print from, for example, the Command Window, select **File > Page Setup**.

The Page Setup dialog box opens for that tool.
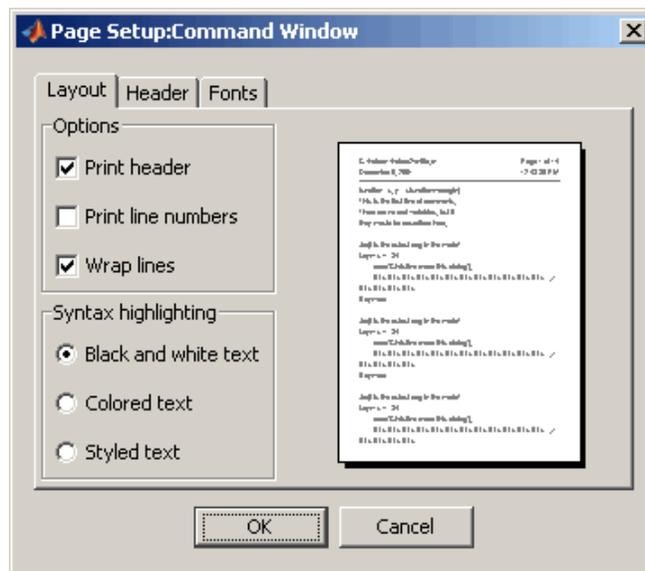
2 Click the **Layout**, **Header**, or **Fonts** tab in the dialog box and set those options for that tool, as detailed in subsequent sections.

3 Click **OK**.

4 After specifying the options, select **File > Print** in the tool you want to print from, for example, the Command Window.

The contents from the tool are printed, using the options you specified in Page Setup.

Layout Options for Page Setup

You can specify the following layout options. A preview area shows you the effects of your selections:

- **Print header** — Print the header specified in the **Header** pane.

- **Print line numbers** — Print line numbers.

- **Wrap lines** — Wrap any lines that are longer than the printed page width.

- **Syntax highlighting** — For keywords and comments that are highlighted in the Command Window, specify how they are to appear in print. Options are black and white text (that is, no highlighting), colored text (for use with a color printer), or styled text. For styled text, keywords appear in bold, comments appear in italics, and all other text appears in the normal style. Only keywords and comments you input in the Command Window are highlighted; output is not highlighted.

Header Options for Page Setup

If you want to print a header, select the **Layout** tab and then select **Print header**. Then select the **Header** tab and specify how the elements of the header are to appear. A preview area shows you the effects of your selections:

- **Page number** — Format for the page number, for example `# of n`

- **Border** — Border style for the header, for example, `Shaded box`

- **Layout** — Layout style for the header. For example, `Standard one line` includes the date, time, and page number all on one line

Fonts Options for Page Setup

Specify the font to be used for the printed contents:

1 From **Choose font**, select the element, either `Body` or `Header`, where `Body` text is everything except the `Header`.

2 Select the font to use for that element. For example, select **Use Command Window font** for `Body` text if you want the printed text to be the same as the font that appears in the Command Window. This is the font specified in **File > Preferences > Fonts > Custom** for the `Command Window`.

3 Repeat for the other element. If you did not select **Print header** on the **Layout** pane, you do not need to specify the Header font. As an example, for Header text, select **Use custom font** and then specify the font characteristics — type, style, and size. After you specify a custom font, the **Sample** area shows how the font will look.

Web Browser

Some tools in MATLAB and related products display HTML documents in the MATLAB Web Browser. For example, after using the Editor/Debugger's cell features to publish an M-file to HTML, you view the HTML file in the MATLAB Web Browser. Because the MATLAB Web Browser is a desktop tool, you can dock it and perform other desktop operations on it.

Example of Web browser displaying results of M-file published to HTML format.
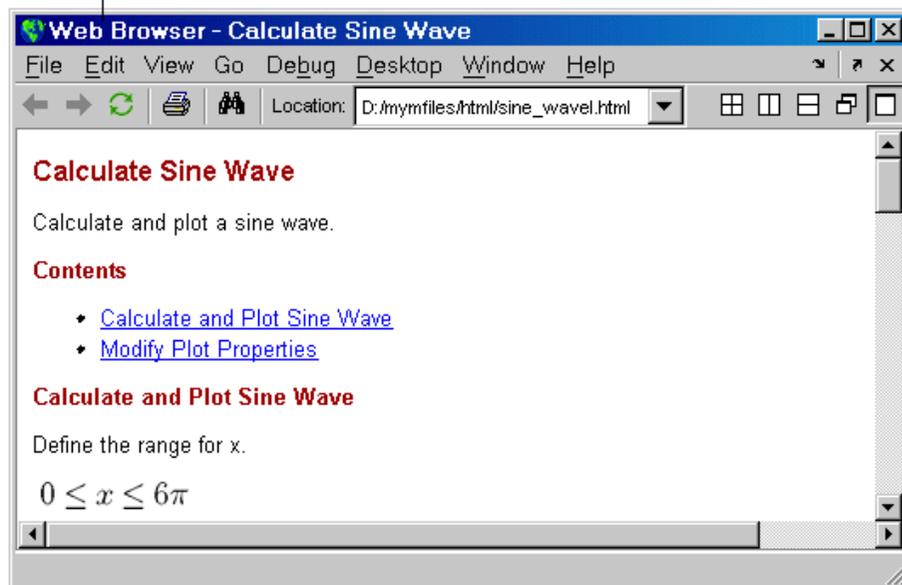
To display an HTML document in the Web Browser, double-click the document name in the Current Directory browser or use the web function. The web function supports arguments that display documents in your system browser, for example, Netscape, or in the Help browser.

The toolbar buttons and menu items in the Web Browser are similar to those found in the Help browser display pane. For more information, see "Viewing Documentation in the Help Browser" on page 4-26.

One feature of the Web Browser not found in the Help browser is the **Location** field. In the Web Browser, type a URL in the field to display that Web page.

Like any Web browser, the MATLAB Web Browser might not support all of the HTML or related features used in a particular Web site or HTML page. For example, the MATLAB Web Browser does not support the display of .bmp (bitmap) image files. Instead use .gif or .jpeg formats for image files in HTML pages. As another example, it does not support HTML pages you generate directly from Microsoft Word and PowerPoint.

Internet Connection and Fonts for Web Browser — Web Preferences

To specify a proxy server to connect from the MATLAB Web Browser to the Internet, use Web preferences. You might need to specify this preference if you have a firewall, for example. If you have a firewall and do not specify the proxy settings, links from the Web Browser to URLs will not work.

Select **File > Preferences > Web**. By default, the check box **Use a proxy server to connect to the Internet** is not selected. This is for when you have a direct connection to the Internet.

To specify a proxy server, select the check box and specify the **Proxy host** and **Proxy port**. See your system administrator for the information you need to specify the proxy settings. As an example, 172.16.10.8 illustrates the format for host, and 3128 is the type of value you enter for port.

Fonts for Web Browser. To modify the font used in the Web Browser, select **File > Preferences > Fonts**. The Web Browser uses the font settings you specify for HTML Proportional Text tool. For more information about setting fonts, click the **Help** button in the preference pane for **Fonts**.

Accessing The MathWorks on the Web

You can access popular MathWorks Web pages from the MATLAB desktop. Select one of the following items from the **Help > Web Resources** menu.

For most items, the selected Web page then opens in your default system Web browser, for example, Netscape:

- **The MathWorks Web Site** — Home page of the MathWorks Web site (`http://www.mathworks.com`).

- **Products & Services** — MathWorks Products and Services page (`http://www.mathworks.com/products/`) with information about the full family of products.

- **Support** — MathWorks Support page (`http://www.mathworks.com/support`) where you can look for solutions to problems you are having, or report new problems.

- **MathWorks Account**

  - **Login in or Create Account** — Login page for MathWorks Account (`http://www.mathworks.com/accesslogin/`). If you are registered, your main account page displays. Otherwise, you are directed to a page where you register online. Registration allows you to view your product registration and license information and helps you stay up to date on the latest MATLAB developments.

  - **Get Passcodes and Manage Licenses** — If you have a MathWorks Account, displays your Licenses page.

  - **Get Product Trials** — If you have a MathWorks Account, provides access to trial versions of products.

- **MATLAB Central** — MATLAB Central Web site (`http://www.mathworks.com/matlabcentral/`) for the MATLAB user community. It includes MATLAB contest entries and results, a MATLAB screen saver, and these technical resources:

  - **MATLAB File Exchange** — Code library of files contributed by MathWorks customers and employees, available for free download and use with MathWorks products.

  - **MATLAB Newsgroup Access** — Provides access to the Usenet newsgroup for MATLAB and related products, `comp.soft-sys.matlab`, where you can post and answer questions, as well as view the archives.

- **MATLAB Newsletters** — Access to online versions of News and Notes and MATLAB Digest. News and Notes is published twice a year and contains feature articles, technical notes, and product information for MATLAB

users. MATLAB Digest, an electronic bulletin consisting of technical notes, solutions, and timely announcements to the user community, is issued more frequently. See `http://www.mathworks.com/company/newsletters`.

Check for Updates

This features allows you to easily determine if more recent versions of your MathWorks products are available. Select **Help > Check for Updates**. A dialog box appears, listing the version numbers of all MathWorks products installed on your system. Click **Check for Updates** in the dialog box, which accesses the MathWorks Web site and reports back for each product if a newer version is available or if your version is the latest.

Terms of Use and Patents

Access the terms of use and patent information for MathWorks products.

Preferences

In this section...
"Setting Preferences" on page 2-59
"Summary of Preferences" on page 2-60
"Preferences File — matlab.prf" on page 2-61

Setting Preferences

Use preferences to specify options for MATLAB tools, as follows:

1 Select **File > Preferences**.

2 In the left pane of the Preferences dialog box, preferences appear for MATLAB tools as well as for any other MathWorks products installed on your system.

Choose a tool and click the **+** to display more preferences for that tool. From the expanded list, select the entry you want. The right pane shows the preferences for that item.

3 Change settings. Click **Apply** or **OK** to set the preferences. Preferences take effect immediately. They remain persistent across MATLAB sessions.

Note that some tools allow you to control these settings from within the tool without setting a preference. Use that method if you only want the change to apply to the current session.

Function Alternative

Open the Preferences dialog box using the `preferences` function.

Summary of Preferences

Preference	What You Can Specify
General Preferences	Toolbox path caching, figure window printing, `delete` function behavior, MAT-file save formats, confirmation dialogs, source control, and multithreaded computation.
Keyboard	Key bindings, tab completion, and delimiter matching for the Command Window and the Editor/Debugger.
Fonts	Font type, style, and size for desktop tools. Customize for any tool.
Colors	Colors for text, background, syntax highlighting, and hyperlinks in desktop tools.
M-Lint	Show or hide M-Lint messages in the Editor/Debugger M-Lint automatic code analyzer and in the M-Lint Code Check Report.
Command Window	Numeric format and display, accessibility, and tab size.
Command History	Display, filtering, and saving.
Editor/Debugger	Editor type, startup options, display, tab size and indenting, language, including M-Lint messages, publishing, and autosave.
Help	Product filter and synchronization.
Web	Internet proxy server settings.
Current Directory	Number of entries in history and display options.
Array Editor	Numeric format, use of **Enter** key, and decimal separator.
Workspace	Statistical calculation options.
GUIDE	Display options for GUI-building tool.
Time Series Tools	Property Editor dialog and x-axes warning dialog.
Figure Copy Template	Application, text, line, uicontrols, axis, format, background color, and size.
Other products	Preferences for other installed MathWorks products.

Preferences File — matlab.prf

Preferences are stored in a preferences file, `matlab.prf`. Type `prefdir` in the Command Window to see the full pathname for the preferences directory that contains `matlab.prf`. The preference directory also contains related files.

On Macintosh platforms, the directory might be in a hidden folder, for example, `myname/.matlab/R2007b`. To access the directory, select **Go > Go to Folder** in the Mac OS Finder. In the resulting dialog box, type the path returned by `prefdir` and press **Enter**.

The `matlab.prf` file is loaded when MATLAB starts and is overwritten when you close MATLAB.

The exact name of the preferences directory that MATLAB uses depends on the release. When you install a new version of MATLAB, MATLAB tries to use your existing preferences from the previous version, where possible. For more information on the preference directory filename and the preference migration process, see the reference page for `prefdir`.

Fonts Preferences for Desktop Tools

In this section...
"Setting Desktop Fonts" on page 2-62
"Desktop Code Font and Desktop Text Font" on page 2-63
"Custom Fonts Preferences" on page 2-67
"Changing the Font — Example" on page 2-68
"Antialiasing for Desktop Fonts on Linux/UNIX" on page 2-69
"Making Fonts Available to MATLAB" on page 2-69

Setting Desktop Fonts

Use desktop font preferences to specify the font characteristics for MATLAB desktop tools. The font characteristics are

- Name (also called family or type), for example, select SansSerif
- Style, for example, select bold
- Size in points, for example, type 11 points

Select **File > Preferences > Fonts** to set fonts for desktop tools. You can specify the font to be used by all tools that primarily display code such as the Command Window, and specify the font to be used by all other desktop tools. Or you can separately specify the font for any desktop tool.

Select the font characteristics from the lists shown. For font size, not all entries are shown. You can type in a size, including one not shown.

You can set some font options differently for printing — see "Printing and Page Setup Options for Desktop Tools" on page 2-52.

For information about making additional fonts available to MATLAB, see "Making Fonts Available to MATLAB" on page 2-69.

With the code and text font styles, you can easily apply the same font to all desktop tools that display code or text, respectively.

Type a font size if it is not in the list.

Preferences

☐ General
☐ Keyboard
☐ Fonts
　　Custom
　Colors
　Command Window
　Command History
☐ Editor/Debugger
　Help
　Web
　Current Directory
　Array Editor
　GUIDE
　Time Series Tools
☐ Figure Copy Template
　Report Generator
　SystemTest
　Instrument Control
☐ Virtual Reality
☐ Simulink

Fonts Preferences

Desktop code font
Currently used by: Command Window, Command History, Editor

| Monospaced | Plain | 10 |

Sample

The quick brown fox jumps over the lazy dog. 1234567890

Desktop text font
Currently used by: Help Navigator, HTML Proportional Text, Current Directory, Workspace, Array Editor

| SansSerif | Plain | 10 |

Sample

The quick brown fox jumps over the lazy dog. 1234567890

OK Cancel Apply Help

Desktop Code Font and Desktop Text Font

You specify separate font characteristics for tools that primarily display code (**Desktop code font**), such as the Command Window, and tools that primarily display text (**Desktop text font**), such as the Current Directory

browser. Many users prefer that code display in a monospace font to provide better alignment, and prefer a more narrow font style for text information. With the desktop code font preference, you set just one preference to apply a monospace style to all tools that display code (except the Help and Web Browsers). Similarly, you can set just one preference to apply a text font to all desktop tools that display text.

The following illustrations show how the Editor/Debugger would look using a monospace font and a proportional font. Note that a monospace font is useful when you care about alignment, but a proportional font uses less space.

With a monospaced font, all characters are the same width. Here, the font is 10 pt. Monospace. Note the 10th character in each line aligns with the Editor/Debugger's right-hand text limit, which is set to column 10.
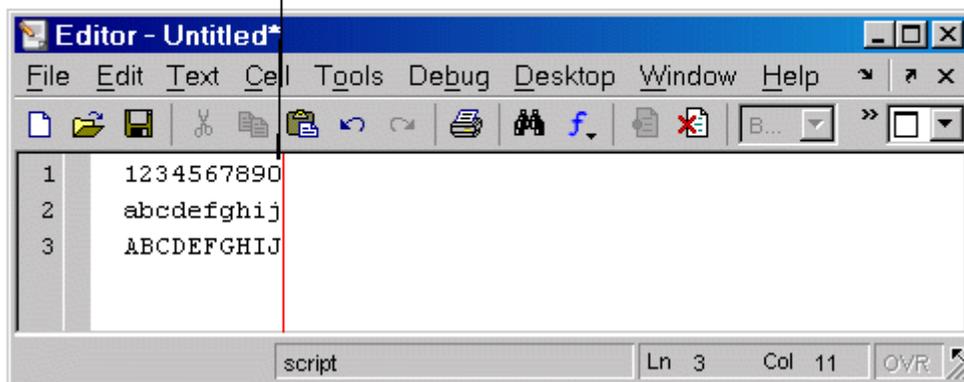
With a proportional font, characters are different widths. Here, the font is 10 pt. SansSerif. Each line contains 10 characters but the length of each line differs. The Editor/Debugger's right-hand text limit is not relevant.
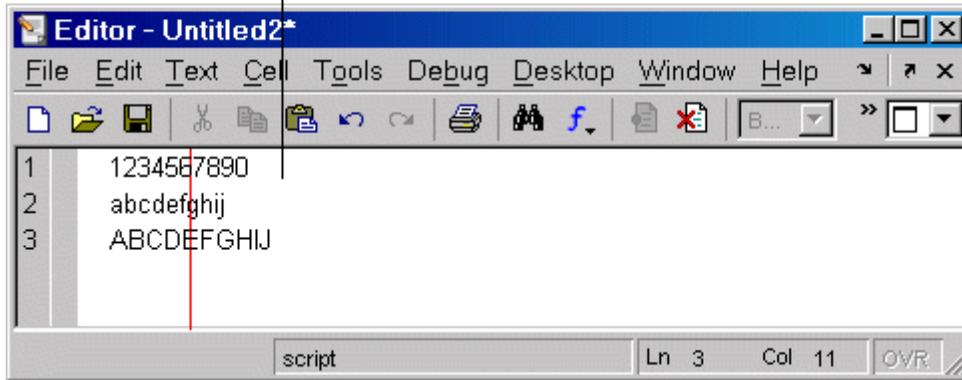
Default Font Settings

Default settings are listed in the following table. Note that Lucida Console approximates the fixedsys font available in earlier versions of MATLAB.

Font Type	Default Characteristics and Sample	Tools Using Font Type by Default
Desktop code font	Monospaced, Plain, 10 point `Sample code font`	• Command History • Command Window • Editor/Debugger (which also applies to the Shortcuts Editor)
Desktop text font	SansSerif, Plain, 10 point `Sample text font`	• Array Editor • Current Directory browser (which also applies to the Path browser) • Help Navigator • HTML Proportional Text. This is the font used for noncode text in the Web browser (including, for example, HTML reports generated from cell publishing), Profiler, and Help browser display pane. While you can select the font name, you cannot change the font style (for example, to bold or italic) for HTML Proportional Text. Changes to size affect noncode and code text. • Workspace browser

When you change a font characteristic for **Desktop code font**, the characteristic takes effect for all tools that use the desktop code font. The same is true when you change a font characteristic for **Desktop text font**.

After changing a characteristic, a sample in the dialog box shows how it will look. Click **Apply** or **OK** to make the change take effect in the desktop tools.

See Also

"Preferences" on page 2-59

Custom Fonts Preferences

If you do not want to use the current settings for "Desktop Code Font and Desktop Text Font" on page 2-63, you can specify that a tool use the code font, the text font, or a different font. Select **File > Preferences > Fonts**. Click **+** and select **Custom**. The **Fonts Custom Preferences** pane appears.

Use custom fonts preferences to specify the tools that use the code style font and the tools that use the text style font. You can also apply a custom font to any tool.
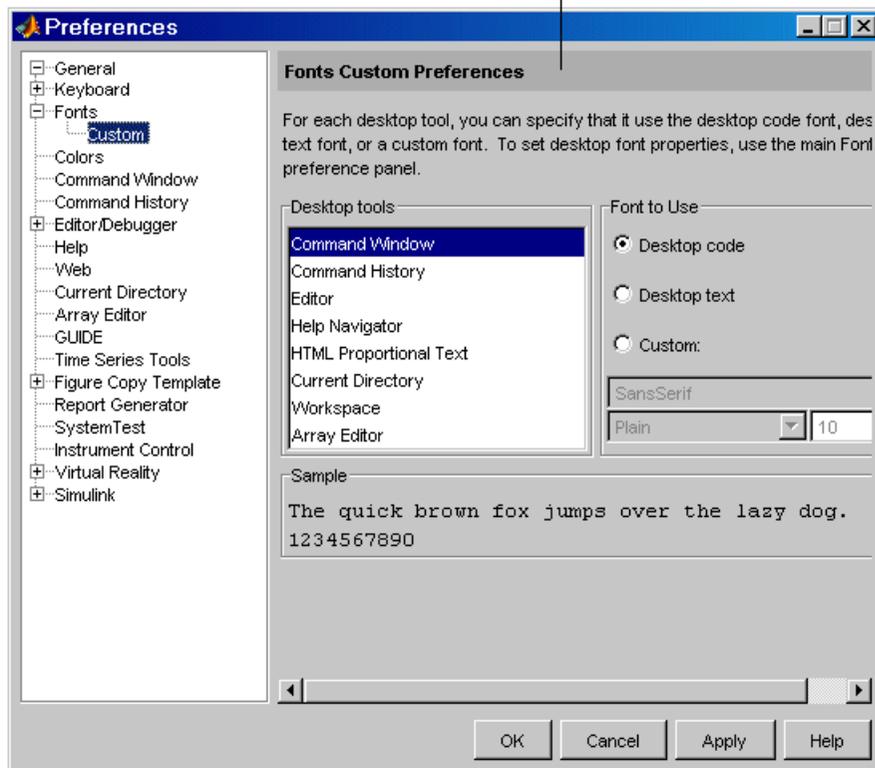
Select a tool from the **Desktop tools** list. The type of font it uses, code or text, appears under **Font to Use**. In the illustration shown, the Command Window uses the **Desktop code font**, which is defined in the **Fonts** pane as described in the previous section.

To change the font characteristics the selected tool uses, select a different radio button. For **Custom**, you then specify the font characteristics for that tool.

Changing the Font — Example

This example changes the default settings (see "Default Font Settings" on page 2-65) for the desktop code font, changes the Command History font preference so that it uses the desktop text font instead of the code font, and specifies a custom font for the Current Directory browser:

1 Change the characteristics for the desktop code font. On the **Fonts** pane, set the **Desktop code font** to Times New Roman, Plain, 14 point. Use the default for the **Desktop text font**, SansSerif, Plain, 10 point. Click **Apply**.

2 Make the Command History window use the desktop text font. Select **Fonts**, click **+**, select **Custom**, and then select Command History from **Desktop tools**. Select the **Desktop text** radio button.

3 Apply a custom font to the Current Directory browser. Select Current Directory from **Desktop tools.** Select the **Custom** radio button. Select Arial Narrow and Plain, and type 11 in the size field. Click **OK**.

The following table details the results of the changes.

Tool	Font Type	Font Characteristics
Command Window	Desktop code	Times New Roman, Plain, 14 point
Command History	Desktop text	SansSerif, Plain, 10 point
Editor/Debugger	Desktop code	Times New Roman, Plain, 14 point
Help Navigator	Desktop text	SansSerif, Plain, 10 point
HTML Proportional Text	Desktop text	SansSerif, Plain, 10 point
Current Directory	Custom	Arial Narrow, Plain, 11 point

Tool	Font Type	Font Characteristics
Workspace	Desktop text	SansSerif, Plain, 10 point
Array Editor	Desktop text	SansSerif, Plain, 10 point

See Also

For help about how MATLAB stores preferences and help for other preferences, see "Preferences" on page 2-59.

Antialiasing for Desktop Fonts on Linux/UNIX

To give the desktop a smoother appearance in Linux/UNIX, select the antialiasing preference on the **Preference > Fonts** pane. The preference apply to all fonts.

Note The antialiasing option is not necessary on Windows or Mac, because MATLAB follows the operating system's font settings on these platforms.

Making Fonts Available to MATLAB

On Windows platforms, desktop components (such as the Command Window and Workspace browser), figure windows, and uicontrols support only TrueType and OpenType fonts. Some graphics objects can render bitmapped fonts as well, such as `xlabel`, `ylabel`, `title`, and `text`.

To make a new compatible font available to MATLAB, install the font by selecting **Start > Control Panel > Fonts** in the Windows desktop, and then selecting **File > Install New Font**. Restart MATLAB so that it can use the font.

Colors Preferences for Desktop Tools

In this section...
"Setting Colors Used in Desktop Tools" on page 2-70
"Desktop Tool Colors" on page 2-72
"Syntax Highlighting Colors" on page 2-73
"Other Colors" on page 2-75
"See Also" on page 2-75

Setting Colors Used in Desktop Tools

Desktop color preferences specify the colors used in MATLAB desktop tools and the colors that convey syntax highlighting. Select **File > Preferences > Colors** to set color preferences for desktop tools. You can set some color options differently for printing — see "Printing and Page Setup Options for Desktop Tools" on page 2-52.

To set colors for text and the background, clear the **Use system colors** check box and then select colors from the palettes.

Specify the color of hyperlinks in the Command Window and Help browser Index pane.

Preferences

- General
- Keyboard
- Fonts
- Colors
- M-Lint
- Command Window
- Command History
- Editor/Debugger
- Help
- Web
- Current Directory
- Array Editor
- Workspace
- GUIDE
- Time Series Tools
- Figure Copy Template

Colors Preferences

Desktop tool colors

☑ Use system colors

Text   Background

M-file syntax highlighting colors

Keywords   Comments

Strings   Unterminated strings

System commands   Errors

Sample

```
% create a file for output
!touch testFile.txt
fid = fopen('testFile.txt', 'w');
for i=1:10
    fprintf(fid,'%6.2f \n, i);
end
```

Other colors

Hyperlinks   ☑ M-Lint autofix highlight

Restore Default Colors

OK   Cancel   Apply   Help

Desktop Tool Colors

Use **Desktop tool colors** to change the color of the text and background in the desktop tools. The colors also apply to the Import Wizard. The colors do not apply to the HTML display pane nor to the Web Browser.

Select the check box **Use system colors** if you want the desktop to use the same text and background colors that your platform (for example, Windows) uses for other applications.

To specify different text and background colors, follow these steps:

1 Clear the **Use system colors** check box.

2 Click the arrow next to the **Text** color and choose a new color from the palette shown.

When you choose a color, the **Sample** area in the dialog box updates to show you how it will look.

3 Click the arrow next to the **Background** color and choose a new color.

If you use a gray background color, a selection in an inactive window will not be visible.

4 Click **Apply** or **OK** to see the changes in the desktop tools.

Click **Restore Default Colors** to return to the default settings for desktop tool colors, as well as for syntax highlighting colors.

The following illustration shows how the Current Directory browser looks with blue-green text and a beige background. These colors are only discernible in the online version of this documentation.

Background color   Text color
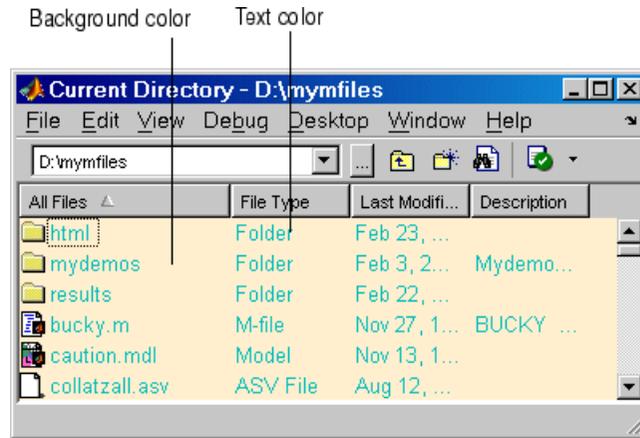
Gray Background Color

For some UNIX platforms, there is a gray background color for desktop tools, such as the Editor/Debugger. This occurs when the preference for **Desktop tool colors** is set to **Use system colors**, and the system's window manager uses gray as the background color default. To change the color, clear the check box for **Use system colors** and then select a new **Background** color from the palette.

Syntax Highlighting Colors

In the Command Window, Command History, Editor/Debugger, and Shortcuts callback area, MATLAB conveys syntax information via different colors to help you easily identify elements, such as if/else statements. This is known as syntax highlighting.

In the Command Window, only the input you type is highlighted; output from running MATLAB functions is not highlighted. In the Editor/Debugger, you can specify syntax highlighting preferences for use with files in M, C/C++, Java, and HTML. For details, click the **Help** button in the Preferences dialog box for the Editor/Debugger to see Language Preferences in the online documentation.

Use preferences to specify the syntax highlighting colors. When you choose a color, the **Sample** area in the dialog box updates to show you how it will look.
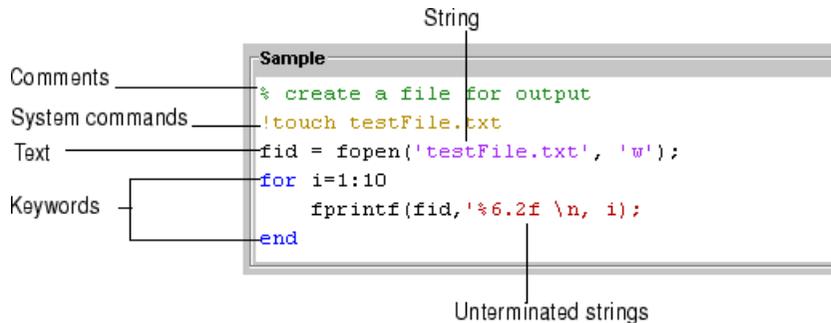
The default colors are listed here:

- **Keywords** — Flow control functions, such as `for` and `if`, as well as the continuation ellipsis (`...`), are colored blue.

- **Comments** — All lines beginning with a `%`, designating the lines as comments in MATLAB, are colored green. Similarly, the block comment symbols, `%{` and `%}`, as well as the code in between, appear in green. Text following the continuation ellipsis on a line is also green because it is a comment.

- **Strings** — Type a string and it is colored maroon. When you complete the string with the closing quotation mark (`'`), it becomes purple. Note that for functions you enter using command syntax instead of function syntax, the arguments are highlighted as strings. This is to alert you that in command notation, variables are passed as literal strings rather than as their values. For more information, see "MATLAB Command Syntax" in the MATLAB Programming documentation.

- **Unterminated strings** — A single quote without a matching single quote, and whatever follows the quote, are colored maroon. This might alert you to a possible error.

- **System commands** — Commands such as `!` (shell escape) are colored gold.

- **Errors** — Error text that appears after you run code, including any hyperlinks, is colored red.

Click **Restore Default Colors** to return to the default settings for syntax highlighting colors and desktop tool colors.

Other Colors

Specify the color for **Hyperlinks**, which applies to links in the Command Window and Help browser **Index** pane. If you use a dark background color for those tools, be sure to use a light or other contrasting color for hyperlinks so that you can see them.

With the **M-Lint autofix highlight** preference selected, code that M-Lint can automatically correct is highlighted in the Editor/Debugger. Use the palette to change the highlight color. For more information, see "M-Lint Automatic Code Analyzer in the Editor/Debugger" on page 6-88.

See Also

For information about other preferences and how MATLAB stores preferences, see "Preferences" on page 2-59.

General Preferences for MATLAB

In this section...
"Setting General Preferences for MATLAB" on page 2-76
"Default Behavior of the Delete Function" on page 2-78
"MAT-Files Preferences" on page 2-79
"Confirmation Dialogs Preferences" on page 2-81
"Multithreading Preferences" on page 2-84

Setting General Preferences for MATLAB

Select **File > Preferences > General** from any desktop tool to access **General Preferences**.
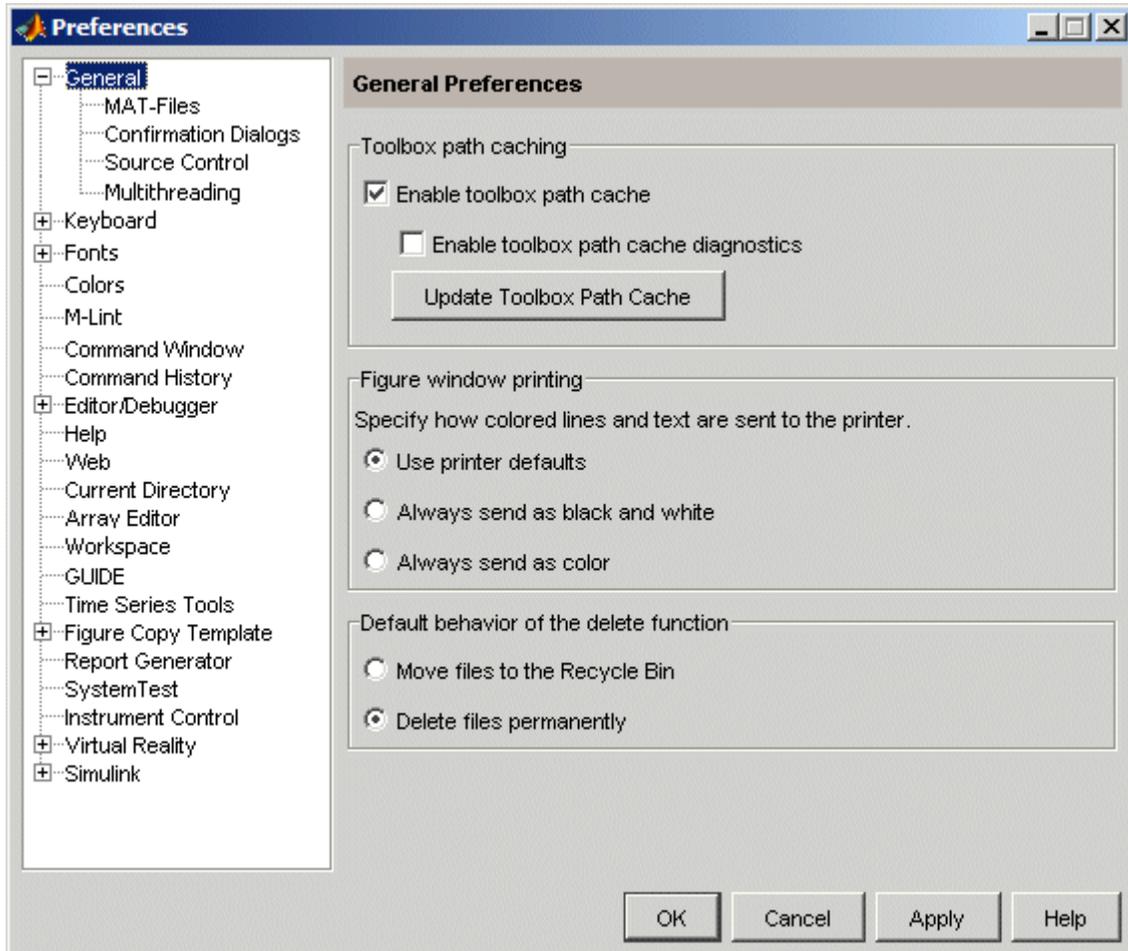
These preferences apply to all relevant tools in MATLAB.

- Toolbox path caching preference — see "Toolbox Path Caching in MATLAB" on page 1-17

- Figure window printing — see "Printing and Exporting" in MATLAB Graphics documentation

- "Default Behavior of the Delete Function" on page 2-78

- "MAT-Files Preferences" on page 2-79
- "Confirmation Dialogs Preferences" on page 2-81
- Chapter 9, "Source Control Interface"
- "Multithreading Preferences" on page 2-84

Default Behavior of the Delete Function

Files you delete using the `delete` function are permanently removed by
default. There is no opportunity to retrieve them.

You can use this preference to instead move deleted files to the Recycle Bin
on Windows, to the Trash Can on Macintosh, or to a `tmp` directory on UNIX
platforms. Then, you can recover any accidentally deleted files from these
locations. Deleted files in these locations are not automatically removed; you
must remove them using operating system features, such as **Empty Recycle
Bin** on Windows. When you select this preference, `delete` might run slower.

Function Alternative

The MATLAB delete preference actually sets the state of the `recycle` function
upon startup and when you change the preference. You can override the
behavior of the preference by setting the `recycle` function state. For example,
regardless of the preference setting, when you run

```
recycle('off')
delete('thisfile.m')
```

MATLAB permanently removes `thisfile.m` from the current directory. Files
you subsequently remove using `delete` are also permanently removed, unless
you reapply the preference to recycle or run `recycle('on')`. Regardless of the
state of the `recycle` function when you end a session, the next time you start
MATLAB, the setting for the preference is honored. For more information, see
the `recycle` and `delete` reference pages.

Note that this preference and the `recycle` function do not apply to files
you delete using the Current Directory browser. For more information, see
"Cutting or Deleting Files and Directories" on page 5-45.

MAT-Files Preferences

The **MAT-file save format** sets the default version compatibility option MATLAB uses when saving MAT-files. Use these options if you use multiple versions of MATLAB or share MAT-files with others who run a different version of MATLAB. The setting applies when you use the save function as well as when you use **Save** menu items for MAT-files, such as **File > Save Workspace As** from any desktop tool.

The MAT-file preference also applies to saving FIG-files, which include plots, as well as GUIs you create with GUIDE.
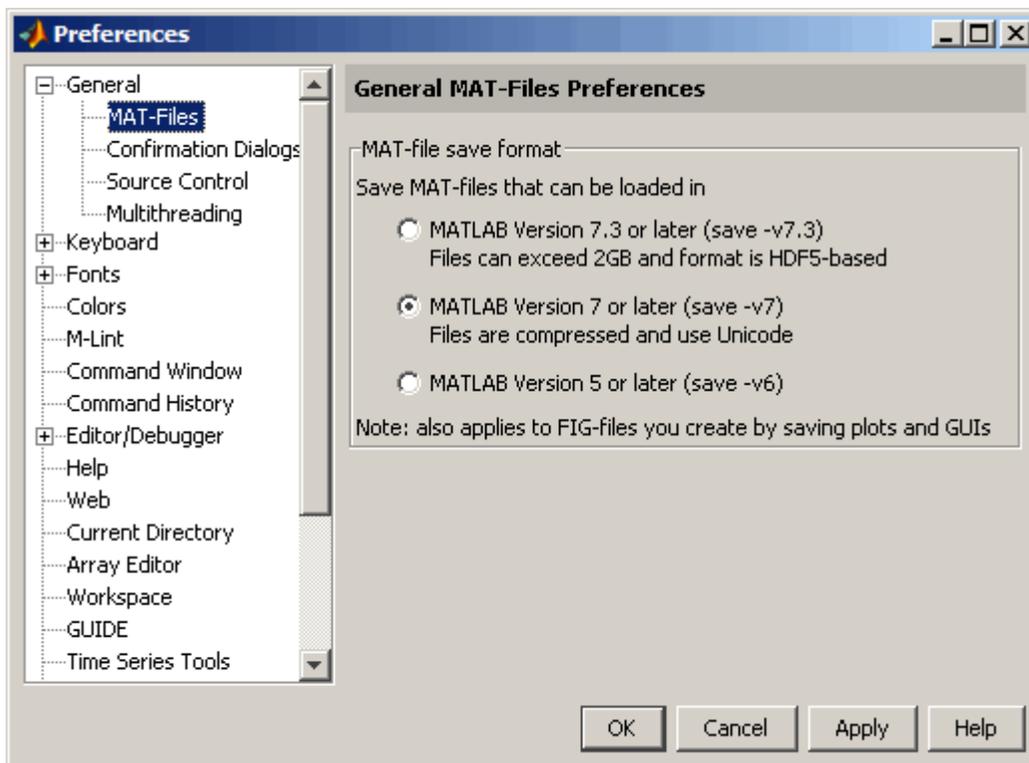
Options are

• **MATLAB Version 7.3 or later (save -v7.3)**—Starting in MATLAB Version 7.3, you can save data that is larger than 2 GB on platforms that

allow it, which is the primary purpose of this option. Using this option is equivalent to running save ‑v7.3. This format of the resulting MAT-file is HDF5-based. You cannot load these MAT-files into any versions prior to MATLAB Version 7.3; in those cases, use one of the other two options.

• **MATLAB Version 7 or later (save -v7)**—Starting in MATLAB Version 7, MATLAB compresses the data when saving a MAT-file, thereby reducing the storage space required. When you load the MAT-file, MATLAB automatically uncompresses the data. In addition, MATLAB uses Unicode character encoding for strings when you save a MAT-file, making the data accessible to other MATLAB users, regardless of the default character encoding scheme used by their systems. MAT-files saved with this option work in all MATLAB 7 versions. Using this option is equivalent to running save ‑v7.

• **MATLAB Version 5 or later (save -v6)**—Releases of MATLAB prior to Version 7 did not save compressed MAT-files. They also did not use Unicode character encoding, which sometimes prevented the exchange of MAT-files among users, particularly when they used localized systems. Specify this option to save MAT-files for use with versions prior to MATLAB Version 7. Using this option is equivalent to running save ‑v6.

Like other preferences, the **MAT-file save format** preference gets its initial value from the preference file for the previous installed version. For example, if the setting in your MATLAB 7.4 preference is ‑v6, when you upgrade from MATLAB Version 7.4 (R2007a) to MATLAB Version 7.5 (R2007b), the initial value in Version 7.5 is ‑v6.

If you upgrade from a version prior to MATLAB Version 7.3, or if you do not have a previous MATLAB version installed, the initial value is ‑v7.

Note For more information about MAT-file save formats, including restrictions, see Version Compatibility Options and Remarks in the save reference page.

Function Alternative
You can override the **MAT-file save format** preference by using the save function with a specified version compatibility option. For occasional use, this

might be more convenient than changing the preference. For example, use
`save` with the `-v6` option to ensure compatibility with MATLAB versions prior
to Version 7. For more information, see the `save` reference page.

Confirmation Dialogs Preferences

These preferences instruct MATLAB to display or not display specific
confirmation dialog boxes.
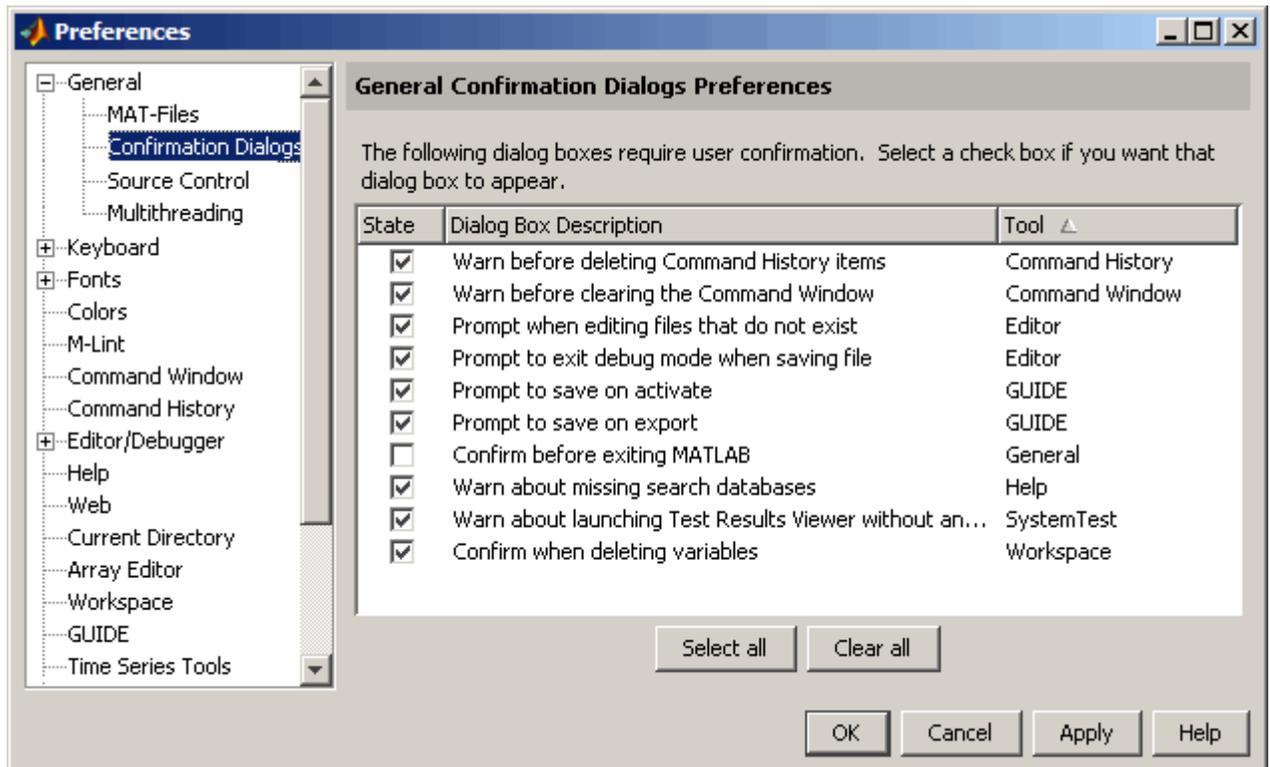
When the check box for a confirmation dialog is selected and you perform the
action it refers to, the confirmation dialog box appears. If you clear that check
box, the dialog box does not appear when you perform the action.

When the confirmation dialog box does appear, it includes a **Do not show this prompt again** check box. If you select the check box in the dialog box, it automatically clears the check box for the confirmation preference.

For example, select the check box **Warn before deleting Command History items**. Then select **Edit > Delete Selection** in the Command History, MATLAB displays the following confirmation dialog box.

If you select the **Do not show this prompt again** check box and click **OK**, the confirmation dialog box will not appear the next time you delete items from the Command History window. In addition, the **Warn before deleting Command History items** check box in the **Confirmations Dialogs** preferences pane is cleared.

The following table summarizes the confirmation dialog boxes.

Confirmation Dialogs Check Box Item	About the Confirmation Dialog Box	For More Information
Warn before deleting Command History items	Appears when you delete entries from the Command History window.	"Deleting Entries from the Command History Window" on page 3-57
Warn before clearing the Command Window	Appears when you clear the Command Window content using menu items. Does not appear when you use the clc function.	"Clearing the Command Window" on page 3-32

Confirmation Dialogs Check Box Item	About the Confirmation Dialog Box	For More Information
Prompt when editing files that do not exist	Appears when you type `edit filename`, if `filename` does not exist in the current directory or on the MATLAB path.	"Function Alternative" on page 6-9
Prompt to exit debug mode when saving file	Appears when you try to save a modified file while in debug mode.	"Ending Debugging" on page 6-119
Prompt to save on activate	Appears when you have unsaved changes to a figure and M-file, and then activate the GUI, by clicking the **Run** button, for example.	"GUIDE Preferences" in the GUIDE documentation
Prompt to save on export	Appears when you have unsaved changes to a figure and M-file, and then select **File > Export**.	"GUIDE Preferences" in the GUIDE documentation
Confirm before exiting MATLAB	Appears when you quit MATLAB.	Quitting MATLAB
Warn about missing search databases	Appears if you have help files in the Help browser for non-MathWorks products and the search database for those files has not been updated for the version of MATLAB you are running.	Contact the provider of the help files to obtain the correct version of the search database. Without the most current version, you can use the help files in the Help browser, but the Help browser search will not include those files in search results.
Confirm when deleting variables	Appears when you delete variables from the workspace using menu items. Does not appear with the `clear` function.	"Deleting Workspace Variables" on page 5-8

Multithreading Preferences

If you run MATLAB on a multiple-CPU system (multiprocessor or multicore), you can use multithreaded computation, which can improve performance for some operations. For more information, see .

Accessibility

Software Accessibility Support

MathWorks products includes a number of modifications to make them more accessible to all users. Software accessibility support for blind and visually impaired users includes

- Support for screen readers and screen magnifiers, as described in "Assistive Technologies" on page 2-87

- Command-line alternatives for most graphical user interface (GUI) options

- Keyboard access to GUI components

- A clear indication of the current cursor focus

- Information available to assistive technologies about user interface elements, including the identity, operation, and state of the element

- Nonreliance on color coding as the sole means of conveying information about working with a GUI

- Noninterference with user-selected contrast and color selections and other individual display attributes, as well as noninterference for other operating system-level accessibility features

- Consistent meaning for bitmapped images used in GUIs

- HTML documentation that is accessible to screen readers

Keyboard access to the user interface includes support for "sticky keys," which allow you to press key combinations (such as **Ctrl+C**) sequentially rather than simultaneously.

Except for scopes and real-time data acquisition, the MathWorks software does not use flashing or blinking text, objects, or other elements having a flash or blink frequency greater than 2 Hz and lower than 55 Hz.

The MathWorks believes that its products do not rely on auditory cues as the sole means of conveying information about working with a GUI. However, if you do encounter any issues in this regard, please report them to the MathWorks Technical Support group.

```
http://www.mathworks.com/contact_TS.html
```

Documentation Accessibility Support

Documentation is available in HTML format for all MathWorks products in this release.

Accessing the Documentation

To access the documentation with a screen reader, go to the documentation area on the MathWorks Web site at

```
http://www.mathworks.com/access/helpdesk/help/helpdesk.html
```

Navigating the Documentation

Note that the first page that opens lists the products. To get the documentation for a specific product, click the link for that product.

The table of contents is in a separate frame. You can use a document's table of contents to navigate through the sections of that document.

Because you will be using a general Web browser, you will not be able to use the search feature included in the MATLAB Help browser. You will have access to an index for the specific document you are using. The cross-product index of the MATLAB Help browser is not available when you are using a general Web browser.

Products

The documentation for all products is in HTML and can be read with a screen reader. However, for most products, most equations and most graphics are not accessible.

The following product documentation has been modified (as described below) to enhance its accessibility for people using a screen reader such as JAWS:

- MATLAB (many sections, but not the function reference pages (however, M-file help is accessible))

- Excel Link

- Optimization Toolbox

- Signal Processing Toolbox

- Statistics Toolbox

Documentation Modifications

Modifications to the documentation include the following:

- Describing illustrations in text (either directly or via links)

- Providing text to describe the content of tables (as necessary)

- Restructuring information in tables to be easily understood when a screen reader is used

- Providing text links in addition to any image mapped links

Equations

Equations that are integrated in paragraphs are generally explained in words. However, most complex equations that are represented as graphics are not currently explained with alternative text.

Assistive Technologies

Note To take advantage of accessibility support features, you must use MathWorks products on a Microsoft Windows platform.

Tested Assistive Technologies

The MathWorks has tested the following assistive technologies:

- JAWS 5.0, 6.0, and 7.0 for Windows (screen reader) from Freedom Scientific
- Built-in accessibility aids from Microsoft, including the Magnifier and "sticky keys"

Use of Other Assistive Technologies

Although The MathWorks has not tested other assistive technologies, such as other screen readers or ZoomText Xtra (screen magnifier) from Ai Squared, The MathWorks believes that most of the accessibility support built into its products should work with most assistive technologies that are generally similar to the ones tested.

If you use other assistive technologies than the ones tested, The MathWorks is very interested in hearing from you about your experiences.

Installation Notes for Accessibility Support

Note If you are not using a screen reader such as JAWS, you can skip this section.

This section describes the installation process for setting up your MATLAB environment to work effectively with JAWS.

Use the regular MATLAB installation script to install the products for which you are licensed. The installation script has been modified to improve its accessibility for all users.

Note Java Access Bridge 2.0 is installed automatically when you install MATLAB.

After you complete the product installation, there are some additional steps you need to perform to ensure JAWS works effectively with MathWorks products.

Setting Up JAWS

Make sure that JAWS is installed on your machine. If it is, there is probably a shortcut to it on the Windows desktop.

Setting up JAWS involves these tasks:

1 Add the Access Bridge to your Windows path (for networked installations only).

2 Create the `accessibility.properties` file.

These tasks are described in more detail below.

(For Networked Installations Only) Add Access Bridge to Your Path.

If you are running MATLAB in a networked installation environment (that is, if the MATLAB Installer was not run on your machine), you need to take the following steps to add Access Bridge to your Windows path.

Note This procedure assumes your Windows **Start** button is set to Classic mode. To set Classic mode, from the **Start** button, select **Settings**. Next select **Task Bar and Menu**. Then select the **Start Menu** tab and make sure the `Classic Start Menu` option is enabled. Click **OK** and you are done.

---

1 From the **Start** button, select **Settings**, next select **Control Panel**. Scroll down and click the **System** icon to display the System Properties dialog box**.**

2 In the **System Properties** dialog box, select the **Advanced** tab.

3 Click **Environment Variables**.

4 Under **System variables**, select the `Path` option.

5 Click the **Edit** button.

6 To the start of the `Path` environment variable, add the directory that contains `matlab.exe`; for example:

```
C:\matlab\bin\win32;
```

Be sure to include that semicolon between the end of this directory name and the text that was already there.

7 Click **OK** three times.

8 If JAWS is already running, exit and restart.

> **Note** JAWS must be started with these path changes in effect to work properly with MATLAB.

Create the accessibility.properties File.

1 Create a text file that contains the following two lines:

```
screen_magnifier_present=true
assistive_technologies=com.sun.java.accessibility.AccessBridge
```

2 Use the filename `accessibility.properties`.

3 Move the `accessibility.properties` file into

   matlabroot\sys\java\jre\win32\jre1.5.0_07\lib\

JAWS Pronunciation Dictionary. As a convenience, The MathWorks provides a pronunciation dictionary for JAWS. This dictionary is in a file called `MATLAB.jdf`.

During installation, the file is copied to your system under the MATLAB root directory at `sys\Jaws\matlab.jdf`.

To use the dictionary, you must copy it to the `\SETTINGS\ENU` folder located beneath the JAWS root installation directory.

You need to restart JAWS and MATLAB for the settings to take effect.

Testing
After you install JAWS and set up your environment as described above, you should test to ensure JAWS is working properly:

1 Start JAWS.

2 Start MATLAB.

JAWS should start talking to you as you select menu items and work with the MATLAB user interface in other ways.

Troubleshooting

This section identifies workarounds for some possible issues you may encounter related to accessibility support in MathWorks products.

JAWS Does Not Detect When the MATLAB Installation Has Started

When you select setup.exe, the Windows copying dialog box opens and you are informed. After the files have been copied, the installation splash screen opens, and then the installer starts. However, JAWS does not inform you that the installer has begun: the installer either starts up below other windows or applications or it is minimized. Since the installer is not an active item, nothing is read.

Therefore, check the Windows applications bar for the installer. After you go to the installer, you can use JAWS to perform the installation.

JAWS Stops Speaking

When many desktop components are open, JAWS with MATLAB sometimes stops speaking.

If this happens, close most of the desktop components, exit MATLAB, and restart.

Command Output Not Read

In the MATLAB Command Window, JAWS does not automatically read the results of commands.

To read command output, first select **File** > **Preferences** > **Command Window**, select the option **Use arrow keys for navigation instead of command history recall**, and click **OK**. Then, in the Command Window,

press the arrow keys to move to the command output and use JAWS keystrokes to read the output.

With this preference set, you cannot use arrow keys to recall previous commands. Instead use the following key bindings:

- Windows key bindings:
  - Previous history: **Ctrl+up arrow**
  - Next history: **Ctrl+down arrow**
- Emacs key bindings:
  - Previous history: **Ctrl+p**
  - Next history: **Ctrl+n**

To return to using the up and down arrow keys to recall previous commands, clear the preference.

Some GUI Menus Are Treated as Check Boxes

For some GUIs (for example, the figure window), menus are treated by JAWS as though they are check boxes, whether or not they actually are.

You can choose a menu item for such GUIs by using accelerator keys (e.g., **Ctrl+N** to select **New Figure**), if one is associated with a menu item. You can also use mnemonics for menu navigation (e.g., **Alt+E**).

Note that check boxes that you encounter by tabbing through the elements of a GUI are handled properly.

Text Ignored in Some GUIs

For some dialog boxes, JAWS reads the dialog box title and any buttons, but ignores any text in the dialog box.

Also, in parts of some GUIs, such as some text-entry fields, JAWS ignores the label of the field. However, JAWS will read any text in the text box.

Running Functions — Command Window and History

If you have an active Internet connection, you can watch the Working in the Development Environment video demo and the Command History video demo for an overview of the major functionality. The Command Window is where you run (execute) MATLAB statements, while the Command History is a log of the statements you have run.

Preferences for the Command
Window (p. 3-40)

Specify options for text, display, tab
size, accessibility, and indenting
for the Command Window and the
Editor/Debugger.

Command History Window (p. 3-49)

View session histories. Run
statements, copy entries, search, and
print the history. Set preferences.

Preferences for Command History
(p. 3-59)

Specify how often to automatically
save the history file and the types of
statements to exclude.

The Command Window

In this section...
"About the Command Window" on page 3-3
"Opening the Command Window" on page 3-3
"Command Window Prompt" on page 3-4
"Getting Started Message Bar in the Command Window" on page 3-5

About the Command Window

The Command Window is one of the main tools you use to enter data, run MATLAB functions and other M-files, and display results. If you have an active Internet connection, you can Working in the Development Environment video demo for an overview of the major functionality.

Opening the Command Window

When the Command Window is not open, access it by selecting **Command Window** from the **Desktop** menu. Alternatively, open the Command Window with the commandwindow function.

If you prefer a simple command line interface without the other MATLAB desktop tools, select **Desktop > Desktop Layout > Command Window Only**. For more information, see "Arranging the Desktop" on page 2-6.
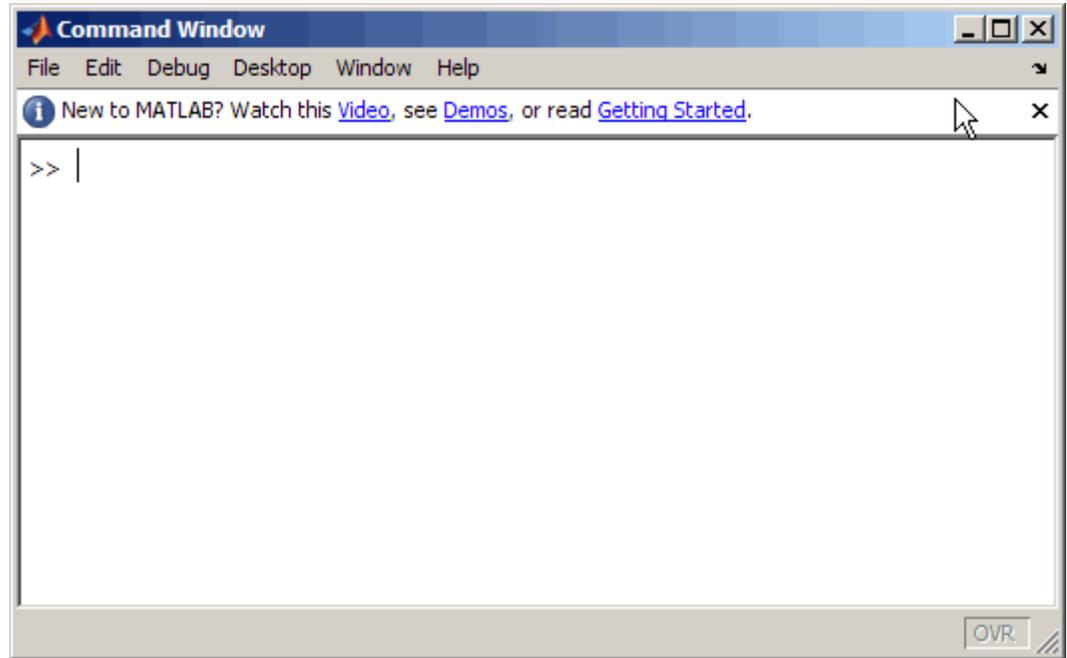
Command Window Prompt

The Command Window prompt, >>, is where you enter statements. For example, you can enter a MATLAB function with arguments, or assign values to variables. The prompt indicates that MATLAB is ready to accept input from you. When you see the prompt, you can enter a variable or run a statement. This prompt is also known as the command line.

When MATLAB displays the K>> prompt in the Command Window, MATLAB is in debug mode. Type dbquit to return to normal mode. For more information, see Chapter 6, "Editing and Debugging M-Files"

MATLAB displays the EDU>> prompt for the MATLAB Student Version.

Getting Started Message Bar in the Command Window

Just below the Command Window menu bar is a message bar that includes links to a video, demos, and information on getting started with MATLAB. If you want to remove the message bar in the Command Window, click the Close box in the right corner of the bar.
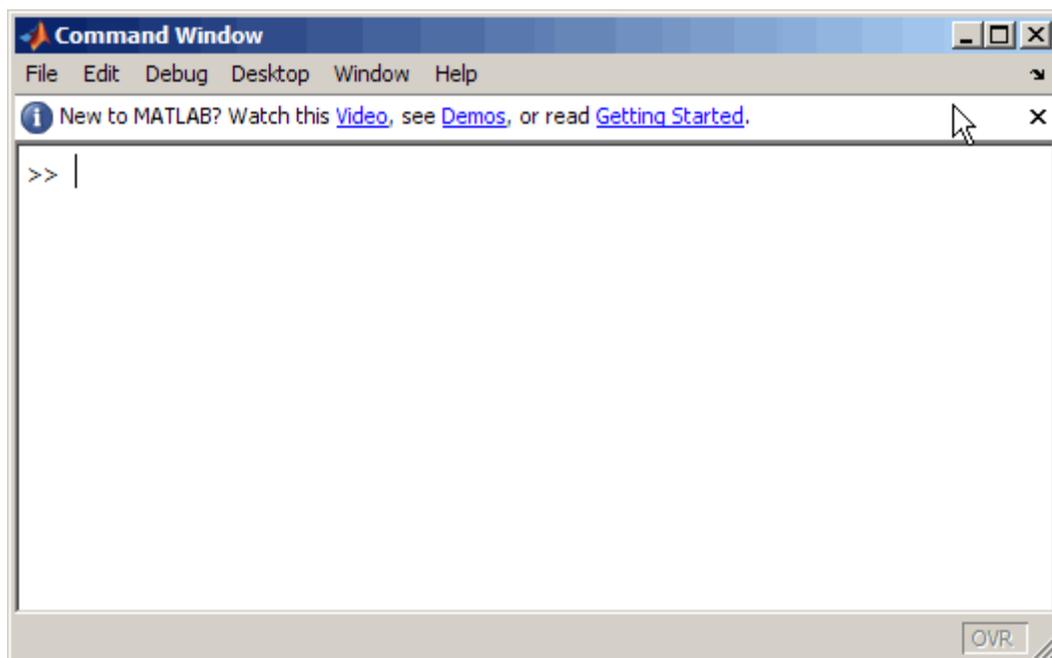
If after having closed it, you want to display the information bar again, use "Preferences for the Command Window" on page 3-40.

In addition to the message bar, there are numerous other ways to access the documentation and demos, including using the **Help** menu in most tools. Furthermore, if you place your cursor in a function name, you can right-click and then choose **Help on Selection** from the context menu to see documentation for that function. The reference page for that function opens in a popup window, or if the reference page does not exist, the M-file help appears. For more information help on selection, see "Getting Pop-Up

Help for Functions" on page 4-49—for more general information on help, see
Chapter 4, "Help for Using MATLAB".

Running Functions and Programs, and Entering Variables

In this section...
"Running Statements at the Command Line Prompt" on page 3-7
"Running External Programs" on page 3-9
"Evaluating or Opening a Selection" on page 3-12
"Displaying Hyperlinks in the Command Window" on page 3-13

Running Statements at the Command Line Prompt

Entering Variables and Running Functions

At the prompt, enter data and run functions. For example, to create A, a 3-by-3 matrix, type

```
A = [1 2 3; 4 5 6; 7 8 10]
```

When you press the **Enter** or **Return** key after typing the line, MATLAB responds with

```
A =

     1     2     3
     4     5     6
     7     8    10
```

To run a function, type the function including all arguments and press **Enter** or **Return**. MATLAB displays the result. For example, type

```
magic(2)
```

and MATLAB returns

```
ans =
  1     3
  4     2
```

Definition of a Statement. All of the information you type before pressing **Enter** or **Return** is known as a statement. This can include:

- Variable assignments: For example, `a = 3`

- Commands: M-files provided with MATLAB or toolboxes that do not accept input arguments, for example, `clc`, which clears the Command Window.

- Scripts: M-files (MATLAB program files) you write that do not take input arguments or return output arguments, for example, `myfile.m`.

- Functions and their arguments: M-files that can accept input arguments and return output arguments, for example, `magic`.

Some functions support a form that does not require an input argument, thereby operating as commands. For convenience, the term function is used to refer to both functions and commands.

When you enter program control statements, such as `if ... end`, the prompt does not appear until you complete the set of functions. In the following example, you press **Enter** at the end of each line, but the prompt does not appear until you complete the set of statements with `end`.
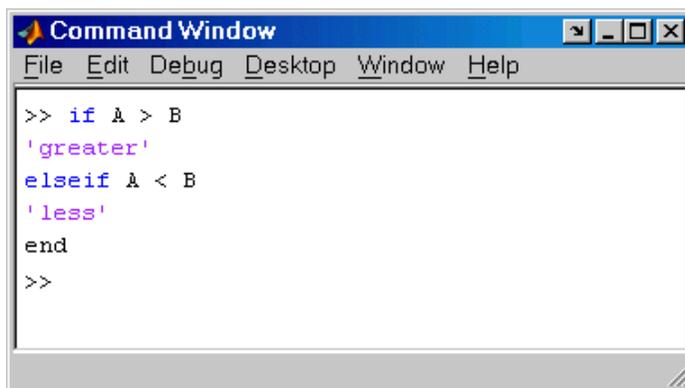
Running M-Files

Run M-files, files that contain code in the MATLAB language, the same way that you would run any other MATLAB function. Type the name of the M-file in the Command Window and press **Enter** or **Return**. The M-file must be in

the MATLAB current directory or on the MATLAB search path — for details, see "Search Path" on page 5-23. You can also use the `run` function and specify the full pathname to an M-file script.

To determine the name of the M-file currently running, use `mfilename`.

Examining Errors

If an error message appears when you run an M-file, click the underlined portion of the error message, or position the cursor within the filename and press **Ctrl+Enter**. The offending M-file opens in the Editor/Debugger, scrolled to the line containing the error.

Processing Order

In MATLAB, you can only run one process at a time. If MATLAB is busy running one function, any further statements you issue are buffered in a queue. The next statement will run when the previous one finishes.

Interrupting a Running Program

You can stop a running program by pressing **Ctrl+C** or **Ctrl+Break** at any time. On Macintosh platforms, you can also use **Command+.** (the Command key and the period key) to stop the program. For certain operations, stopping the program might generate errors in the Command Window.

For M-files that run a long time, or that call built-ins or MEX-files that run a long time, **Ctrl+C** does not always effectively stop execution. Typically, this happens on Windows rather than UNIX platforms. If you experience this problem, you can help MATLAB break execution by including a `drawnow`, `pause`, or `getframe` function in your M-file, for example, within a large loop. Note that **Ctrl+C** might be less responsive if you started MATLAB with the `-nodesktop` option (an option only for UNIX platforms).

Running External Programs

The exclamation point character, !, sometimes called bang, is a *shell escape* and indicates that the rest of the input line is a command to the operating system. Use it to invoke utilities or call other executable programs without quitting MATLAB. On UNIX, for example,

```
!vi yearlystats.m
```

invokes the vi editor for a file named `yearlystats.m`. After the external
program completes or you quit the program, the operating system returns
control to MATLAB. Add `&` to the end of the line, such as

```
!dir &
```

on Windows platforms to display the output in a separate window or to run
the application in background mode. For example

```
!excel.exe &
```

opens Excel and returns control to the Command Window so you can continue
running MATLAB statements.

The maximum length of the argument list provided as input to the bang (!)
command is determined by any restrictions maintained within the operating
system. If you are running the Windows 2003 Server, for example, the length
of the argument list input to the bang command cannot exceed 512 characters.

See the reference pages for the `unix`, `dos`, and `system` functions for details
about running external programs that return results and status.

Note To execute operating system commands with specific environment
variables, include all commands to the operating system within the `system`
call. Separate the commands using `&` (ampersand) for DOS, and `;` (semicolon)
for UNIX. This applies to the MATLAB `!` (bang), `dos`, `unix`, and `system`
functions. Another approach is to set environment variables before starting
MATLAB.

On Macintosh platforms, you cannot run AppleScript directly from MATLAB.
However, you can run the Macintosh OS X `osascript` function from the
MATLAB `unix` or `!` (bang) function to run AppleScript from MATLAB.

UNIX System Path and Running UNIX Programs from MATLAB

To run a UNIX program from MATLAB if its directory is not on the UNIX
system path MATLAB uses, take one of the actions described here.

Change Current Directory in MATLAB. Change the current directory in
MATLAB to the directory that contains the program you want to run.

Modify the UNIX System Path MATLAB Uses. Add the directories to the
system path from the shell. The exact steps depend on your shell. This is
an example using sh:

1 At the system command prompt, type

```
export PATH="$PATH:<mydirectory>"
```

where `<mydirectory>` is the directory that contains the program you want
to run.

2 Start MATLAB.

3 In MATLAB, type

```
!echo $PATH
```

The directory containing the file is added to the system path that MATLAB
uses. This change applies only to the current session of the terminal window.

Automatically Modify System Path at MATLAB Startup. If you want
to add a directory to the PATH environment variable each time you start
MATLAB, perform these steps:

1 In a text editor, open the file MATLAB/bin/matlab. This file is used to
start MATLAB.

2 Add this line to the beginning of the matlab file

```
export PATH="$PATH:<mydirectory>"
```

where `<mydirectory>` is the directory you want to add to the path.

If you run a tsch shell instead of a bash shell, use setenv instead of export.

3 Save the file.

The matlab file will modify the PATH environment variable, and then start MATLAB.

Evaluating or Opening a Selection

Make a selection in the Command Window and press **Enter** or **Return**. The selection is appended to whatever is at the prompt, and MATLAB executes it.

Similarly, you can select a statement from any MATLAB desktop tool, right-click, and select **Evaluate Selection** from the context menu. Alternatively, after making a selection, use the shortcut key, **F9**, or for some tools, press **Enter** or **Return**. For example, you can scroll up in the Command Window, select a statement you entered previously, and then press **Enter** to run it. If you try to evaluate a selection while MATLAB is busy, for example, running an M-file, execution waits until the current operation is done.

You can open a function, file, variable, or Simulink model from the Command Window. Select the name in the Command Window, and then right-click and select **Open Selection** from the context window. This runs the open function for the item you selected so that it opens in the appropriate tool:

- M-files and other text files open in the Editor/Debugger.
- Figure files (.fig) open in a figure window.
- Variables open in the Array Editor.
- Models open in Simulink.

See the open reference page for details about what action occurs if there are name conflicts. If no action exists to work with the selected item, **Open selection** calls edit.

Function Alternative

Use open or edit to open a file in the Editor/Debugger. Use type to display the M-file in the Command Window.

Displaying Hyperlinks in the Command Window

You can use MATLAB commands to create hyperlinks in the Command Window. The created hyperlink can:

- Open a Web page in a MATLAB browser using an `href` string.
- Transfer files via the file transfer protocol (FTP).
- Run a MATLAB M-file using the `matlabcolon` (`matlab:`) command.

Hyperlinks to Web Pages

When creating a hyperlink to a Web page, append a full hypertext string on a single line as input to the `disp` or `fprintf` command. For example, the command

```
disp('<a href = "http://www.mathworks.com">The MathWorks Web Site</a>')
```

displays the hyperlink

```
The MathWorks Web Site
```

in the Command Window.

When you click this link, a MATLAB Web browser opens and displays the requested page.

Transferring Files via FTP

To create a link to an FTP site, enter the site address as input to the `disp` command as shown below.

```
disp('<a href = "ftp://ftp.mathworks.com">The MathWorks FTP Site</a>')
```

This command displays

```
The MathWorks FTP Site
```

as a link in the Command Window.

When you click this link, a MATLAB browser opens and displays the requested FTP site.

Running MATLAB Functions by Hyperlink

Use `matlab:` to run a specified statement when you click a hyperlink in the Command Window. For example

```
disp('<a href="matlab:magic(4)">Generate magic square</a>')
```

displays

Generate magic square

When you click the link `Generate magic square`, MATLAB runs `magic(4)`. Alternatively, you can press **Ctrl+Enter** if the cursor is positioned in the link text. You can use the `disp`, `error`, `fprintf`, or `warning` function with this feature. Change the hyperlink color using **Colors Preferences** — see "Colors Preferences for Desktop Tools" on page 2-70. For more information, including examples, see the `matlabcolon` (`matlab:`) reference page.

Controlling Input

Case and Space Sensitivity

Uppercase and Lowercase for Variables

With respect to case, MATLAB requires an exact match for variable names. For example, if you have a variable a, you cannot refer to that variable as A.

Uppercase and Lowercase for Files and Functions

With respect to functions, filenames, objects, and classes on the search path or in the current directory, MATLAB prefers an exact match with regard to case. MATLAB runs a function if you do not enter the function name using the exact case, but displays a warning the first time you do this.

To avoid ambiguity and warning messages, always match the case exactly. It is a best practice to use lowercase only when running and naming functions. This is especially useful when you use both Windows and UNIX platforms because their file systems behave differently with regard to case.

3-15

Note that if you use the `help` function, function names are shown in all uppercase, for example, `PLOT`, solely to distinguish them. Some functions for interfacing to Java do use mixed case and the M-file help and documentation accurately reflect that.

Examples. The directory `first` is at the top of the search path and contains the file `A.m`. If you type `a` instead of `A`, MATLAB runs `A.m` but issues a warning. When you type `a` again during that session, MATLAB runs `A.m` but does not show the warning.

Add the directory `second` after `first` on the search path, with the file `a.m` in `second`. The directory `first` contains `A.m`, while `second` contains `a.m` Type `a`. MATLAB runs `a.m` but displays a warning the first time you do this.

Spaces in Expressions

Blank spaces around operators such as `-`, `:`, and `( )`, are optional, but they can improve readability. For example, MATLAB interprets the following statements the same way.

```
y = sin (3 * pi) / 2
y=sin(3*pi)/2
```

Syntax Highlighting

Some entries appear in different colors to help you better find elements, such as matching `if`/`else` statements. This is known as syntax highlighting. You can change the colors using preferences. Note that output does not appear with syntax highlighting, except for errors. For more information, see "Colors Preferences for Desktop Tools" on page 2-70.

Default colors are shown here—to change them, use **Preferences -> Colors**.

Keywords, like these for program control, are blue.

Closed strings are purple.

Unclosed strings are maroon.
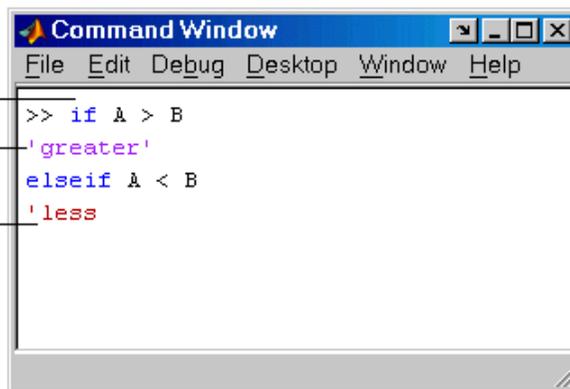
```
Command Window
File  Edit  Debug  Desktop  Window  Help

>> if A > B
'greater'
elseif A < B
'less
```

Matching Delimiters (Parentheses)

You can set a preference for MATLAB to notify you about matched and unmatched delimiters. For example, when you type a parenthesis, bracket, or brace, MATLAB highlights the matched delimiter in the pair. To set these preferences, select **File > Preferences > Keyboard > Delimiter Matching**. This feature is also available in the Editor/Debugger.

For more information, see "Delimiter Matching" on page 3-46.

Cut, Copy, Paste, and Undo Features

Use the **Cut**, **Copy**, **Paste**, **Undo**, and **Redo** features from the **Edit** menu when working in the Command Window. You can also access some of these features in the context menu for the Command Window.

Undo applies to some of the actions listed in **Edit** menu. You can undo multiple times in succession until there are no remaining actions to undo. Select **Edit > Redo** to reverse an undo.

If you use **Enter**, you cannot edit a line after entering it, even though you have not completed the flow. In that event, use **Ctrl+C** to end the flow, and then enter the statements again.

Enter Multiple Lines Without Running Them

To enter multiple lines before running any of them, use **Shift+Enter** or **Shift+Return** after typing a line. This is useful, for example, when entering a set of statements containing keywords, such as if ... end. The cursor moves down to the next line, which does not show a prompt, where you can type the next line. Continue for more lines. Then press **Enter** or **Return** to run all of the lines.

This allows you to edit any of the lines you entered before you pressing **Enter** or **Return**.

Entering Multiple Functions in a Line

To enter multiple functions on a single line, separate the functions with a comma (,) or semicolon (;). Using the semicolon instead of the comma will suppress the output for the command preceding it. For example, put three functions on one line to build a table of logarithms by typing

```
format short; x = (1:10)'; logs = [x log10(x)]
```

and then press **Enter** or **Return**. The functions run in left-to-right order.

Entering Long Statements (Line Continuation)

If a statement does not fit on one line, enter three periods (...) , also called dots, stops, or an ellipsis, at the end of the line to indicate it continues on the next line. Then press **Enter** or **Return**. Continue typing the statement on the next line. You can repeat the ellipsis to add a line break after each line until you complete the statement. When you finish the statement, press **Enter** or **Return**.

For items in single quotation marks, such as strings, you must complete the string in the line on which it was started. For example, completing a string as shown here

```
headers = ['Author Last Name, Author First Name, ' ...
'Author Middle Initial']
```

results in

```
headers =
Author Last Name, Author First Name, Author Middle Initial
```

MATLAB produces an error when you do not complete the string, as shown here:

```
headers = ['Author Last Name, Author First Name, ...
Author Middle Initial']

??? headers = ['Author Last Name, Author First Name, ...
Error: Missing variable or function.
```

Note that MATLAB ignores anything appearing after the `...` on a line, and continues processing on the next line. This effectively creates a comment out of the text following the `...` on a line. For more information, see "Commenting Out Part of a Statement" on page 6-19.

Recalling Previous Lines

Use the arrow, tab, and control keys on your keyboard to recall, edit, and reuse functions you typed earlier. For example, suppose you mistakenly enter

```
rho = (1+ sqt(5))/2
```

Because you misspelled `sqrt`, MATLAB responds with

```
Undefined function or variable 'sqt'.
```

Instead of retyping the entire line, press the up arrow ↑ key. The previously typed line is redisplayed. Use the left arrow key to move the cursor, add the missing r, and press **Enter** or **Return** to run the line. Repeated use of the up arrow key recalls earlier lines, from the current and previous sessions. Using the up arrow key, you can recall any line maintained in the Command History window.

Similarly, specify the first few characters of a line you entered previously and press the up arrow key to recall the previous line. For example, type the letters `plo` and then press the up arrow key. This displays the last line that started with `plo`, as in the most recent `plot` function. Press the up arrow key

again to display the next most recent line that began with `plo`, and so on. Then press **Enter** or **Return** to run the line. This feature is case sensitive.

If the up arrow key moves the cursor up but does not recall previous lines, clear the accessibility preference. For more information, see "Accessibility" on page 3-43.

Another way to view and access commands from the current and previous MATLAB sessions is with the Command History window — see "Command History Window" on page 3-49.

Tab Completion in the Command Window

MATLAB helps you automatically complete the names of these items as you type them in the Command Window:

- Function or model on the search path or in the current directory
- Filename or directory
- Variable, including structures, in the current workspace
- Handle Graphics property for figure in the current workspace

Type the first few characters of the item name and then press the **Tab** key. To use tab completion, you must have the tab completion preference for the Command Window selected. For details, see "Keyboard Preferences" on page 3-43.

Tab completion is also available in the Editor/Debugger, but there are some slight differences in usage. See "Tab Completion in the Editor/Debugger" on page 6-21.

These examples demonstrate how to use tab completion in the Command Window:

- "Basic Example — Unique Completion" on page 3-21
- "Multiple Possible Completions" on page 3-21
- "Tab Completion for Directories and Filenames" on page 3-24
- "Tab Completion for Structures" on page 3-24

• "Tab Completion for Properties" on page 3-25

Basic Example — Unique Completion

This example illustrates a basic use for tab completion. After creating a variable, `costs_march`, type

```
costs
```

and press **Tab**. MATLAB automatically completes the name of the variable, displaying

```
costs_march
```

Then complete the statement, adding any arguments, operators, or options, and press **Return** or **Enter** to run it. In this example, if you just press **Enter**, MATLAB displays the contents of `costs_march`. If MATLAB does not complete the name `costs_march` but instead moves the cursor to the right, you do not have the preference set for tab completion. If MATLAB displays `No Completions Found`, `costs_march` does not exist in the current workspace.

You can use tab completion anywhere in the line, not just at the beginning. For example, if you type

```
a = cost
```

and press **Tab**, MATLAB completes `costs_march`. You can also select `co` or position the cursor after `co` and press **Tab** to complete `costs_march`.

Multiple Possible Completions

If there is more than one name that starts with the characters you typed, when you press the **Tab** key, MATLAB displays a list of all names that start with those characters. For example, type

```
cos
```

and press **Tab**. MATLAB displays
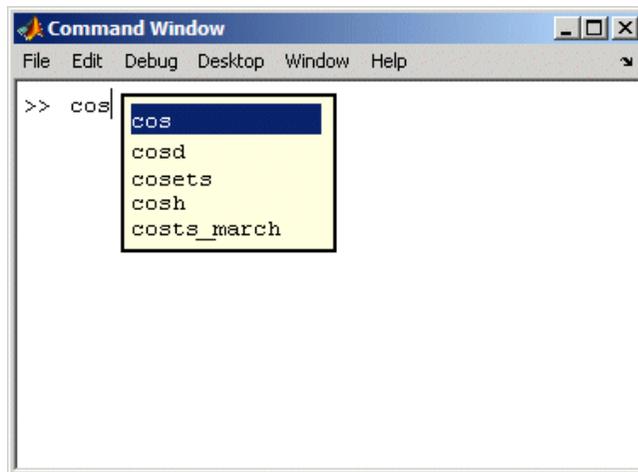
The resulting list of possible completions includes the variable name you created, `costs_march`, but also includes functions that begin with `cos`, including `cosets` from Communications Toolbox, if it is installed on the system and on the MATLAB search path. MATLAB completes variable names in the currently selected workspace, and the names of functions and models on the MATLAB search path or in the current directory.

Continue typing to make your entry unique. For example, type the next character, such as `t` in the example. MATLAB selects the first item in the list that matches what you typed, in this case, `costs_march`. Press **Enter** (or **Return**) or **Tab** to select that item, which completes the name at the prompt. In the example, MATLAB displays `costs_march` at the prompt. Add any arguments, and press **Enter** again to run the statement.

You can navigate the list of possible completions using up and down arrow keys, and **Page Up** and **Page Down** keys. You can clear the list without selecting anything by pressing **Escape**. Note that the list of possible completions might include items that are not valid commands, such as private functions.

Narrowing Completions Shown. You can narrow the list of completions
shown by typing a character and then pressing **Tab** if the Command Window
preference **Tab key narrows completions** is selected. This is particularly
useful for large lists. For example, type cam and press **Tab** to see the possible
completions. There is a scroll bar with the list because there are too many
completions to be seen at once.
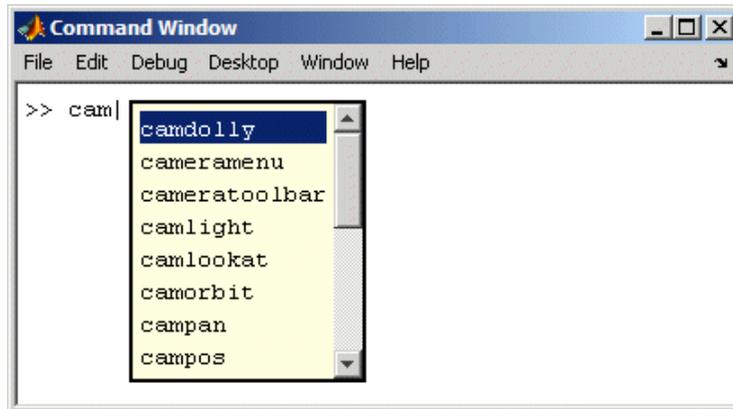
Type p and press **Tab** again. MATLAB narrows the list, showing only all
possible camp completions.
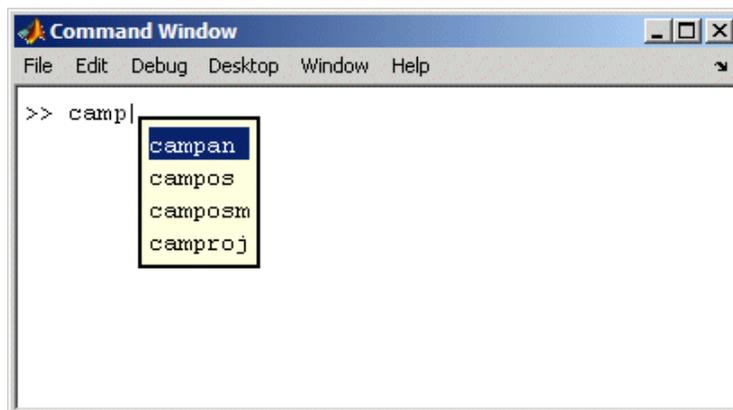
Continue narrowing the list in the same way. For the above example, type o and press **Tab** to further narrow the list. Press **Enter** or **Return** to select an item, which completes the name at the prompt.

Tab Completion for Directories and Filenames

Tab completion works for directories and filenames in MATLAB functions. For example, type

```
edit d:/
```

and press **Tab**.

MATLAB displays the list of directories and files in d, from which you can choose one. For example, type

```
mym
```

and press **Tab**.

MATLAB displays

```
edit d:/mymfiles/
```

where mymfiles is the only directory on your d drive whose name begins with mym. Continue using tab completion to display and complete directory names or filenames until you finish the edit statement.

Tab completion for directories and filenames is not supported for functions you write.

Tab Completion for Structures

For structures in the current workspace, after the period separator, press **Tab**. For example, type

```
mystruct.
```

and press **Tab** to display all fields of mystruct. If you type a structure and include the start of a unique field after the period, pressing **Tab** completes that structure and field entry.

For example, type

```
mystruct.n
```

and press **Tab**, which completes the entry `mystruct.name`, where `mystruct` contains no other fields that begin with `n`.

Tab Completion for Properties

Complete property names for figures in the current workspace using tab completion, as in this graphics example. Here, `f` is a figure. Type

```
set(f, 'pap
```

and press **Tab**. MATLAB displays
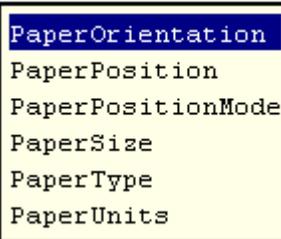
Select a property from the list. For example, type

```
u
```

and press **Enter**. MATLAB completes the property, including the closing quote.

```
set(f, 'paperunits'
```

Continue adding to the statement, as in this example

```
set(f, 'paperunits', 'c
```

and press **Tab**. MATLAB automatically completes the property

```
set(f, 'paperUnits', 'centimeters'
```

because `centimeters` is the only possible completion.

Keyboard Shortcuts in the Command Window

Following is the list of arrow and control keys that serve as shortcuts for using the Command Window. In addition to these shortcut keys (sometimes called hot keys), you can use shortcuts for menu items, which you can view on the menus, as well as general desktop shortcuts described in "Keyboard Shortcuts" on page 2-40. If you select the Emacs (MATLAB standard) preference for key bindings (see "Command Window Key Bindings" on page 3-44 for an explanation), you can also use the **Ctrl+key** combinations shown in the table.

Key or Mouse Action for Windows Preference	Control Key for MATLAB standard (Emacs) Preference	Key or Mouse Action for Macintosh Preference	Operation
↑	**Ctrl+P**	↑	Recall *p*revious line — for details, see "Recalling Previous Lines" on page 3-19. See also "Command History Window" on page 3-49, which is a log of previously used functions, and "Keeping a Session Log" on page 3-33.
			With the **Accessibility** preference selected, moves the cursor up a line when it is above the prompt. In that event, use **Ctrl+↑** to recall previous lines for Windows and Macintosh key bindings.

Key or Mouse Action for Windows Preference	Control Key for MATLAB standard (Emacs) Preference	Key or Mouse Action for Macintosh Preference	Operation
↓	**Ctrl+N**	↓	Recall *n*ext line — for details, see "Recalling Previous Lines" on page 3-19. Works only after using the up arrow or **Ctrl+P**. With the **Accessibility** preference selected, moves the cursor down a line when it is above the prompt. In that event, use **Ctrl+↓** to recall previous lines for Windows and Macintosh key bindings.
Ctrl+Home	None	**Home**	Move to top of Command Window.
Ctrl+End	None	**End**	Move to end of Command Window.
None	None	**Cmd+Home**	Move cursor and scroll to top of Command Window.
None	None	**Cmd+End**	Move cursor and scroll to end of Command Window.
None	None	**Shift+Cmd+Home**	Select to top of Command Window.
None	None	**Shift+Cmd+End**	Select to end of Command Window.
←	**Ctrl+B**	←	Move *b*ack one character.
→	**Ctrl+F**	→	Move *f*orward one character.
Ctrl+←	None	**Option+←**	Move *l*eft one word.
Ctrl+→	None	**Option+→**	Move *r*ight one word.
Home	**Ctrl+A**	**Cmd+←**	Move to beginning of current statement. With Macintosh key bindings, move to beginning of current line.
End	**Ctrl+E**	**Cmd+→**	Move to *e*nd of current statement. With Macintosh key bindings, move to end of current line.

Key or Mouse Action for Windows Preference	Control Key for MATLAB standard (Emacs) Preference	Key or Mouse Action for Macintosh Preference	Operation
Esc	**Ctrl+U**	**Esc**	Clear the command line when cursor is at the command line. Otherwise, move cursor to command line.
Delete	**Ctrl+D**	Forward **Delete**	Delete character after cursor.
Backspace	**Ctrl+H**	**Delete**	Delete character before cursor.
None	**Ctrl+K**	None	Cut contents (*k*ill) from cursor to end of current line.
Insert	None	None	Change to overwrite mode from insert mode, or change to insert mode from overwrite mode. View current mode in the status bar: OVR is gray for insert mode. In overwrite mode, what you type replaces existing text and the cursor is a wide block. (Not supported on Macintosh platforms.)
Double-click	None	Double-click	Select current word. To select additional words, hold mouse after second click and continue dragging left or right.
None	None	**Shift+Option+**←	Select to previous word.
None	None	**Shift+Option+**→	Select to next word.
Triple-click	None	None	Select current line. To select additional lines, hold mouse after second click and continue dragging up or down.
Shift+Home	None	**Shift+Cmd+**←	Select from cursor to beginning of statement. With Macintosh key bindings, select to beginning of line.

Key or Mouse Action for Windows Preference	Control Key for MATLAB standard (Emacs) Preference	Key or Mouse Action for Macintosh Preference	Operation
Shift+End	None	**Shift+Cmd+**→	Select from cursor to end of statement. With Macintosh key bindings, select to end of line.
Enter in selection	None	None	Append selection to statement at command line and execute it.
Ctrl+Enter in hyperlink	None	**Ctrl+Enter** in hyperlink	Open hyperlink displayed in Command Window. For example, in the hyperlink of an error message, opens the file in the Editor/Debugger at that line number.

Navigating Above the Command Line

To look at or copy information in the Command Window that is above the command line (>> prompt), use the mouse and scroll bar, key combinations such as **Ctrl+Home**, and search features. By default, the up and down arrow keys recall statements so you cannot use them to move the cursor when it is above the command line.

To use the up and down arrow keys to move the cursor when it is above the command line, select **File > Preferences > Command Window**, and select the **Accessibility** preference.

Controlling Output

In this section...
"Echoing Execution" on page 3-30
"Suppressing Output" on page 3-30
"Paging of Output in the Command Window" on page 3-30
"Formatting and Spacing Numeric Output" on page 3-31
"Clearing the Command Window" on page 3-32
"Printing Command Window Contents" on page 3-33
"Keeping a Session Log" on page 3-33

Echoing Execution

To display each function within a statement as it executes, run echo on. For details, see the echo reference page.

Suppressing Output

If you end a statement with a semicolon (;) and then press **Enter** or **Return**, MATLAB runs the statement but does not display any output. This is particularly useful when you generate large matrices. For example, running

```
A = magic(100);
```

creates A but does not show the resulting matrix in the Command Window.

Paging of Output in the Command Window

If output in the Command Window is lengthy, it might not fit within the screen and display too quickly for you to see it without scrolling back to it. To avoid that problem, use the more function to control the paging of output in the Command Window. By default, more is off.

After you type more on, MATLAB displays only a page (a screen full) of output, pauses, and displays

```
--more--
```

indicating there is more output to display. Press one of the following keys.

Key	Action
Enter or **Return**	To advance to the next line
Space Bar	To advance to the next page
q	To stop displaying the output

You can scroll in the Command Window to see input and output that are no longer in view. As an alternative to scrolling, you can use the up and down arrow keys if the Command Window Accessibility preference is selected.

Formatting and Spacing Numeric Output

By default, numeric output in the Command Window is displayed as 5-digit scaled, fixed-point values, called the short format. To change the numeric format of output for the current and future sessions, set the Command Window preference for text display. The text display format affects only how numbers are shown, not how MATLAB computes or saves them.

Function Alternative

Use the format function to control the output format of the numeric values displayed in the Command Window. The format you specify applies until you change it or until the end of the session. More advanced alternatives are listed in the "See Also" section of the format reference page.

Examples of Formats

Here are a few examples of the various formats and the output produced from the following two-element vector x.

```
x = [4/3 1.2345e-6]

format short
   1.3333    0.0000

format short e
```

```
     1.3333e+000   1.2345e-006

format +
++
```

A complete list and description of available formats is in the reference page for format. For more control over the output format, use the sprintf and fprintf functions.

Controlling Spacing

To control spacing in the output, use the Command Window preference for text display or the format function. Use

```
format compact
```

to suppress blank lines, allowing you to view more information in the Command Window. To include the blank lines, which can help make output more readable, use

```
format loose
```

Clearing the Command Window

Select **Clear Command Window** from the **Edit** menu or context menu to clear it. This does not clear the workspace, but only clears the view. Afterwards, you still can use the up arrow key to recall previous functions. A confirmation dialog box appears if you select the preference for it; see preferences for "Confirmation Dialogs Preferences" on page 2-81 for more information.

Function Alternative

Use clc to clear the Command Window. Similar to clc is the home function, which moves the prompt to provide a clear screen, but does not clear the text so you can still scroll up to see it.

Printing Command Window Contents

To print the complete contents of the Command Window, select **File > Print**. To print only a selection, first make the selection in the Command Window and then select **File > Print Selection**.

Specify printing options for the Command Window by selecting **File > Page Setup**. For example, you can print with a header. For more information, see "Printing and Page Setup Options for Desktop Tools" on page 2-52.

Keeping a Session Log

The diary Function

The diary function creates a copy of your MATLAB session in a disk file, including keyboard input and system responses, but excluding graphics. You can view and edit the resulting text file using any text editor, such as the Editor/Debugger. To create a file on your disk called sept23.out that contains all the functions you enter, as well as MATLAB output, enter

```
diary('sept23.out')
```

To stop recording the session, use

```
diary('off')
```

To view the file, run

```
edit('sept23.out')
```

Other Session Logs

There are two other means of viewing session information:

- The Command History window, which contains a log of all functions executed in the current and previous sessions.

- The logfile startup option — see "Startup Options" on page 1-12.

Searching in the Command Window

In this section...
"Introduction" on page 3-34
"Find Dialog Box" on page 3-34
"Incremental Search" on page 3-35

Introduction

You can search for specified text that appears in the Command Window, where the text was either part of input you supplied, or output displayed by MATLAB. After finding the desired text, you can copy and paste it to the prompt in the Command Window to run it, or into an M-file or other file.

See also "Recalling Previous Lines" on page 3-19, "Tab Completion in the Command Window" on page 3-20, and "Keyboard Shortcuts in the Command Window" on page 3-26 for techniques to reuse previous statements and navigate in the Command Window. To find files and text in files, see "Finding Files and Content Within Files" on page 5-49.

Find Dialog Box

Select **Find** from the **Edit** menu to search for specified text in the Command Window using the Find dialog box. Complete the dialog box. The search begins at the current cursor position. MATLAB finds the text you specified and highlights it. Click **Find Next** or **Find Previous** to find another occurrence, or use the keyboard shortcuts **F3** and **Shift+F3**.
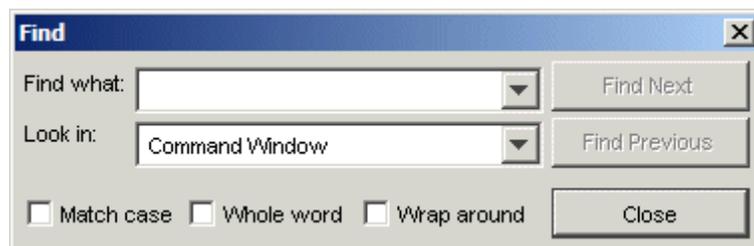
MATLAB beeps when a search for **Find Next** reaches the end of the Command Window, or when a search for **Find Previous** reaches the top of the Command Window. If you have **Wrap around** selected, it continues searching after beeping.

Note that you can only search for text currently displayed in the Command Window. To increase the amount of information maintained in the Command Window, increase the setting for the command session scroll buffer size in **Command Window Preferences**, and do not clear the Command Window.

Change the selection in the **Look in** field to search for the specified text in other MATLAB desktop tools.

Incremental Search

With the incremental search feature, the cursor moves to the next or previous occurrence of the specified text in the Command Window. It is similar to the Emacs search feature. To use the incremental search feature in the Command Window,

1 Position the cursor where you want the search to begin.

2 How you begin the incremental search depends on your setting for the Command Window key bindings preference:

- Press **Ctrl+S** for **Emacs**, or
- Press **Ctrl+Shift+S** for **Windows**

3 To look for the previous occurrence, press **Ctrl+R** or **Ctrl+Shift+R** instead.

An incremental search field, **Inc Search**, appears at the bottom of the Command Window and is preceded by **F** for a forward search, or **R** when you are looking for the previous occurrence (reverse search).

Search begins at current cursor position.
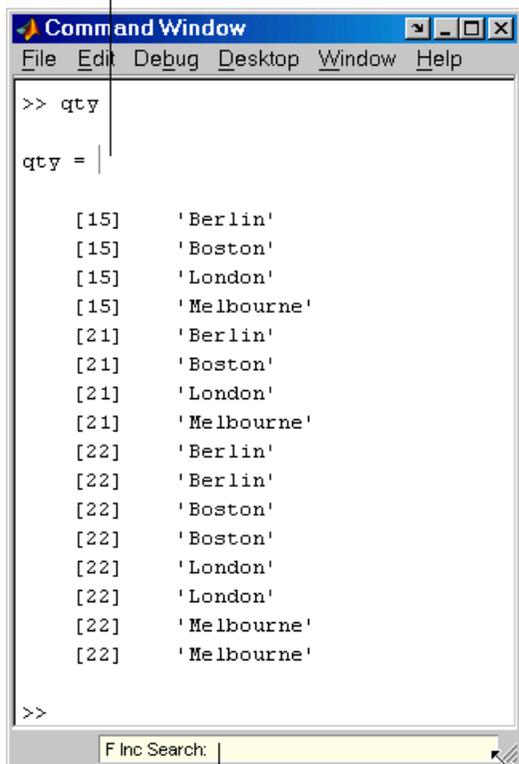
Incremental search field.

4 In the **Inc Search** field, type the text you want to find. For example, look for Boston.

As you type the first letter, b, the first occurrence of that letter in the Command Window after the current cursor position is highlighted. For the example shown, the first occurrence of b is highlighted, the b in Berlin. Note that incremental search allows for case sensitivity — see "Case Sensitivity in Incremental Search" on page 3-38.

MATLAB finds the next b.

```
>> qty

qty =

    [15]      'Berlin'
    [15]      'Boston'
    [15]      'London'
    [15]      'Melbourne'
    [21]      'Berlin'
    [21]      'Boston'
    [21]      'London'
    [21]      'Melbourne'
    [22]      'Berlin'
    [22]      'Berlin'
    [22]      'Boston'
    [22]      'Boston'

F Inc Search: b
```

When you type the next letter, the first occurrence of the text becomes highlighted. In the example, when you add the letter o to the b so that the **Inc Search** field now has bo, the bo in Boston becomes highlighted.

- If you mistype in the **Inc Search** field, use the **Back Space** key to remove the last letters and make corrections.

- After finding the bo, you can press **Ctrl+W** to complete that word. In this example, Boston appears in the **Inc Search** field.

5 To find the next occurrence of Boston in the Command Window, press **Ctrl+S**. To find the previous occurrence of the text, press **Ctrl+R**

6 If MATLAB beeps, it means either that the text was not found, or the search wrapped past the end (or beginning) of the Command Window and continued at the beginning (or end).

- When the text is not found, **Failing** appears in the incremental search field. Modify the search term in the incremental search field and try again. Use **Ctrl+G** to automatically remove characters back to the last successful search. For example, if `plode` fails, **Ctrl+G** removes the `de` from the search term because `plo` does exist in the Command Window.

7 To end the incremental search, press **Esc** or **Enter**, or any other key that is not a character or number.

The **Inc Search** field no longer appears. The cursor is at the position where the text was last found, with the search text highlighted.

Incremental search is also available in the Editor/Debugger — see "Incremental Search" on page 6-51.

Case Sensitivity in Incremental Search

When you enter lowercase letters in the **Inc Search** field, for example, `b`, incremental search looks for both lowercase and uppercase instances of the letters, for example `b` and `B`. However, if you enter uppercase letters, for example, `B`, incremental search only looks for instances that match the case you entered.

In the example, enter b0 in the **Inc Search** field and incremental search does not find any matching text.

Incremental
search is
case-sensitive
when you enter
uppercase
letters. Here, no
matches are
found for b0.

```
Command Window                          ▣ _ □ ✕
File  Edit  Debug  Desktop  Window  Help

>> qty

qty =

        [15]      'Berlin'
        [15]      'Boston'
        [15]      'London'
        [15]      'Melbourne'
        [21]      'Berlin'
        [21]      'Boston'
        [21]      'London'
        [21]      'Melbourne'
        [22]      'Berlin'
        [22]      'Berlin'
        [22]      'Boston'

Failing  F Inc Search: bO
```
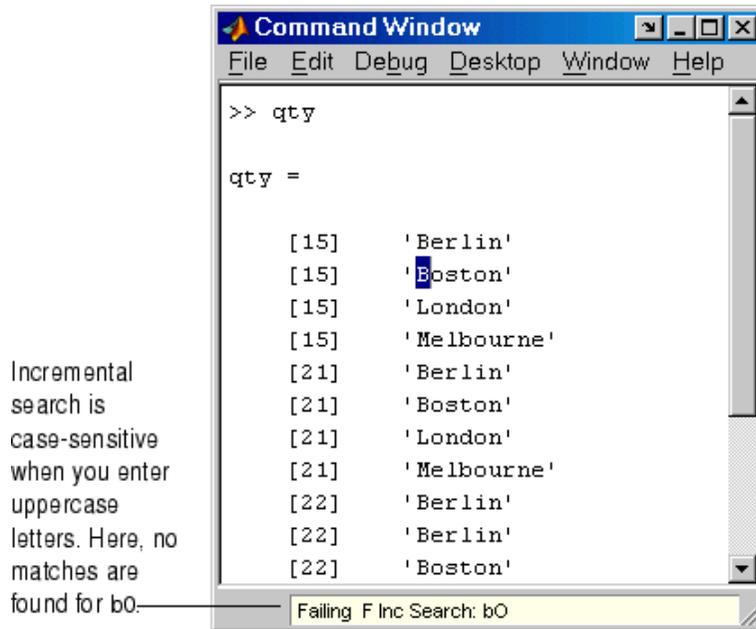
Preferences for the Command Window

In this section...
"Text, Display, Accessibility, and Tab Size Preferences" on page 3-40
"Keyboard Preferences" on page 3-43

See also:

- "Fonts Preferences for Desktop Tools" on page 2-62
- "Confirmation Dialogs Preferences" on page 2-81

Text, Display, Accessibility, and Tab Size Preferences

To set these preferences for the Command Window, select **File > Preferences** and then select **Command Window** in the left pane of the Preferences dialog box.
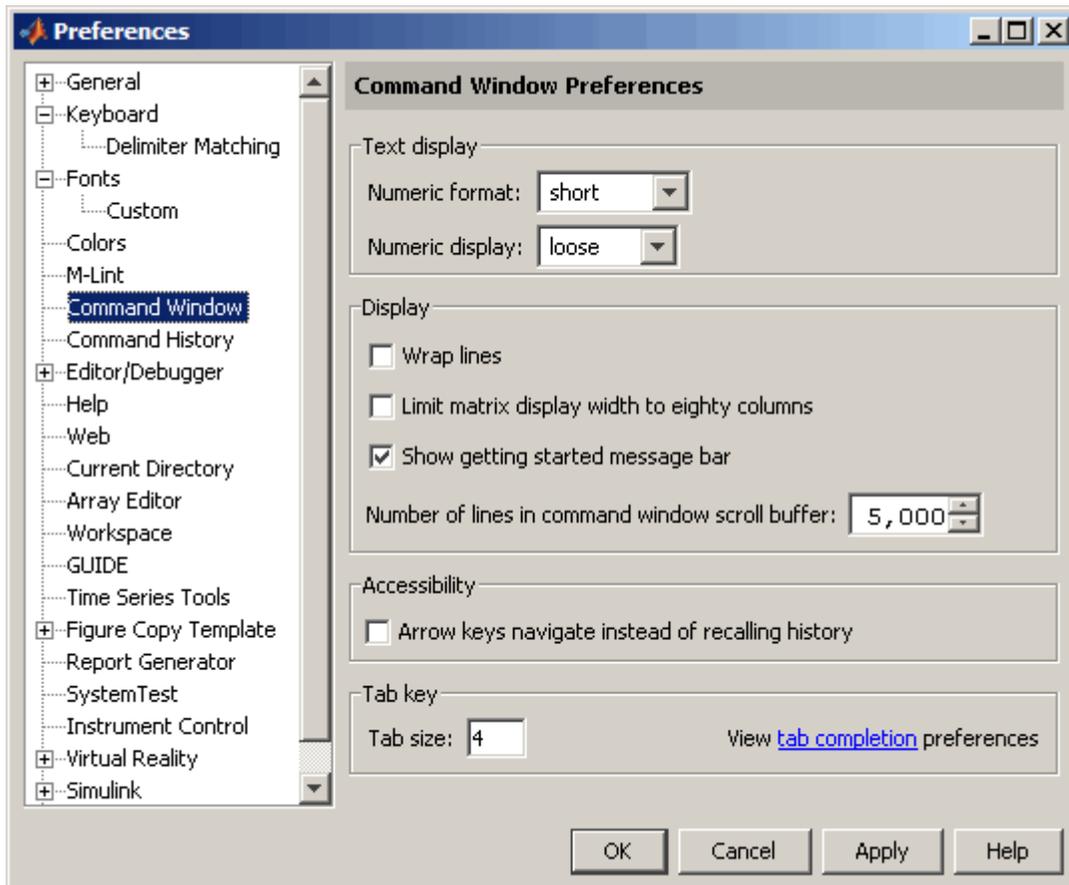
Text Display

Specify the format, that is, how output appears in the Command Window.

Numeric format. Specify the output format of numeric values displayed in the Command Window. This affects only how numbers are displayed, not how MATLAB computes or saves them. The format reference page includes the list of available formats, with examples.

Numeric display. Specify spacing of output in the Command Window. To suppress blank lines, use `compact`. To display blank lines, use `loose`. For more information, see the reference page for `format`.

Display

Wrap lines. Select to make a single line of input or output in the Command Window break into multiple lines in order to fit within the current width of the Command Window. This is useful for console mode. With this option selected, an entire line is visible without scrolling, and the horizontal scroll bar does not appear because it is not needed.

Show getting started message bar. Select to display the getting started message bar in the Command Window. It appears beneath the menu bar and contains links to a video, documentation, and demos. For more information, see "Getting Started Message Bar in the Command Window" on page 3-5

Limit matrix display width to eighty columns. When selected, MATLAB displays only 80 columns of matrix output, regardless of the width of the Command Window. Clear the check box if you make the Command Window wider than 80 columns and want matrix output to fill the width of the Command Window. See also the `display` reference page.

To determine the number of columns and rows that will display in the Command Window, given its current size, use

```
get(O,'CommandWindowSize')
```

With the matrix display width preference cleared, the number of columns is based on the width of the Command Window. With the preference set to 80 columns, the number of columns is always 80.

Number of lines in command window scroll buffer. Set the number of lines maintained in the Command Window, from `1,000` to `25,000`. This is the number of lines you can see when you scroll vertically. A larger buffer means you can view more lines and it provides a larger base for search features, but requires more memory.

This preference setting does not impact the number of lines you can recall when you use the up arrow key in the Command Window. Using the up arrow key, you can recall all lines shown in the Command History window, regardless of how many lines you can see in the Command Window.

Accessibility

Select this option to use the up and down arrow keys to move the cursor when it is above the command line. With this preference selected, use the **Ctrl+** up arrow or down arrow key to recall statements for Windows and Macintosh key bindings, or **Ctrl+P** and **Ctrl+N** for **MATLAB standard (Emacs)** key bindings.

Clear this preference to use the up and down arrow keys to recall statements. Use the mouse and other features to move the cursor when above the command line.

Tab key

Tab size. Number of spaces assigned to a tab stop when displaying output. The default is four spaces, except on UNIX platforms where the default is eight spaces. This does not apply when the tab completion preference is selected.

Keyboard Preferences

To set key binding, tab completion, and delimiter matching preferences for the Command Window and the Editor/Debugger, select **File > Preferences** and then select **Keyboard** in the left pane of the Preferences dialog box.
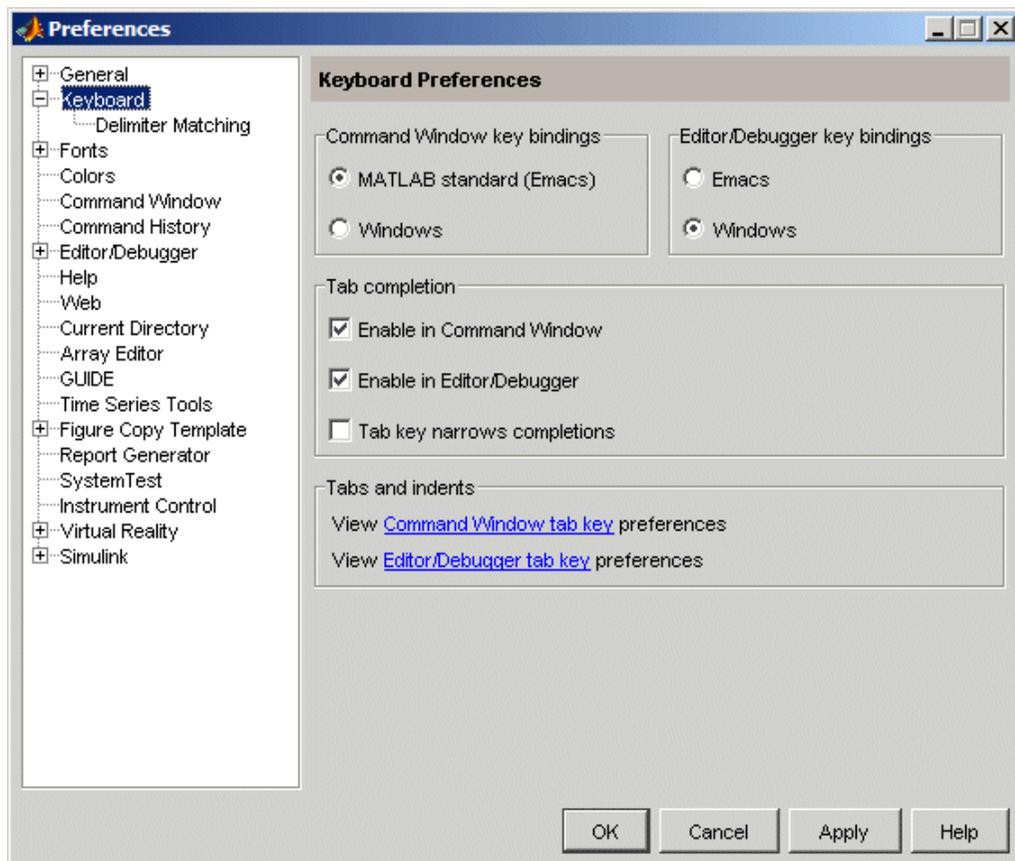
- "Command Window Key Bindings" on page 3-44
- "Editor/Debugger Key Bindings" on page 3-45
- "Tab Completion" on page 3-45
- "Tabs and Indents" on page 3-46
- "Delimiter Matching" on page 3-46

Command Window Key Bindings

Specify the keyboard shortcuts (key bindings) to be used at the command line.

MATLAB standard (Emacs). Allows you to use the control keys listed in "Keyboard Shortcuts in the Command Window" on page 3-26, which should be familiar to existing MATLAB users and Emacs users. For example, **Ctrl+A** moves the cursor to the beginning of the line.

Windows. Allows you to use standard Windows control keys. For example, **Ctrl+A** is the shortcut for **Edit > Select All**, which selects the entire contents of the Command Window.

Macintosh. This option is available only on Macintosh platforms. It allows you to use Macintosh keys, such as the **Command** key instead of the **Ctrl** key.

Editor/Debugger Key Bindings

Specify the keyboard shortcuts (key bindings) to be used by the Editor/Debugger. The Editor/Debugger key bindings are also used by other tools, for example, the **Callback** field in the Shortcut Editor dialog box.

Select **Windows**, **Emacs**, or **Macintosh** (available only on Macintosh platforms), depending on which convention you want the Editor/Debugger to follow for accelerators and shortcuts. The accelerators on the menus change after you change this option.

For example, when you select Windows key bindings, the shortcut to paste a selection is **Ctrl+V**. When you select Emacs key bindings, the shortcut to paste a selection is **Ctrl+Y**. When you select Macintosh key bindings, the shortcut to paste a selection is **Command+V**. You can see the accelerator on the **Edit** menu for the **Paste** item.

Tab Completion

Enable in Command Window. Select the check box to use tab completion when typing functions in the Command Window. Clear the check box if you do not want to use the tab completion feature. In that event, when you press the **Tab** key, MATLAB moves the cursor to the next tab stop rather than completing a function — see also the preference for "Tab size" on page 3-43.

Enable in Editor/Debugger. Select the check box to use tab completion when typing functions in the Editor/Debugger. Clear the check box if you do not want to use the tab completion feature. In that event, when you press the **Tab** key, MATLAB moves the cursor to the next tab stop rather than completing a function — see also in the online documentation.

Tab key narrows completions. Select this check box to narrow the list of possible completions shown by typing another character and pressing **Tab**. For details, see "Narrowing Completions Shown" on page 3-23.

Tabs and Indents

The links go to the panes where you can view and set preferences for

- **Tab** key size in the Command Window, which is used when the tab completion preference is not set

- **Tab** key size and indenting preferences in the Editor/Debugger

Delimiter Matching

To set these preferences, select **File > Preferences > Keyboard > Delimiter Matching** . These preferences apply to the Command Window and the Editor/Debugger.

With these preferences selected, MATLAB alerts you to matched and unmatched delimiters based on the MATLAB language syntax rules. For example, when you type a parenthesis or another delimiter, MATLAB highlights the matched parenthesis or delimiter in the pair.

Delimiter pairs are parentheses ( ), brackets [ ], and braces { }. For the Editor/Debugger , paired language keywords are also matched. Paired language keywords include for, if, while, else, and end statements.

In the following illustration, MATLAB underlines the left parenthesis in the pair when you move over the right parenthesis using an arrow key.

```
len(nl) = sum(sqrt(dot(temp',temp')));
```

If the matching delimiter is not visible on the screen, a pop-up window appears and shows the line containing the matching delimiter. In the Editor/Debugger, the line number is included. Click in the pop-up window to go to that line.
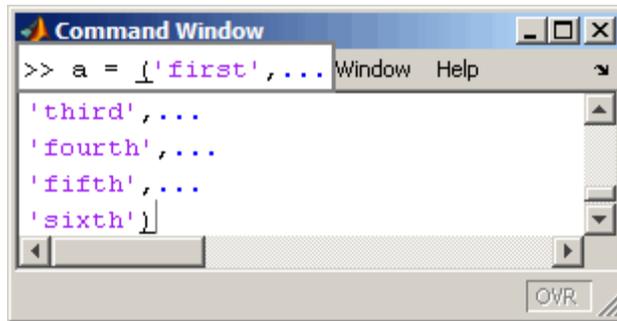
Match while typing. Select the check box if you want to be alerted to matches and mismatches in pairs of delimiters as you type them. Then choose how you want MATLAB to alert you to matches by selecting an entry from **Show match with**. When you type a closing (or opening) delimiter in the Command Window or Editor/Debugger, MATLAB alerts you based on the option you choose:

- Balance — The corresponding delimiter is highlighted briefly.
- Underline — Both delimiters in the pair are underlined briefly.
- Highlight — Both delimiters in the pair are highlighted briefly.

Choose how you want MATLAB to alert you to mismatches using **Show mismatch with**. When you type a closing delimiter that does not have an opening match, MATLAB alerts you based on the option you choose:

- Beep — MATLAB beeps.
- Strikethrough — The delimiter you typed is briefly crossed out.
- None — There is no action.

3-47

Match on arrow key. Select the check box if you want to be alerted to matches and mismatches in pairs of delimiters when you use an arrow key to move the cursor over a delimiter. Then choose how you want MATLAB to alert you to matches by selecting an entry from **Show match with**. When you move the arrow over a closing (or opening) delimiter in the Command Window or Editor/Debugger, MATLAB alerts you based on the option you choose:

- Underline — Both delimiters in the pair are underlined briefly.

- Highlight — Both delimiters in the pair are highlighted briefly.

Choose how you want MATLAB to alert you to mismatches by selecting an entry from **Show mismatch with**. When you move an arrow key over a delimiter that does not have a match, MATLAB alerts you based on the option you choose:

- Beep — MATLAB beeps.

- Strikethrough — The delimiter is briefly crossed out.

- None — There is no alert.

Command History Window

In this section...
"Overview" on page 3-49
"Viewing Statements in the Command History Window" on page 3-50
"Using Statements from the Command History Window" on page 3-51
"Searching in the Command History Window" on page 3-52
"Printing the Command History Window" on page 3-57
"Deleting Entries from the Command History Window" on page 3-57

Overview

The Command History window displays a log of the statements most recently run in the Command Window. If you have an active Internet connection, you can watch the Command History video demo for an overview of the major functionality.

To show or hide the Command History window, use the **Desktop** menu. Alternatively, use commandhistory to open the MATLAB Command History window when it is closed, or to select it when it is open. For details, see "Arranging the Desktop" on page 2-6.

Timestamp marks the start of each session. Select it to select all statements in the history for that session.

Click - to hide history for that session. Click + to expand.

Select one or more lines and right-click to copy, evaluate, create a shortcut, or create an M-file from the selection.
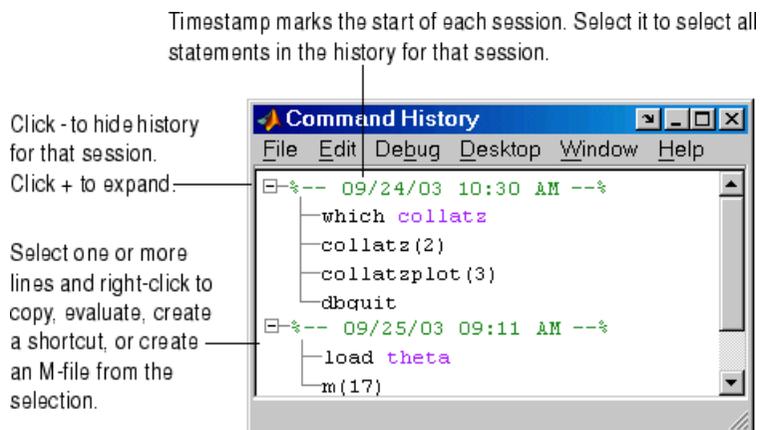
MATLAB provides other options for viewing a history of statements. See also the following sections:

- "Recalling Previous Lines" on page 3-19, which describes using the up arrow in the Command Window

- The `diary` function reference page

- "Startup Options" on page 1-12, which includes the `logfile` startup option

Viewing Statements in the Command History Window

The Command History window lists statements you ran in the current session and in previous sessions. The time and date for each session appear at the top of the history of statements for that session. Use the scroll bar or the up and down arrow keys to move through the Command History window.

Click **-** to hide the history for a session, and click **+** to show it. Select a timestamp to select all entries for that session. With a timestamp selected, you can press the **+** or **-** key on the numeric keypad to show and hide entries.

The Command History window and the Command Window's statement recall feature save to the file `history.m`, which is stored in the same directory as MATLAB preferences. Type `prefdir` in the Command Window to see the location of the file. The `history.m` file is loaded when MATLAB starts, and it stores a maximum of 20,000 bytes, deleting the oldest entries as needed to maintain that size.

MATLAB automatically saves the `history.m` file throughout the session according to the **Saving** preference you specified. You can choose to automatically exclude certain statements from being written to the `history.m` file with the **Settings** preference. For details, see "Preferences for Command History" on page 3-59.

Using Statements from the Command History Window

You can select entries in the Command History window and then perform the following actions for the selected entries.

Action	How to Perform the Action
Run statements in the Command Window	Double-click an entry (entries) in the Command History window to execute the statement(s) in the entries. For example, double-click `edit myfile` to open `myfile.m` in the Editor/Debugger. You can also run the statements in an entry by right-clicking the entry and selecting **Evaluate Selection** from the context menu, or by selecting an entry and pressing **Enter** or **Return**.
Edit and run statements in the Command Window	Select an entry or entries and then select **Copy** from the context menu. Paste the selection into the Command Window. Alternatively, drag the selection to the Command Window. Then in the Command Window, edit the statements, and press **Enter** or **Return** to execute them.
Copy statements to another window	Select an entry or entries and then select **Copy** from the context menu. Paste the selection into an open M-file in the Editor/Debugger or any application. Alternatively, drag the selection from the Command History window to an open M-file or another application.
Create an M-file from statement(s)	Select an entry or entries and then right-click and select **Create M-File** from the context menu. The Editor/Debugger opens a new M-file that contains the statements you selected from the Command History window.
Create a shortcut from statement(s)	Select an entry or entries and then right-click and select **Create Shortcut** from the context menu. Alternatively, drag the selection to the **Shortcuts** toolbar. The Shortcut Editor opens and the selected statements appear in the **Callback** field. For more information, see "Shortcuts for MATLAB — Easily Run a Group of Statements" on page 2-32.

Searching in the Command History Window

There are two types of search in the Command History window:

- "Finding Next Entry By Letter" on page 3-52
- "Finding Text" on page 3-56

After finding an entry, you can copy and paste it into an M-file or any file, or you can right-click and select **Evaluate Selection** to run the entry.

Finding Next Entry By Letter

Type a letter in the Command History window. The Command History window searches backwards to find the last previous entry that begins with that letter as illustrated in this example:

1 Position the cursor at anywhere in the Command History window.
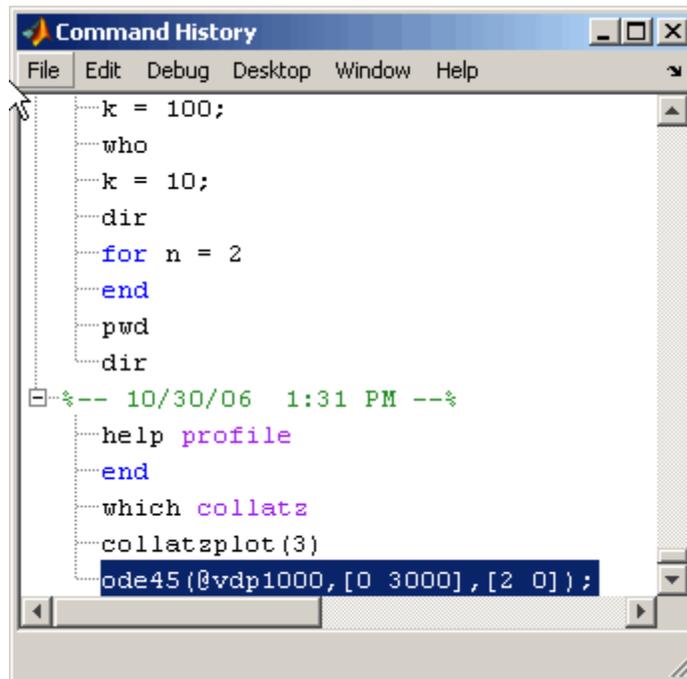
2 Type the first letters of the entry you want to find. For example, type `my`.

The Command History window searches backwards and selects the previous entry that begins with the letters you typed; in this example, you typed `my`, and the Command History finds `myfor`.

As you begin typing that a small yellow-background pop-up window, `Search history for:`, appears at the top of the Command History window. This window keeps track of your search target as you type additional letters to narrow the focus of your search.

If the search finds a matching entry in a sessions that is collapsed, it expands the session and selects the entry.

Incremental search target. Changes
as you type additional letters.
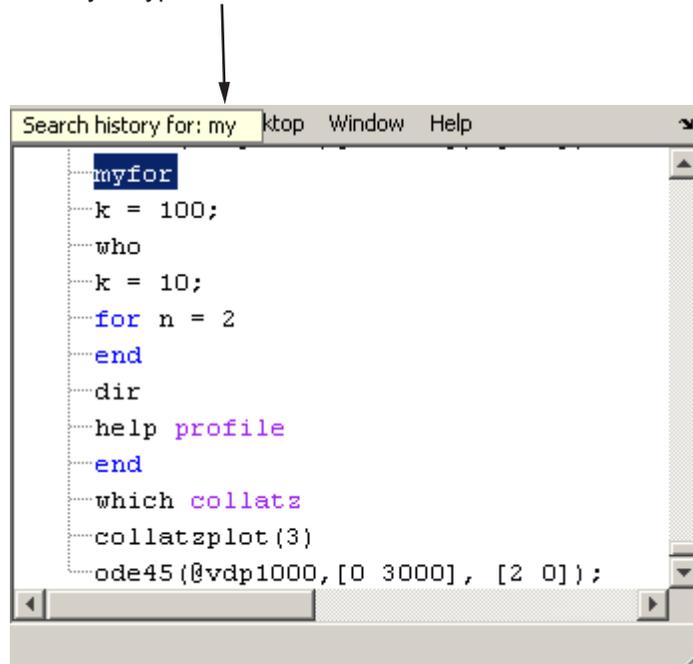
3 Now type an `s` to extend the search to `mys`. The Command History window continues to search backwards, stopping next at the function `mysurf`.
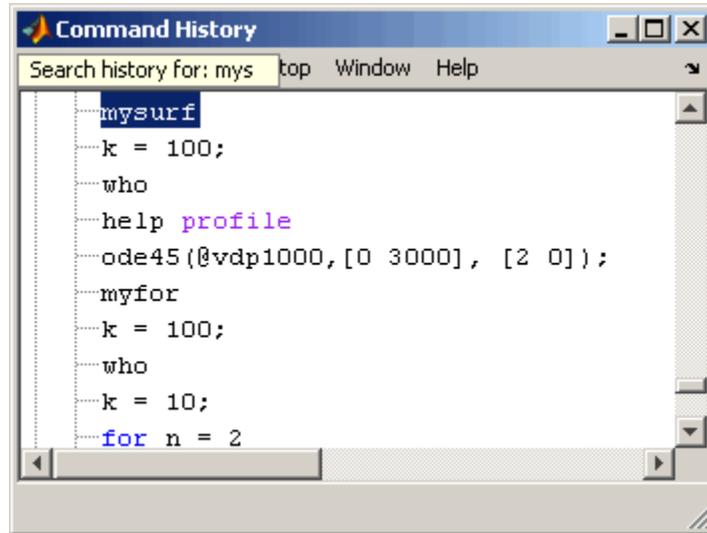
Finding Multiple Occurrences of the Entry. You can use the up and down arrow keys to search for the next or the previous occurrence of the entry you just found.

When you press **Ctrl** and the up or down arrow key, each occurrence of the entry remains highlighted while you search for additional instances.
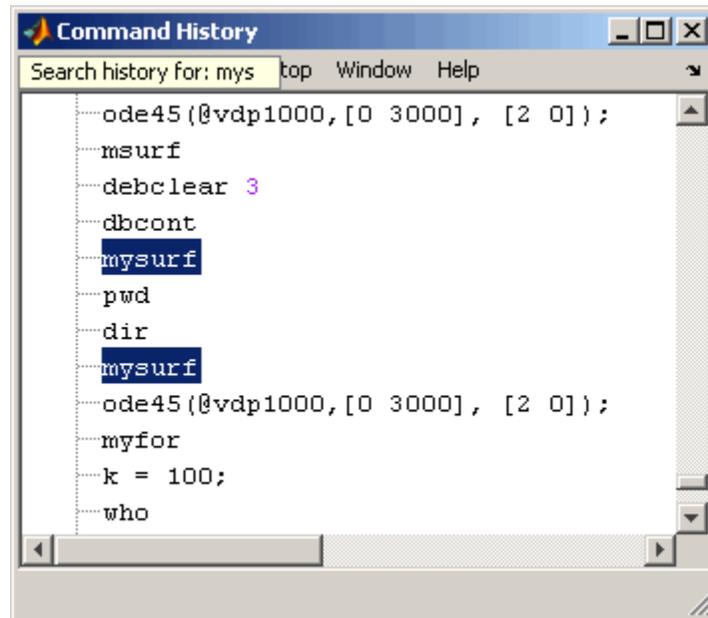
To highlight all instances of the entry, press **Ctrl+A**. In the example below, all instances of entries beginning with my are highlighted.
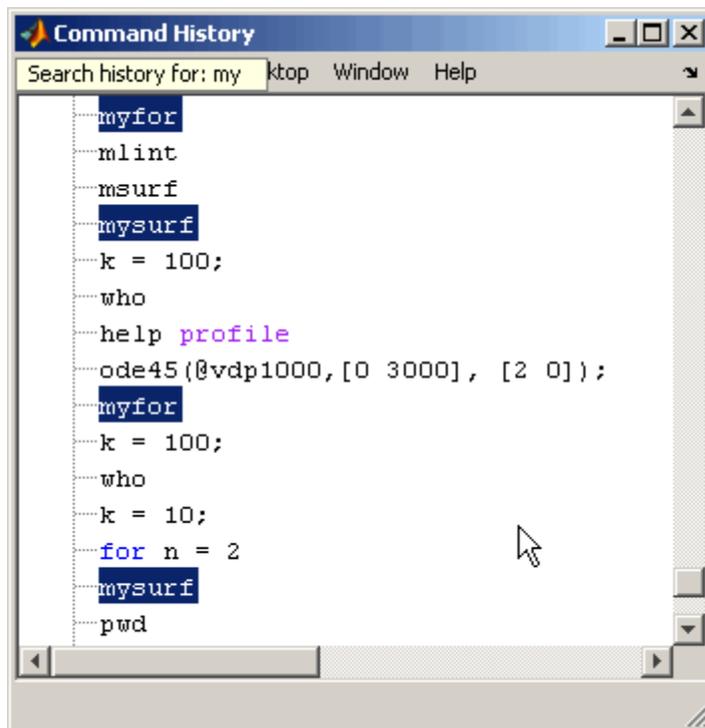
Finding Text

Select **Find** from the **Edit** menu to search for specified text using the Find dialog box. Complete the dialog box. The search begins at the current cursor position. MATLAB finds the text you specified and highlights it. Click **Find Next** or **Find Previous** to find another occurrence, or use the keyboard shortcuts **F3** and **Shift+F3**. Find looks for visible entries only, that is, it does not find entries in collapsed nodes.

Search for specified text in the Command History window.

MATLAB beeps when a search for **Find Next** reaches the end of the Command History window, or when a search for **Find Previous** reaches the top of the Command History window. If you have **Wrap around** selected, it continues searching after beeping.

Change the selection in the **Look in** field to search for the specified text in other MATLAB desktop tools.

Printing the Command History Window

To print the contents of the Command History window, select **File > Print** or **Print Selection**. Specify options for printing by selecting **File > Page Setup**. For example, you can print the history with a header. For more information, see "Printing and Page Setup Options for Desktop Tools" on page 2-52.

The printed version is sized to fit the page. If there is a long statement in the Command History, the reduced page size might be difficult to read. As a workaround, either use **Print Selection**, where the long statement is not part of the selection, or remove any extremely long statements from the Command History before printing it.

Deleting Entries from the Command History Window

Delete entries from the Command History window when you feel there are too many and it becomes inconvenient to find the ones you want. All entries remain until you delete them or until the history.m file exceeds its maximum size, when MATLAB automatically deletes the oldest entries—see "Viewing Statements in the Command History Window" on page 3-50.

To delete entries in the Command History window, first select the entries to delete, using one of these methods:

- Select a single entry.

- **Shift**+click or **Ctrl**+click to select multiple entries.

- Select the timestamp for a session to select all entries for that session. Then use **Shift**+click or **Ctrl**+click to select multiple timestamps with all of their entries.

Then right-click and select **Delete selection** from the context menu, or press the **Delete** key. A confirmation dialog box might appear; see preferences for "Confirmation Dialogs Preferences" on page 2-81 for more information.

To delete all entries, select **Edit > Clear Command History**, or select **Clear Entire History** from the context menu.

After deleting entries from the Command History window, you will not be able to recall those statements in the Command Window as described in "Recalling Previous Lines" on page 3-19.

Preferences for Command History

In this section...

Introduction

Using Command History preferences, you can choose to exclude statements from the `history.m` file and specify how often to save it. The `history.m` file is used for both the Command History window and statement recall in the Command Window.

To set preferences for the `history.m` file, select **File > Preferences**, and then select **Command History** in the Preferences dialog box.

Settings

Specify the types of statements to exclude from the `history.m` file. Note that when you exclude statements from the `history.m` file, you cannot recall them in the Command Window as described in "Recalling Previous Lines" on page 3-19, nor can you view them in the Command History window.

Save Exit/Quit Commands

Select the check box to save `exit` and `quit` commands in the `history.m` file.

Save Consecutive Duplicate Commands

Select the check box if you want consecutive executions of the same statement to be saved to the `history.m` file.

For example, with this option selected, run `magic(5)`, and then run `magic(5)` again. The `history.m` file saves two consecutive entries for `magic(5)`. With this option cleared, for the same example, the command history file saves only

one entry for `magic(5)`. If you then run `magic(10)`, the command history file saves both entries, `magic(5)` followed by `magic(10)`.

Saving

Use **Saving** preferences to specify how often to automatically save the `history.m` file during a MATLAB session.

Save History File On Quit

Select this option to save the `history.m` file when you end the MATLAB session. If the session does not end via a normal termination, that is, via the `exit` or `quit` functions, **File > Exit MATLAB**, or the MATLAB desktop Close box, the history file is not saved for that session.

Save After n Commands

Select this option to save the `history.m` file after `n` statements are added to the file. For example, when you select the option and set `n` to `10`, after every 10 statements are added, the history file is automatically saved. Use this option if you don't want to risk losing entries to the saved history because of an abnormal termination, such as a power failure.

Don't Save History File

Select this option if you do not want to save the `history.m` file. This feature is useful when multiple users share the same machine and do not want other users to view the statements they have run.

Note that any entries already in the `history.m` file remain. Prior to setting this preference, you might want to remove any existing entries. Follow the instructions in "Deleting Entries from the Command History Window" on page 3-57.

See Also

Additional preferences that relate to the Command History are

- "Fonts Preferences for Desktop Tools" on page 2-62
- "Confirmation Dialogs Preferences" on page 2-81

Help for Using MATLAB

The primary means for getting help is the Help browser, which provides documentation for all your installed products. Other forms of help are available including M-file help and Technical Support solutions. If you have an active Internet connection, you can watch the Help and Documentation video demo for an overview of the major functionality.

Help Functions (p. 4-44)   Use functions to get information,
                                            such as `help` and `doc`.

Getting Pop-Up Help for Functions   Get pop-up help (help on selection)
(p. 4-49)   for functions from within the
                                            Editor/Debugger and the Command
                                            Window.

Other Forms of Help (p. 4-51)   Use product-specific help features,
                                            download M-files, contact Technical
                                            Support, see documentation for
                                            other MathWorks products, view a
                                            list of other books, and participate in
                                            a MATLAB newsgroup.

Help Browser Overview

In this section...
"About the Help Browser" on page 4-3
"Opening the Help Browser" on page 4-3
"Resizing the Help Browser" on page 4-5
"Types of Documentation" on page 4-7
"Accessing Documentation on the Web" on page 4-8
"Adding Help Files" on page 4-9
"Documentation in Other Languages" on page 4-9

About the Help Browser

The Help browser is an HTML browser integrated with the MATLAB desktop. Use the Help browser to search and view documentation and demonstrations for MATLAB and all other installed MathWorks products. MATLAB automatically installs the documentation and demos for a product when you install that product.

Opening the Help Browser

To open the Help browser, click the Help button 🔵 in the desktop toolbar, type helpbrowser in the Command Window, or use the **Help** menu in any tool. There are two panes:

- The Help Navigator, on the left, for finding information, includes a **Search for** field, and **Contents**, **Index**, **Search Results**, and **Demos** tabs. For more information, see "Finding Information with the Help Browser" on page 4-10.

- The display pane, on the right, for viewing documentation and demos.

Tabs in the **Help Navigator** pane provide different ways to find information.

Use the close box to hide the pane.

Drag the separator bar to adjust the width of the panes.

View documentation in the display pane.
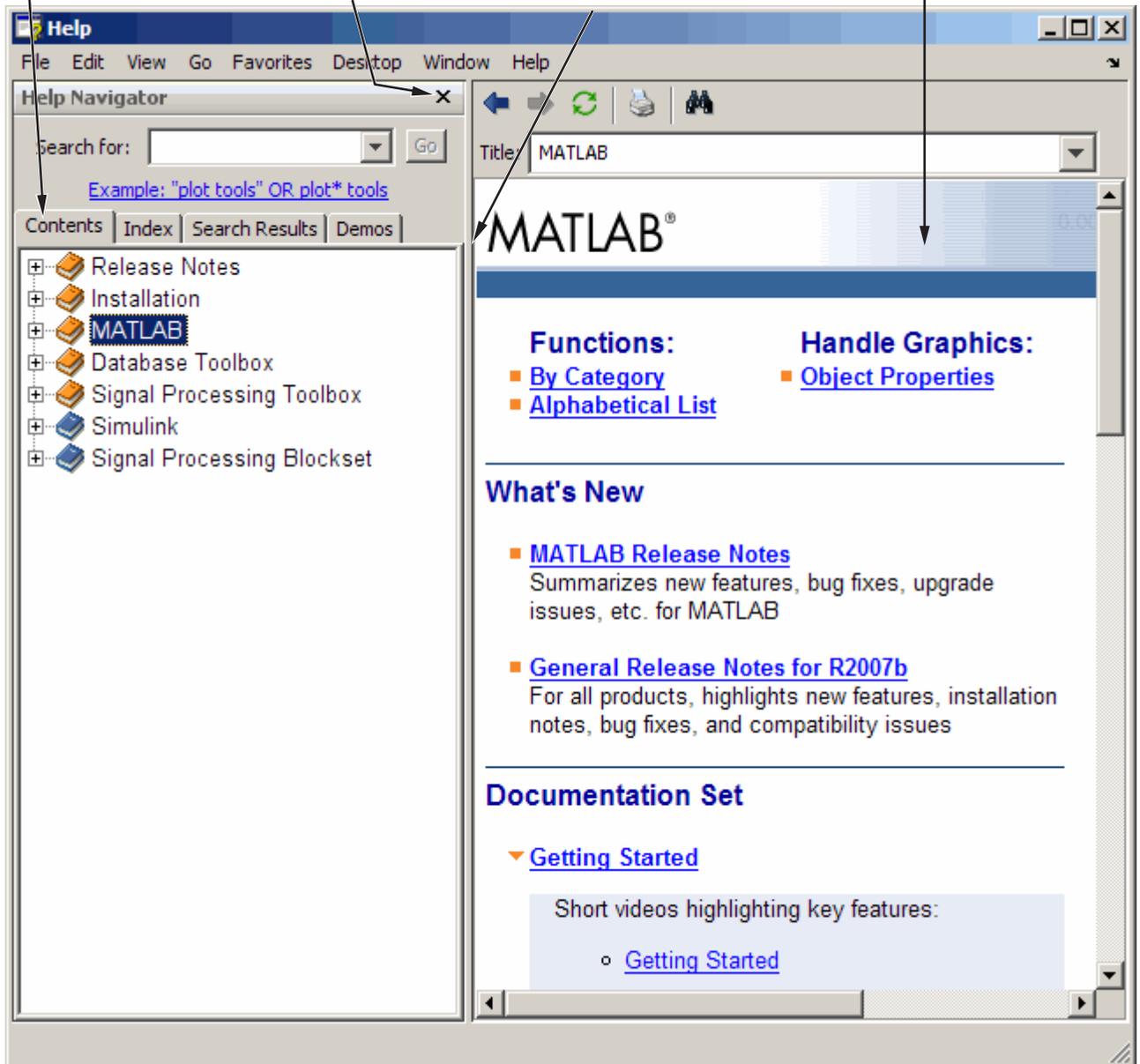
Resizing the Help Browser

To adjust the relative width of the two panes, drag the separator bar between them. You can also change the font in either of the panes — see "Help Fonts and Colors Preferences" on page 4-39.

Once you find the documentation you want, you can close the **Help Navigator** pane so there is more screen space to view the information itself. This is shown in the following figure. To close the **Help Navigator** pane, click the Close box ☒ in the pane's upper right corner. To open the **Help Navigator** pane from the display pane, click the Help Navigator button ▣ on the toolbar. Alternatively, use the **View** menu.

Help

File   Edit   View   Go   Favorites   Desktop   Window   Help

Title: Entering Statements in the Editor/Debugger :: Editing and Debugging M-Files (Desktop Tools and Development E...

Tab Completion in the Editor/Debugger

The Editor/Debugger helps you automatically complete the names of these items as you type them in an M-file:

- Functions or models on the search path or in the current directory
- Variables, including structures, in the current workspace, where the current workspace is shown in the **Stack** on the toolbar.
- Handle Graphics properties for figures in the current workspace

Type the first few characters of the item name and then press the **Tab** key. To use tab completion, you must have the tab completion preference for the Editor/Debugger selected. For details, see Keyboard Preferences.

Tab completion is also available in the Command Window. There are a few minor differences in how tab completion works in the Command Window, the most notable being that Command Window tab completion supports the completion of filenames, whereas the Editor/Debugger tab completion does not.

> **Note**   Tab completion does not complete the names of variables you define in an M-file, but only those variables in the current workspace. This means that while editing, it only completes the names of variables in the base workspace. While debugging, it only completes the names of variables in the current function workspace.

These examples demonstrate how to use tab completion:

- Basic Example — Unique Completion

Types of Documentation

The Help browser and help functions provide access to the following types of information for all installed MathWorks products. The icons shown here appear in the Help browser contents listing to help you quickly identify documentation by type.

Icon	Type of Documentation	Description and When to Use
🟢	Getting Started	Review Getting Started documentation before you begin using a product or feature for the first time. Then, to learn more, go to the user guides, reference pages, demos, and examples.
🔶 or 🔷 or 🟢	Product	MATLAB, toolboxes, and related products use orange book icons 🔶. Simulink, blocksets, and related products use blue book icons 🔷. Link and Target products use green book icons 🟢.
💡	Index of Examples	Accessible via the Help browser **Contents** listing, this is an index of the major examples included in the Help browser documentation.
📄	User Guides (blue)	User guide material contains overviews as well as detailed instructions. Consult it after reviewing Getting Started material.
📄	Reference Pages (orange)	Each function has a reference page that provides the syntax, description, examples, and other information for that function. Each reference page includes links to related functions and additional information. Reference pages are also provided for blocks and properties.
📄	Release Notes	An overview of new products and features in a release. Release Notes also include upgrade information, links to fixed and known problems, and compatibility considerations. Review the Release Notes for all your products when you first start using a new release. Release Notes for the current version include the release notes for multiple prior versions.

Icon	Type of Documentation	Description and When to Use
	Printable Documentation	Most products provide access to the online documentation in a printable format, PDF. Access PDF files via the Help browser and print them from your PDF reader, such as Adobe Acrobat. Most PDF files reside only on the MathWorks Web site, so you need an Internet connection to view them.
none	Demos	MathWorks products come with demonstrations that run key features of the product. Many of the demos run MATLAB code. Use the Help browser **Demos** pane or **Search Results** to access demos for the products you have installed.
none	M-File Help	Get M-file help in the Command Window to quickly access basic information for a function or model. It provides a brief description of a function and its syntax. It is called M-file help because the text of the help is a series of comments at the top of the M-file for a function.

Accessing Documentation on the Web

You can access all product documentation on the MathWorks Web site at
`http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml`.
These are some uses for the Web site version of the documentation:

- Access documentation for products you have not installed.

- Access documentation for the most current version. If you do not see the information you are looking for in the Help browser and know you are not running the most current version of MATLAB, the most current version of the documentation, which is on the Web site, might include more information. Note, though, that the documentation on the Web site might refer to features that are not part of your earlier-version product.

- Access documentation for a prior version of some products (Release 13 with Service Pack 2). Note that the release notes on this page include release notes for multiple prior versions. For example, you can find information about MATLAB Version 6.0 (Release 12) new features and changes.

- Access documentation via your system Web browser, such as when you are not running MATLAB or if you prefer your system Web browser.

To determine the URL for a page in the Help browser, see "View the Page Location" on page 4-30

PDF documentation is available only on the Web site.

You cannot read MATLAB documentation files from the MATLAB installation media. You also cannot use a Web browser to read the documentation files installed with MATLAB because the files are compressed JAR files.

Adding Help Files

You can add your own HTML help files so that they appear in the Help browser. For details, see in the online documentation.

Documentation in Other Languages

The MathWorks documentation is available in English. Japanese versions of MATLAB include documentation that has been translated into Japanese. For more information, go to `http://www.cybernet.co.jp/matlab`.

Finding Information with the Help Browser

In this section...
"Help Navigator" on page 4-10
"Contents in the Help Browser" on page 4-10
"Index for the Help Browser" on page 4-13
"Search Documentation and Demos with the Help Browser" on page 4-16
"Favorites" on page 4-24

Help Navigator

The **Help Navigator** is in the left pane of the Help browser. It provides a table of contents, an index, and a search feature to help you find information.

Contents in the Help Browser

To list the documentation titles and tables of contents for products you installed, click the **Contents** tab in the **Help Navigator** pane. To show documentation for only some of the installed products, use the product filter.
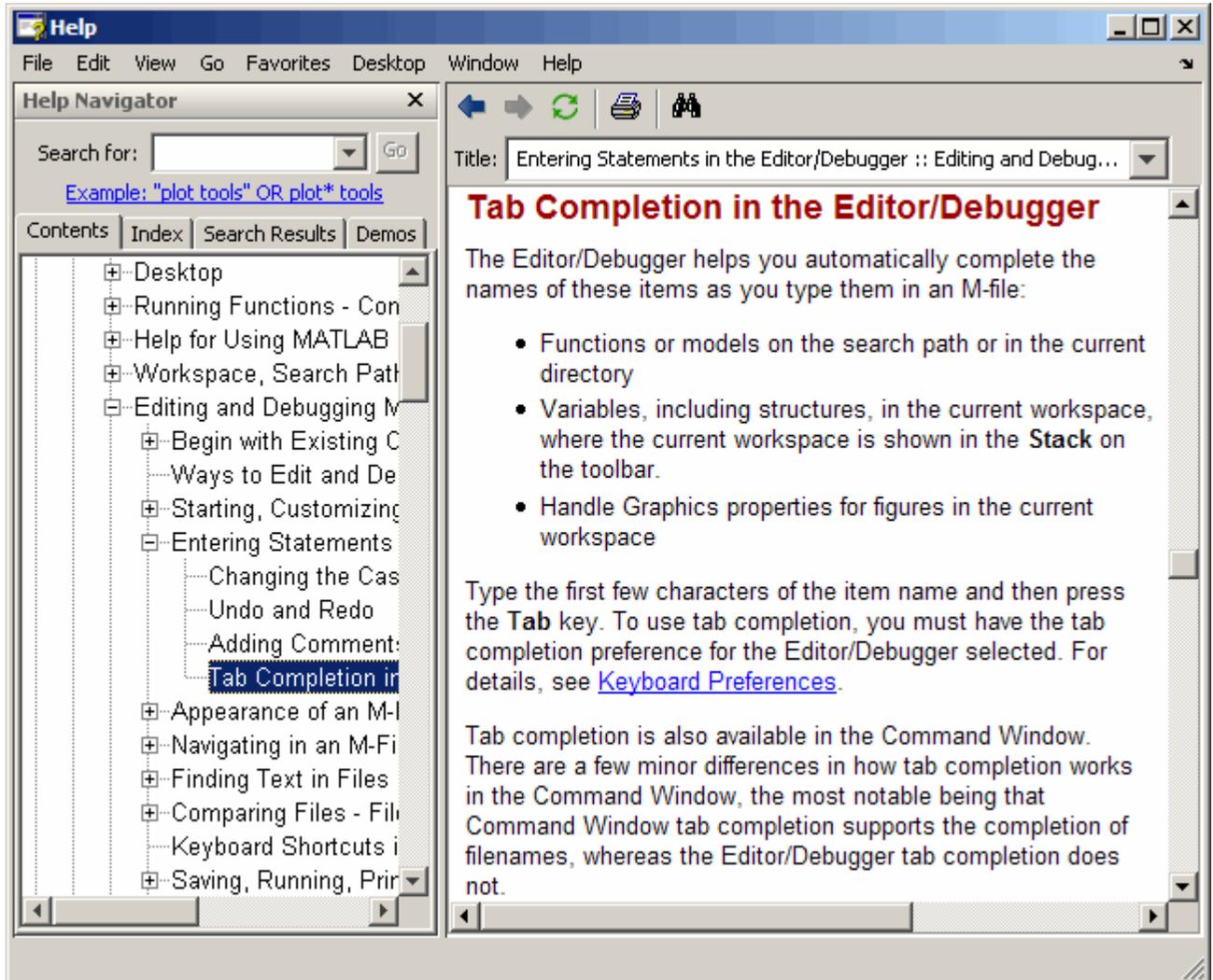
Product Roadmap

When you select a product in the **Contents** pane (any entry with a book icon 📙), such as MATLAB or Communications Toolbox, a *roadmap* of the documentation for that product appears in the display pane. The roadmap includes links to commonly used documentation sections, including

- Function and block references pages

- An index of major examples in the documentation

- The PDF version of the documentation, which is suitable for printing (this is the only direct access from MATLAB to the printable documentation)

Navigate the Contents Listing

In the **Contents** listing, you can

- Click the **+** to the left of an item to show the first page of that document or section in the display pane and expand the listing for that item in the **Help Navigator** pane. You can alternatively: double-click the item, press the right arrow key, or press **+** on the numeric keypad.

- Click the **-** to the left of an item to collapse the listings for that item. You can alternatively: double-click the item, press the left arrow key, or press **-** on the numeric keypad.

- Select an item to show the first page of that document or section in the display pane.

- Press * on the numeric keypad to show all subentries for the selection.

- Use the down and up arrow keys to move through the list of items.

Icons in the Contents Listing

Icons for entries in the top levels of the **Contents** pane listing represent the type of documentation so you can quickly find the kind of information you need for a product. See the table for icons in "Types of Documentation" on page 4-7.

Synchronize the Contents Listing and Demos Listing with the Display Pane

By default, the topic highlighted in the **Contents** pane matches the title of the page appearing in the display pane. The **Contents** pane listing is said to be synchronized with the displayed document. This feature is useful if you access documentation with a method other than the **Contents** pane, for example, using a link in a page in the display pane, or selecting a search result. With synchronization, you know what book and section the displayed page is part of, and where that section fits within the overall book. Note that synchronization

only applies to the major headings in a document. For pages that begin with lower level headings, the **Contents** pane listing does not synchronize.

You can turn off synchronization. To do so, use preferences. See "General — Keep Contents Synchronized" on page 4-38.

Synchronization applies to the **Contents** and **Demos** panes. The page shown in the display pane does not necessarily correspond to the selection in the **Index**, or **Search Results** panes. However, if you return to the **Contents** pane (or **Demos** pane), the displayed page synchronizes with the **Contents** (or **Demos**) pane.

This example illustrates synchronization for after performing a search. When you enter `"interactive plotting"` in the **Search for** field, the **Search Results** pane displays a list of results in the Help Navigator, with the first documentation result selected. The display pane shows the page corresponding to that first result, `Plotting Tools   Interactive Plotting`. When you click the **Contents** tab, the tree automatically expands to show `MATLAB > Graphics > Plots and Plotting Tools` and selects the `Plotting Tools   Interactive Plotting` entry.

This example illustrates synchronization for demos. When you run

```
demo('matlab', 'graphics')
```

the **Demos** pane appears, with the MATLAB Graphics entry selected.

Index for the Help Browser

To find specific index entries (selected keywords) in the MathWorks documentation for installed products, use the **Index** in the **Help Navigator** pane.
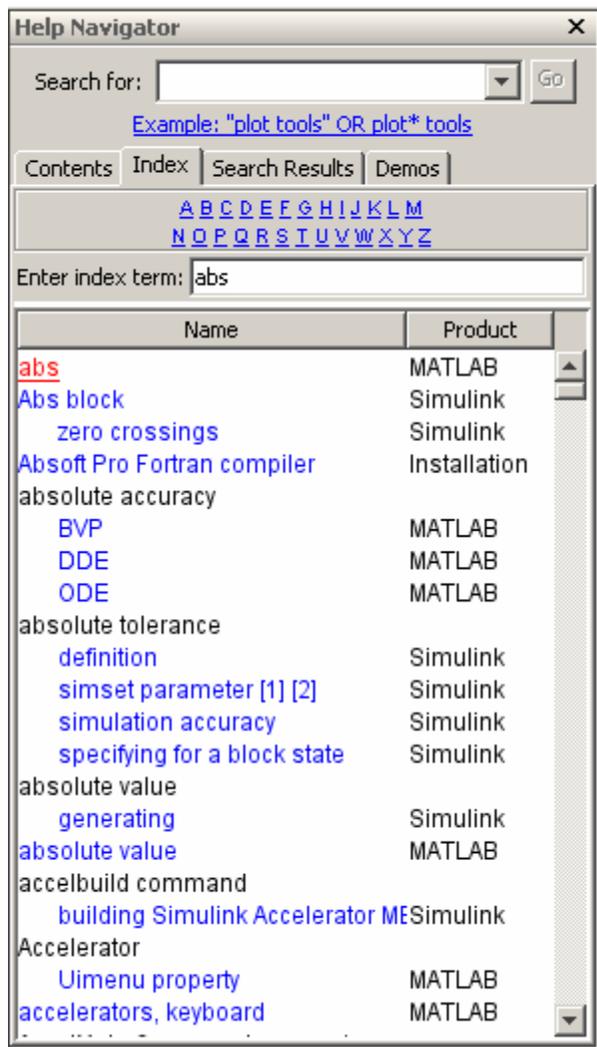
1 Click the **Index** tab.

2 Type a word or words in the **Enter index term** field. As you type, the **Index** pane displays matching entries and their subentries (indented). It might take a moment for the display to appear. The index is not case

sensitive. If there is not a matching entry, it displays the page for the letter that your entry begins with.

The product whose documentation includes the matching index entry is listed next to the index entry, which is useful when there are multiple matching index entries. You might have to make the **Help Navigator** pane wider to see the product.

3 Select a blue index entry from the list (where blue represents a hyperlink) to display the page to which the term refers. Multiple links per entry are denoted by numbers in brackets following the term. (Black index entries are headings and do not link to any page.)

When you select an entry, its color becomes red. The page whose entry you selected appears in the display pane, scrolled to the location that the entry references.

4 To see more matching entries, scroll through the list.

Tips for Using the Index

- To see entries for selected products only, select **File > Preferences > Help** and set the product filter.

- To see entries for all installed products, select **File > Preferences > Help**, and clear the **Enable product filter** check box.

- For more or different results, type a different term or reverse the order of the words you type. For example, if you are looking for an entry about tab completion for `Editor/Debugger`, a subentry does not exist. Instead type `tab completion` and there is a subentry `Editor/Debugger`.

- After selecting an entry, search for specified text in the displayed page using the **Find** tool, accessible from the Find button on the display pane toolbar.

- When there are multiple matching entries, refer to the product associated with each entry, which appears in the second column of the **Index** results. You might need to make the pane wider to see it.

- For different but related results, try using the **Search for** field—for instructions, see "Search Documentation and Demos with the Help Browser" on page 4-16.

- See "Specifying Colors for the Help Browser" on page 4-41 for information about changing the color of hyperlinks in the **Index**.

Search Documentation and Demos with the Help Browser

- "Searching in the Help Browser" on page 4-16

- "Wildcards in Search (Partial Word)" on page 4-21

- "Exact Phrases in Search" on page 4-21

- "Boolean Operators in Search" on page 4-21

- "More About Search" on page 4-22

- "Get Fewer Results" on page 4-22

- "Get More Results" on page 4-23

Searching in the Help Browser

To look for a specific word or phrase in the documentation or demos, use the **Search for** field in the **Help Navigator**.
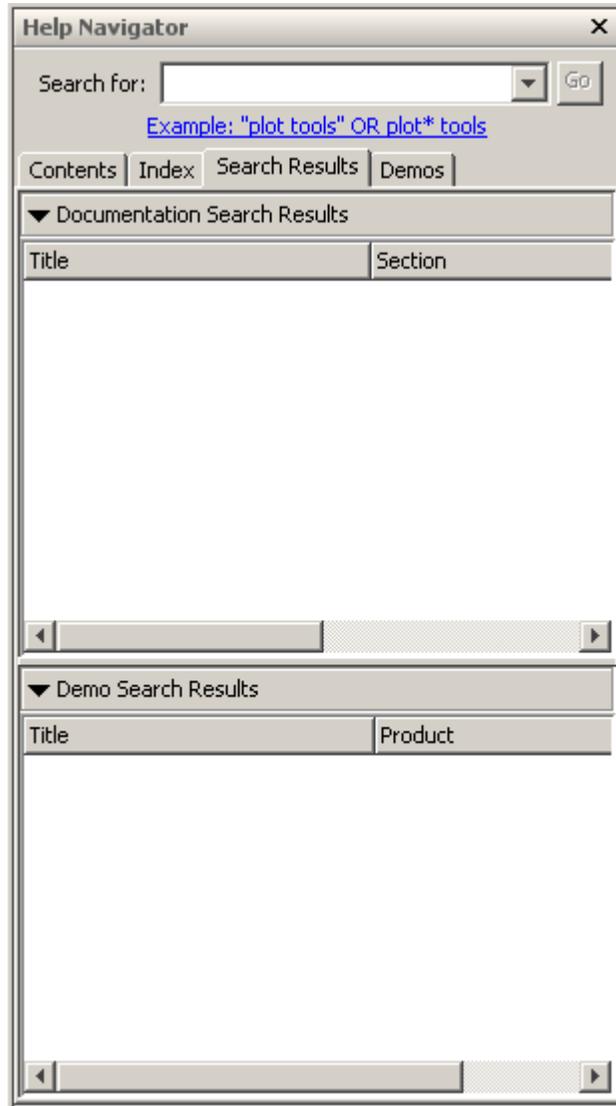
1 To limit (or extend) the products whose documentation and demos are searched, set the product filter.

2 In the **Search for** field, type the word or words you want to find and click **Go** (or press **Enter** or **Return**). Some techniques for honing your search are

- Exact Phrase — Search for an exact phrase by enclosing words in quotation marks, for example, `"plot tools"`. For more information, see "Exact Phrases in Search" on page 4-21.

- Wildcard (Partial Word) — Search for variations of a word, also called partial word searching, by using the wildcard symbol (`*`) in place of letters in a word, for example, `plot* tools`. For more information, see "Wildcards in Search (Partial Word)" on page 4-21.

- Boolean Operators — Add the Boolean operators `AND`, `OR`, and `NOT` between search words to include or exclude words. By default, search assumes an `AND` between all search words and exact phrases. For more information, see "Boolean Operators in Search" on page 4-21.

The documents and demos containing all of the search words are listed in the **Search Results** tab. Two sets of results appear: **Documentation** and **Demo**. The number of results for each appears in parentheses at the top of the listing, and the total number for both sets of results appears in the lower left corner of the Help browser. Both sets of results have additional columns that list the **Product**, and for documentation, another column lists the **Section**. You might need to make the **Help Navigator** pane wider to see all columns.

3 Select an entry from the list of results. By default, the first documentation entry is automatically selected. If there are no documentation results, but there are demo results, the first demo entry is automatically selected.

The selected page appears in the display pane with all occurrences of the search words and exact phrases highlighted, using a different color for each search word or phrase. Highlights remain until you view another page or until you click the Refresh button ⟳ in the toolbar.

In the display pane, use the **Find** tool, accessible from the Find button 🔍 on the toolbar, to find a specified word in that page.

4 Search results are ordered by relevance. For example, for documentation, reference pages that match the search term appear first, followed by titles

that contain all search words, with pages containing a single instance of each search word appearing last. Change the display of search results to more easily find the most relevant results:

- Sort by column — Change the order of the results by clicking a column heading. For example, click **Product** to group results by product. Click **Title** to sort titles alphabetically. A triangular icon indicates the column on which you most recently sorted. Click the heading again to sort by the column but in the reverse order.

  After changing the order of results, to see results ordered by relevance, click **Go** to rerun the search.

- Reorder columns — Change the location of a column by dragging its heading to a new position. For example, you can drag the **Product** column to the middle for documentation results.

- Resize columns — Make columns wider or narrower by dragging the separator bar between the column headings. Similarly, make the Help Navigator wider or narrower.

5 For more results, you can search for the words in the Technical Support database of bug reports, solutions, and notes on the MathWorks Web site by clicking the link at the bottom of the **Search** results pane.
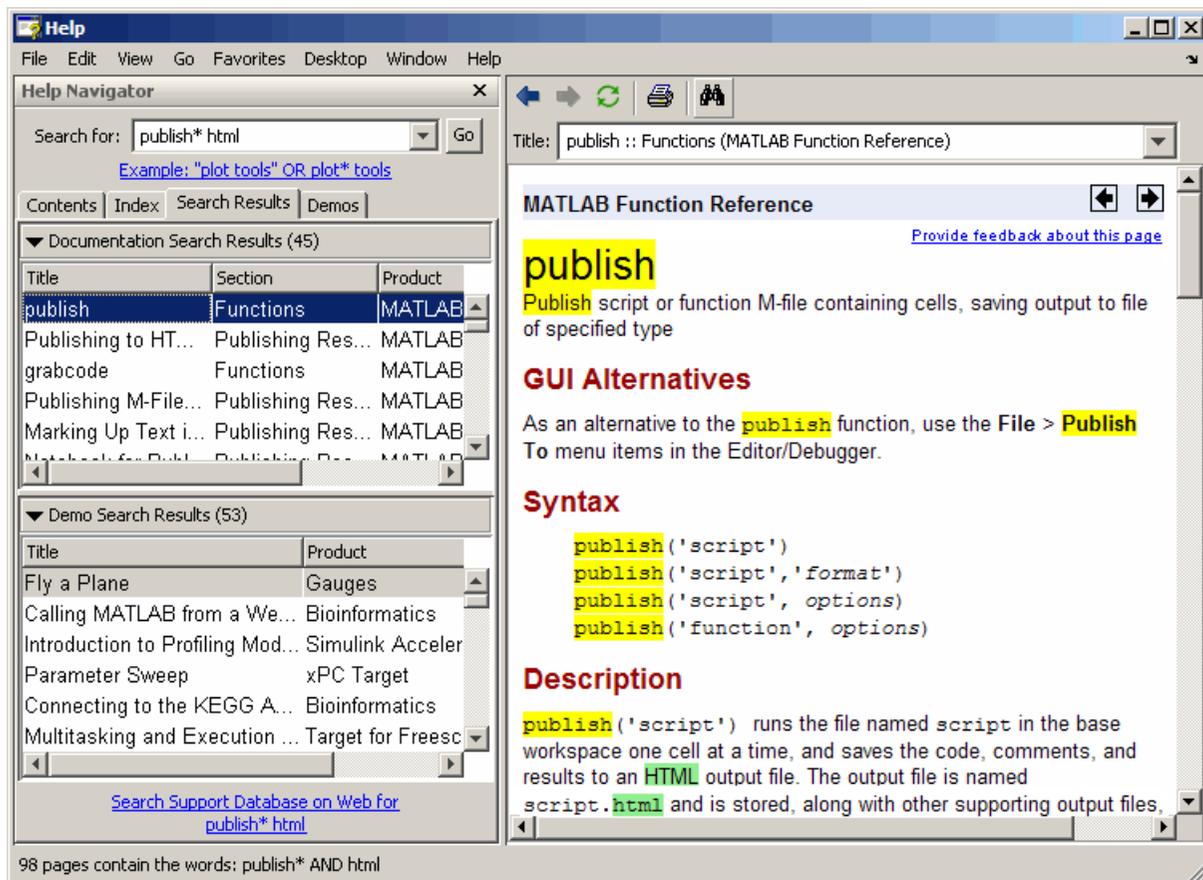
Function Alternative. From the Command Window, use `docsearch` to open the Help browser to the **Search** pane and search for the specified term. For example

```
docsearch('publish* html')
```

finds all pages that contain the word `publish` or its variations, such as `publishing`, `published`, and so on, and also contains the word `html`.

```
docsearch('"plot tools"')
```

finds all pages containing the exact phrase `plot tools`.

For details, see the `docsearch` reference page.

Wildcards in Search (Partial Word)

You can use the wildcard character (`*`) in place of letters or digits in your search terms. For example, `plot*` finds various forms of the word *plot*, such as *plot*, *plots*, and *plotting*. The search term `p*t` also finds those variations of *plot* as well as variations of *print* and *part*, among others.

You can use multiple wildcards in a word or search term. For example, `plot* tool*` finds *plotting tools*, among other terms. The term `p*t* tool*` finds not only *plotting tools*, but also *pointer tooltip*.

You cannot use a wildcard with just one letter or digit, nor can you include wildcards within an exact phrase. You cannot begin a word in a search term with a wildcard character. For example, these fail: `p*`, `"plot* tools"`, `plot *ool`.

Exact Phrases in Search

To find a phrase, type quotation marks around it. For example, `"plot tools"` finds only pages that include `plot tools` together, but does not find pages that include `plot` in one part of the page and `tools` in another part of the page. Specify an exact phrase to reduce the number of irrelevant results. For example, `"plot tools"` finds about 10 pages in MATLAB documentation, while `plot tools` finds about 100 pages.

You can specify more than one exact phrase, such as `"plot tools" "figure palette"` to find pages that contain both `"plot tools"` and `"figure palette"`. You cannot use a wildcard within an exact phrase.

Boolean Operators in Search

The search automatically performs a Boolean AND for multiple words. In the example `publish* html`, it finds all pages that have the word `publish` or its variations, and the word `html`.

You can refine the search by including the Boolean operators `NOT`, `OR`, and `AND` between words. The operators must be in all capital letters and there must be a space before and after each operator. The `NOT`s are evaluated first, followed by the `OR`s, and then the `AND`s.

Example Using Boolean Operators in Search. Type

```
plot* tools NOT time series
```

to find all pages that contain the words `plot` or its variations and `tools`, but not the phrase `time series`.

More About Search

These are the guidelines search uses:

- Insignificant words (a, an, the, of) are ignored.

- Search is not case sensitive.

- Search only finds letters and digits, but not symbols. To find a symbol, look for the word (for example, `plus` instead of +), use the **Index**, or see Operators and Special Characters in the MATLAB Functions — By Category. Another option is to search the PDF documentation, which supports searching for symbols — instructions to access the PDF file are included in "Printing the PDF Version of Documentation" on page 4-42.

- Search find words in comments or code for M-file and Model types demos. It finds comments in the M-file help for M-GUI demos. It does not search video demos.

- If you search for a function that is used in multiple products (called an overloaded function), the reference pages for all those products are listed. Use the **Product** column in **Search Results** to determine the reference page you want.

Get Fewer Results

If there are too many results for the search to be useful, try the following.

Problem	Try These Suggestions
Too many products	Select **File > Preferences > Help** and enable the product filter for specified products. For details, see "Product Filter" on page 4-37.
	Order results by product — click the **Product** column in **Search Results**. If you cannot see the column, make the pane wider.

Problem	Try These Suggestions
Pages are not about search word, but just mention it	Try the **Index** pane to see more important entries for that search word.
Too many irrelevant results	Type more than one word in the **Search for** field. Look for an exact phrase by enclosing words in quotations marks, such as `"plot tools"`. Use Boolean operators (in all capitals), for example, `printing AND figures NOT exporting`.
Topic is not relevant	Look at the **Section** column in **Search Results**, which provides context for the result. If you cannot see the column, make the pane wider.
Want to look only within part of a product's documentation	For products like MATLAB, you might want to search only part of the documentation. There is no way to do this in the Help browser. However, you might be able to accomplish that via PDF search. For example, you can search the "Getting Started with MATLAB" PDF file, or the "MATLAB External Interfaces" PDF file. Instructions to access the PDF file are included in "Printing the PDF Version of Documentation" on page 4-42.

Get More Results

If you want more results, try the following.

Problem	Try These Suggestions
No results for the product	Be sure the product filter is set correctly. Select **File > Preferences > Help** and disable the product filter, or at least ensure the products of interest are selected. For details, see "Product Filter" on page 4-37.

Problem	Try These Suggestions
No results but you know the word should be there	Try variations of the search word by using a wildcard symbol (`*`). For example, search for `preference*` to find all pages that contain either the word `preference` or the word `preferences`.
Not enough information	Try searching the Technical Support database of bug reports, solutions, and technical notes by clicking the link at the bottom of the **Search** results pane.
	If you are not running the most current version of MATLAB, try looking at the most current version on the Web site. It might contain additional information. For more information, see "Accessing Documentation on the Web" on page 4-8.

See Also. "Finding Files and Content Within Files" on page 5-49, which describes the Find Files tool you use to look for files and content within files, such as comments in M-files or code fragments.

Favorites

Favorites are bookmarks to pages in the Help browser documentation and M-file type demos.

Add Favorites

To designate the displayed page as a favorite (that is, to bookmark it),

1 Select **Favorites > Add to Favorites**.

2 The **Favorites Editor** dialog box opens. You can accept the defaults and click **Save**, or make changes to the entries:

  a Use the **Label** provided, or change it to another term.

  b Do not change the entry for **Callback**.

  c Maintain the **Category** as Help Browser Favorites so you can access them from the **Favorites** menu.

  d For **Icon**, keep the default Help icon, or choose another.

A favorite is implemented as a MATLAB shortcut, so the dialog box is the same as for the **Shortcut Editor**.

Favorites from previous releases are not migrated to a new release.

Go to Favorites

Select the **Favorites** menu to view the list of pages you previously designated as favorites (bookmarks). Select an entry and that page appears in the display pane.

Organize Favorites

You can rename, remove, and reorder the list of favorites. Select **Favorites > Organize Favorites**. For more information, click **Help** in the **Organize Favorites** dialog box.

Viewing Documentation in the Help Browser

About the Display Pane

After finding documentation with the **Help Navigator**, view the documentation in the display pane. The following illustration shows the **Help Navigator** closed to provide a larger area for viewing the information.

Help

File   Edit   View   Go   Favorites   Desktop   Window   Help

Title: | Entering Statements in the Editor/Debugger :: Editing and Debugging M-Files (Desktop Tools and Development E... |

Tab Completion in the Editor/Debugger

The Editor/Debugger helps you automatically complete the names of these items as you type them in an M-file:

- Functions or models on the search path or in the current directory
- Variables, including structures, in the current workspace, where the current workspace is shown in the **Stack** on the toolbar.
- Handle Graphics properties for figures in the current workspace

Type the first few characters of the item name and then press the **Tab** key. To use tab completion, you must have the tab completion preference for the Editor/Debugger selected. For details, see Keyboard Preferences.

Tab completion is also available in the Command Window. There are a few minor differences in how tab completion works in the Command Window, the most notable being that Command Window tab completion supports the completion of filenames, whereas the Editor/Debugger tab completion does not.

> **Note**   Tab completion does not complete the names of variables you define in an M-file, but only those variables in the current workspace. This means that while editing, it only completes the names of variables in the base workspace. While debugging, it only completes the names of variables in the current function workspace.

These examples demonstrate how to use tab completion:

- Basic Example — Unique Completion

Browse to Other Pages

Use the arrow buttons in the page and in the toolbar to go to other pages.

View the next page in a document by clicking the Next page button ▣ at the top or bottom of the page. View the previous page in a document by clicking the Previous page button ▣ at the top or bottom of the page. These arrows allow you to move forward or backward within a single document. The arrows at the bottom of the page are labeled with the title of the page they go to.

View the page previously shown by clicking the Back button ← in the display pane toolbar. After using the Back button, view the next page shown by clicking the Forward button → in the display pane toolbar. These buttons work like the forward and back buttons of popular Web browsers. You can also go back or forward by right-clicking a page and selecting **Back** or **Forward** from the context menu.

Links

Click links in the displayed page to go to a related topic for more information on the subject. Links appear underlined and in blue, while visited links appear in purple. Links to Web addresses open in the MATLAB Web Browser. Click the middle mouse button to open the linked page in a separate window.

Find Text in Displayed Pages

To find a phrase in the currently displayed page,

1 Click the Find button 🔍. In the resulting **Find** dialog box, type the word or phrase you are looking for. You can type a partial word, for example, `preference` to find all occurrences of `preference` and `preferences`. Use the check boxes to specify options. Click **Find Next**.

   The search begins at the current cursor position and the page scrolls to the first occurrence of the phrase in the page and highlights it.

2 To find more occurrences in that page, click **Find Next** or **Find Previous** in the **Find** dialog box, or use the keyboard shortcuts **F3** and **Shift+F3**.

   MATLAB beeps when a search for **Find Next** reaches the end of the page, or when a search for **Find Previous** reaches the top of the page. If you have **Wrap around** selected, it continues searching after beeping.

You can change the selection in the **Look in** field to search for the specified text in other MATLAB desktop tools.

See "Search Documentation and Demos with the Help Browser" on page 4-16 for instructions on looking through all the documentation instead of just one page.

Copy Information

To copy information from the display pane, such as code in an example, first select the information. Then right-click and select **Copy** from the context menu. You can then paste the information into another tool, such as the Command Window or Editor/Debugger, or into another application, such as a word processing application.

Evaluate a Selection

To run code examples that appear in the documentation, select the code in the display pane. Then right-click and select **Evaluate Selection** from the context menu. The statements execute in the Command Window.

Open a Selection

In a page in the display pane, select the name of a file that is provided with MATLAB, such as an M-file. Then right-click and select **Open Selection** from the context menu. The file opens in MATLAB. For example, an M-file opens in the Editor/Debugger.

Help on Selection

In a page in the display pane, select the name of a function that is provided with MATLAB. Then right-click and select **Help on Selection** from the context menu. The reference page for that function opens in Help browser.

View the Page Source (HTML)

To view the HTML source for the currently displayed page, select **View > Page Source**. A read-only HTML version of the page appears in a separate window. You can copy selections from the HTML source and paste

them into other tools like the Editor/Debugger or Command Window, or into other applications.

View the Page Location

To view the location of the page currently displayed, select **View > Page Location**. The Help Page Location dialog box appears, providing the full path to the documentation file for both your local system and the MathWorks Web site.
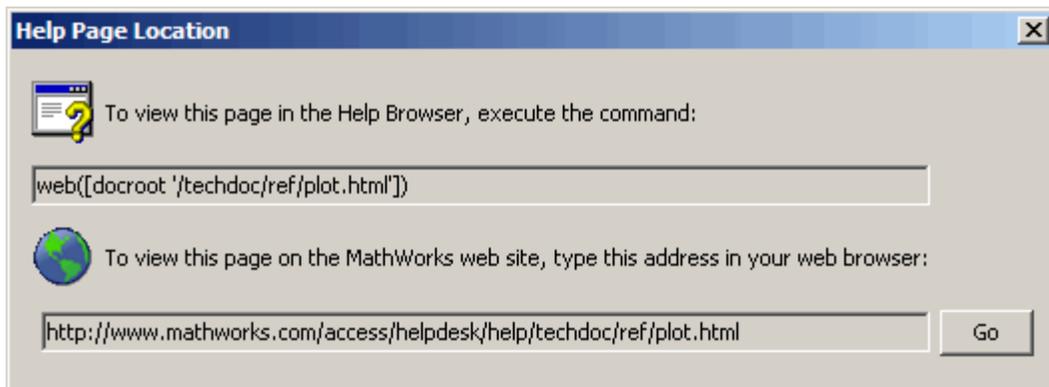
You can copy the information from this window into an e-mail message or other tool to facilitate communication with other users or The MathWorks. For example, if you find a page of documentation that you know would be useful to a colleague running MATLAB, send them the link so they can view the page in the Help browser. Note that the docroot function used with the web function is unsupported, intended only for use by MathWorks products.

Click the **Go** button to view the same documentation page on The MathWorks Web site. This is useful if you do not see the information you are looking for on the page in view and know you are not running the most current version of MATLAB. The documentation for the most current version is on the Web site and might include more information than the documentation for your version. Note, though, that the documentation on the Web site might refer to features that are not part of your earlier-version product. See "Accessing Documentation on the Web" on page 4-8 for more information.

Demos in the Help Browser

In this section...

About Demos

MATLAB and related products include demos that you can access from the Help browser **Demos** pane.

There are four types of demos:

- M-file: Demos that tell a step-by-step story, including source code, commentary, and output. They are published from M-file scripts to HTML output using the Editor/Debugger. The first comment line of the demo M-file begins with two comment symbols (%%), and similarly, two comment symbols (%%) create a cell for each step. The MATLAB Graphics Square Wave from Sine Waves demo is an M-file type demo.

- M-GUI: Stand-alone tools for exploring a feature. An example is the MATLAB Graphics Vibrating Logo demo.

- Model: Simulink block diagrams. An example is the Engine Timing Simulation demo.

- Video: Movies that highlight key features in a tool. They play in your system browser and require the Macromedia Flash Player plug-in. Some also require an Internet connection. An example is the MATLAB Desktop and Command Window demo.

The MATLAB code and Simulink blocks used in the demos (except videos) are available for you to view and copy for use in your own applications.

See also Examples for each product in the **Contents** pane. These examples are similar to demos but are integrated in the documentation.

Using Demos

To access demos for the products you have installed,

1 Click the **Demos** tab in the **Help Navigator**.

You can also access demos from the **Start** button, by using the `demo` function, or from the **Help** menu for some tools.

2 Click the **+** for a product area to list the products or categories that have demos. Then click **+** for a product or product category to list its demos.

All demos for that product or product area are listed in the display pane, and each includes the type of demo along with a thumbnail image that represents output from the demo.

3 Select a specific demo. Information about the demo appears in the display pane.

4 You can then view and run the demo, with specific options depending on the type of demo:

- For M-file demos, click the **Open filename in the Editor** link at the top left. This opens the M-file in the Editor/Debugger. From the Editor/Debugger, run the demo step by step by selecting **Cell > Evaluate Current Cell and Advance**.

  You can also click **Run in the Command Window**, and then follow the instructions that appear in the Command Window. You might need to scroll up to see all of the instructions.

  See also "Running Demos and Base Workspace Variables" on page 4-35.

- For M-GUI demos, click the **Open filename in the Editor** link at the top left. This opens the M-file in the Editor/Debugger.

  Click the **Run this demo** link at the top right to start the GUI. Then follow the instructions in the GUI to proceed through the demo.

- For Model demos, click **Open this model** to open the block diagram.

- For Video demos, click the **Run this demo** link in the top right to play the video. These demos run in your system browser and require the Macromedia Flash Player plug-in. Some also require an Internet connection.

When you double-click a demo name in the **Help Navigator** pane, the demo file opens for M-file and Model demos, or runs for M-GUI and Video demos.

The following example shows the results of running the MATLAB Graphics Square Wave from Sine Waves demo (`xfourier`). In it, MATLAB generates a series of plots, culminating in the final one shown here.
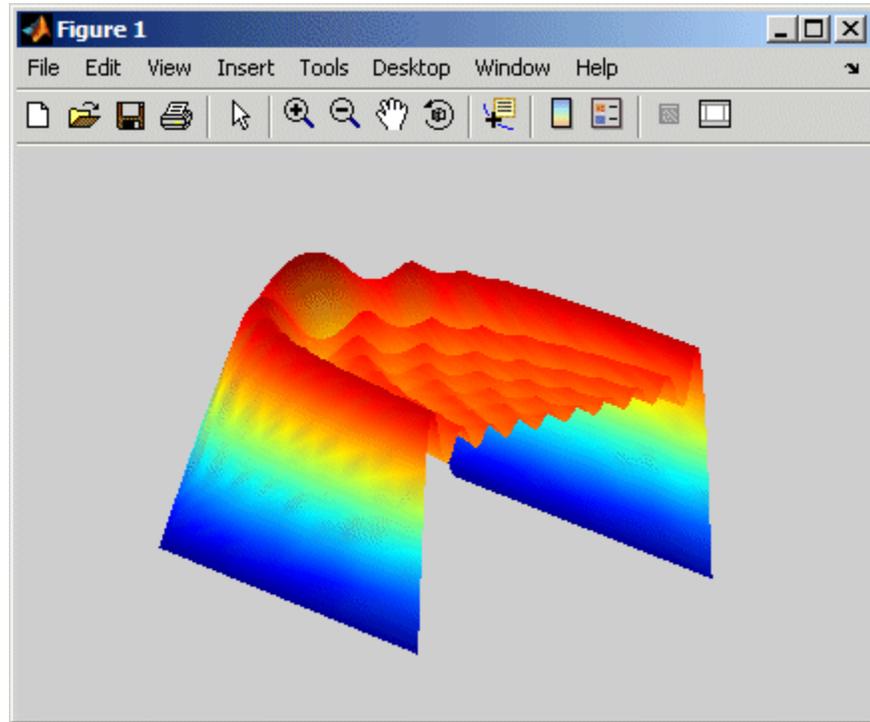
Searching for Demos

You can use the Help browser search feature to find demos. Search find words in comments or code for M-file and Model types demos. It finds comments in the M-file help for M-GUI demos. It does not search video demos. For instructions, see "Search Documentation and Demos with the Help Browser" on page 4-16.

Running Demos and Base Workspace Variables

M-file demos run as scripts. Their variables are created in the MATLAB base workspace. If you have variables in the base workspace when you run an M-file demo, and the demo uses an identical variable name, there could be problems due to variable name conflicts. For example, a variable of yours might be overwritten by the demo. The demo's variables remain in the base workspace until you clear them or quit MATLAB.

Function Alternative

To open the **Demos** pane in the Help browser, type demo in the Command Window. You can go directly to the demos for a specific product. For example

```
demo toolbox signal
```

opens the **Demos** listing for Signal Processing Toolbox.

To run an M-GUI demo, type the demo name in the Command Window. For example, type

```
vibes
```

to run the MATLAB Graphics demo showing an animated L-shaped membrane.

To run an M-file demo step by step from the Command Window, type echodemo followed by the demo name. For example, run

```
echodemo xfourier
```

Typing the demo name for an M-file demo runs the demo, but not step by step.

Typing the name of a model demo opens the block diagram.

Adding Your Own Demos

You can add your own demos so they appear in the **Demos** pane. For details, see in the online documentation.

Preferences for the Help Browser

In this section...
"Product Filter" on page 4-37
"PDF Reader — Specifying Its Location" on page 4-38
"General — Keep Contents Synchronized" on page 4-38
"Help Fonts and Colors Preferences" on page 4-39

Product Filter

If you have MathWorks products in addition to MATLAB, such as Simulink, toolboxes, and blocksets, set the product filter to limit the product documentation and demos used:

1 Select **File > Preferences > Help**.

2 Under **Product filter**, select the check box for **Enable product filter**. Click **Select products**.

   The **Help Product Filter** dialog box opens.

3 Select the products whose documentation and demos you want to appear in the **Help Navigator**. Click **OK**.

   The **Help Navigator** updates to include only those products you specified. The product filter settings are saved for your next MATLAB session.

4 When you want to use documentation and demos for all installed products, in **Help Preferences**, clear the check box for **Enable product filter**.

The Release Notes entry in the **Help Product Filter** dialog box applies to the Release Notes overview document for a release, for example, all products in R2007b, not to the Release Notes for an individual product, for example, MATLAB Release Notes for R2007b. Release Notes for a product are considered part of the product's documentation. For example, MATLAB Release Notes are considered part of MATLAB, and Database Toolbox Release Notes are considered part of Database Toolbox when you use the **Help Product Filter**.

Example Using the Product Filter

If you want to perform a search and have many products installed but know the information you are seeking is in MATLAB or Communications Toolbox, in the **Help Product Filter**, click **Clear All** and then select MATLAB and Communications Toolbox.

The **Contents** only shows MATLAB and Communications Toolbox documentation, the **Index** only shows entries for MATLAB and Communications Toolbox, and the **Search for** feature only looks in and shows results for MATLAB and Communications Toolbox documentation and demos. **Demos** lists only demos for MATLAB and Communications Toolbox.

PDF Reader — Specifying Its Location

If you want to view the PDF version of the documentation, use the link on the roadmap page for that product. To open the PDF file, the Help browser needs to know the location of your PDF reader (for example, Adobe Acrobat).

For Windows systems, MATLAB reads the PDF reader location from the registry, so you do not specify its location.

For UNIX systems, the default PDF reader is Acrobat and MATLAB determines its location. If a different command starts your PDF reader, specify it using preferences. Select **File > Preferences > Help**, and enter the full pathname in the **PDF reader** field or use the Browse for Folder (**...**) button to navigate your file system to select it.

General — Keep Contents Synchronized

By default, the displayed page is synchronized with the **Contents** or **Demos** listings. For more information about this feature, see "Synchronize the Contents Listing and Demos Listing with the Display Pane" on page 4-12.

To turn synchronization off, select **File > Preferences > Help**. Under **General**, clear the check box for **Keep contents tree synchronized with displayed document**. Select the check box to turn synchronization back on.

Help Fonts and Colors Preferences

Set fonts and colors for the Help browser the same way you would for other desktop tools. This section describes the process for the Help browser:

- "Specifying Font Name, Style, and Size" on page 4-39
- "Specifying Colors for the Help Browser" on page 4-41

Specifying Font Name, Style, and Size

You can specify the font name (also called font type or family), style, and size used in the **Help Navigator**.

For the display pane, you can specify the font name and size for the text font, but changes do not impact the style. For the code font, your changes to size apply, but changes to name and style have no impact. The following example shows the results of specifying Comic Sans MS, bold, 14 point for the text font; note that the bold has no effect.

Help

File   Edit   View   Go   Favorites   Desktop   Window   Help

Title: eig :: Functions (MATLAB Function Reference)

MATLAB Function Reference

Provide feedback about this page

eig

Find eigenvalues and eigenvectors

Syntax

```
d = eig(A)
d = eig(A,B)
[V,D] = eig(A)
[V,D] = eig(A,'nobalance')
[V,D] = eig(A,B)
[V,D] = eig(A,B,flag)
```

Description

$d = eig(A)$ returns a vector of the eigenvalues of matrix A.

$d = eig(A,B)$ returns a vector containing the generalized eigenvalues, if A and B are square matrices.

Use the same method as you would to specify fonts for any desktop tool — for more information, see "Fonts Preferences for Desktop Tools" on page 2-62. By default, the **Help Navigator** uses the desktop text font. The display pane is considered to be an HTML Proportional Text tool, and by default, uses the desktop text font.

This example changes the display pane font:

1 Select **File > Preferences > Fonts > Custom**.

2 From the **Desktop tools** list, select HTML Proportional Text. The Help browser display pane is considered to be an HTML Proportional Text tool, as is the MATLAB Web Browser. Changing the font preference affects both tools.

3 For **Font to Use**, select **Custom**, and then specify the font characteristics:

- Name, for example, Comic Sans
- Size in points, for example, 14

   After you make a selection, the **Sample** area shows how the font will look.

4 Click **OK**. The Help display pane fonts use the new settings. The MATLAB Web Browser fonts also use the new settings.

Specifying Colors for the Help Browser

You can specify the background and text color used in the **Help Navigator**. Use the same method as you would to specify the background color for any desktop tool — for more information, see "Colors Preferences for Desktop Tools" on page 2-70.

If the background color preference for your desktop tools is a dark color, you might not be able to see index entries in the **Help Navigator** because they are links, for which the default color is blue. To see the links, change the **Hyperlink** color preference to a light or other contrasting color — for more information, see "Other Colors" on page 2-75.

You cannot specify colors for the Help browser display pane.

Printed Documentation

In this section...
"About Printed Manuals" on page 4-42
"Printing a Page from the Help Browser" on page 4-42
"Printing the PDF Version of Documentation" on page 4-42

About Printed Manuals

Generally, printed manuals are not provided for most products and tools. The printed manuals typically contain less information, and is also sometimes less current than the online documentation. If you want to purchase printed documentation, see the online store at the MathWorks Web site at `http://www.mathworks.com`.

Printing a Page from the Help Browser

To print the page currently shown in the Help browser, select **File > Print**, or click the Print button 🖨 in the display pane toolbar. The Print dialog box appears.

Select **All** in the Print dialog box to print the entire page shown in the display pane. Specifying a range of pages, for example, `1 to 3`, prints the first three pages of the page currently shown in the display pane.

Complete the dialog box and click **OK** to print.

Printing the PDF Version of Documentation

If you need to print more than a few pages of documentation, or if you want the pages to appear as if they came from a printed book, print the PDF version of the documentation. PDF documentation is shown and printed using your PDF reader, usually Adobe Acrobat Reader. The PDF documentation reproduces the look and feel of a printed book. In the PDF document, use links from the table of contents, index, or within the document to go directly to the page of interest within that document. Note that some documentation available from the Help browser is not available in PDF format.

> **Note** The Help browser accesses PDF documentation from the MathWorks Web site. Therefore, you need Internet access to view or print PDF documentation.

1 In the Help browser, click the **Contents** tab and select a product, for example, MATLAB.

The roadmap page opens for that product, providing links to key documentation for that product.

2 Near the end of the roadmap page, listed under **Printing the Documentation Set**, are links for printing the documentation. Select the link for the item you want to print.

The selected document is accessed from the MathWorks Web site. Your PDF reader opens, displaying the documentation.

If you are using a UNIX platform and cannot open the PDF documentation, check the Help preferences. See "PDF Reader — Specifying Its Location" on page 4-38 for more information.

3 To print the documentation, select **Print** from the **File** menu in your PDF reader.

Help Functions

In this section...
"About Help Functions" on page 4-44
"Summary Table of Help Functions" on page 4-44
"View Function Reference Pages — the doc Function" on page 4-45
"Getting Help in the Command Window — the help Function" on page 4-46

About Help Functions

There are several help functions that provide forms of help different than the Help browser documentation, or provide alternative ways to access the Help browser information.

Summary Table of Help Functions

Function	Description
dbtype	Displays specified M-file with line numbers. If you want to see only the input and output arguments for a function, use dbtype function 1, which displays the first line of the M-file.
demo	Displays the **Demos** pane in the Help browser, from which you can access demonstrations for the products you have installed. With an argument, runs the specified demo.
doc	Displays in the Help browser, the reference page for the specified function, block, or property. Usually more extensive than results for the help function, the reference page provides syntax, a description, examples, illustrations, and links to related functions.
docopt	On UNIX systems, specifies Web browser information, used when displaying Internet Web pages.
docsearch	Run the Help browser search feature for the specified term.
help	Displays M-file help (a description and syntax) in the Command Window for the specified function. For MDL-files, displays a description of the model.

Function	Description
`helpbrowser`	Opens the Help browser, the MATLAB interface for accessing documentation.
`helpdesk`	Opens the Help browser. In previous releases, `helpdesk` displayed the Help Desk, which was the precursor to the Help browser. This function will be removed in a future release.
`helpwin`	Displays in the Help browser a list of all functions, and provides access to M-file help for the functions.
`lookfor`	Displays in the Command Window a list and brief description of all functions whose brief description includes the specified keyword.
`web`	Opens the specified URL in the specified browser. Use `web` in your own M-files to display HTML documentation you create for your work.
`whatsnew`	Displays the Release Notes in the Help browser.

View Function Reference Pages — the doc Function

To view the reference page for a function, block, or property in the Help browser, use `doc`. For example, type

```
doc format
```

to view the reference page for the `format` function.

Overloaded Functions with the doc Function

When a function name is used in multiple products, it is said to be an overloaded function. The `doc` function displays the reference page for the first function on the MATLAB search path having that name, and displays a hyperlinked list of the overloaded functions in the Command Window.

For example, using the default search path

```
doc set
```

displays the reference page for the MATLAB `set` function in the Help browser. The Command Window displays a hyperlinked list of the `set` functions located in other directories, such as

```
database/set
```

which is the set function for Database Toolbox. Click a link to go to that set reference page.

To directly get the reference page for an overloaded function, specify the name of the directory containing the function you want the reference page for, followed by the function name. For example, to display the reference page for the set function in Database Toolbox, type

```
doc database/set
```

Some products have more than one function with the same name. For example, MATLAB includes a built-in get function in the graphics directory and a get function in the MATLAB serial directory (for serial port functions). Type

```
doc get
```

The reference page for the MATLAB graphics built-in get function appears, and the Command Window lists overloaded functions in other products. But the list does not include any overloaded functions in the same product. Therefore, get in the MATLAB serial directory is not listed as an overloaded function. Type

```
doc ('get (serial)')
```

to display the reference page for the get function located in the MATLAB serial directory.

Getting Help in the Command Window — the help Function

To quickly view a brief description and syntax for a function in the Command Window, use the help function. For example, typing

```
help bar
```

displays a description and syntax for the bar function in the Command Window. This is called the M-file help. For other arguments you can supply, see the reference page for help.

> **Note** M-file help displayed in the Command Window uses all uppercase characters for the function and variable names to distinguish them from the rest of the text. When typing function names, however, use lowercase characters. Some functions for interfacing to Java do use mixed case; the M-file help accurately reflects that, and you should use mixed case when typing them.

If you need more information than the `help` function provides, use the `doc` function, which displays the reference page in the Help browser. It can include color, images, links, and more extensive examples than the M-file help. For example, typing

```
doc bar
```

displays the reference page for the `bar` function in the Help browser.

Overloaded Functions with the help Function

When a function name is used in multiple products, it is said to be an overloaded function. The `help` function displays M-file help for the first function on the MATLAB search path having that name, and displays a hyperlinked list of the overloaded functions at the end.

For example, using the default search path

```
help set
```

displays M-file help for the MATLAB `set` function, and displays a hyperlinked list of the `set` functions residing in other directories, such as

```
database/set
```

which is the `set` function for Database Toolbox. Click a link to display the M-file help for that `set` function.

To directly get help for an overloaded function, specify the name of the directory containing the function you want help for, followed by the function name. For example, to get help for the `set` function in Database Toolbox, type

```
help database/set
```

Creating M-File Help for Your Own M-Files

You can create M-file help for your own M-files and access it using the `help` command. See the `help` reference page for details.

Help in the Current Directory Browser

The Help Report and the Contents Report provide other ways of looking at and managing help for M-files — see "Directory Reports in Current Directory Browser" on page 7-2.

You can also see the help for an M-file in the Current Directory browser if you have its preference for **Show M, MDL, and MAT file contents** selected.

Help for Model Files

Use the `help` function with an MDL filename to display the complete description for the model file. For example, run

```
help f14_dap.mdl
```

and MATLAB displays the description of the Simulink F-14 Digital Autopilot High Angle of Attack Mode, as defined in the **Model > Properties > Description**.

```
Multirate digital pitch loop control for F-14 control design
demonstration.
```

If Simulink is installed, you do not need to include the `.mdl` extension.

You can see the same description in the Current Directory browser if you have its preference for **Show M, MDL, and MAT file contents** selected.

Getting Pop-Up Help for Functions

MATLAB provides pop-up help for MATLAB functions from both the Editor/Debugger and the Command Window. Pop-up help, also referred to as "help on selection," enables you to view documentation for a function without requiring you to interrupt the flow of your work to open the Help browser.

To access pop-up help, follow these steps:

1 Select a function or click the pointer in a function for which you want information.

2 Press **F1** or right-click and select **Help on Selection**.

Documentation for the function opens in a small pop-up window, similar to that shown in the figure that follows.

3 Move or resize the pop-up window, if desired.

4 Press the **F1** key to toggle focus between the pop-up window and Command Window or the Editor/Debugger.

When you type in the Command Window or the Editor/Debugger, the pop-up window remains open.

5 With the pop-up window active, press the Escape (**Esc**) key to close the pop-up window.

If you want to get documentation on a different function while the pop-up window is open, there is no need to close it. Repeat steps 1 and 2—the pop-up window refreshes with documentation for the new function.

If you want to open the Help browser from the pop-up window, click the **Open Help Browser** link at the bottom of the pop-up window. The Help browser opens and the pop-up window closes.
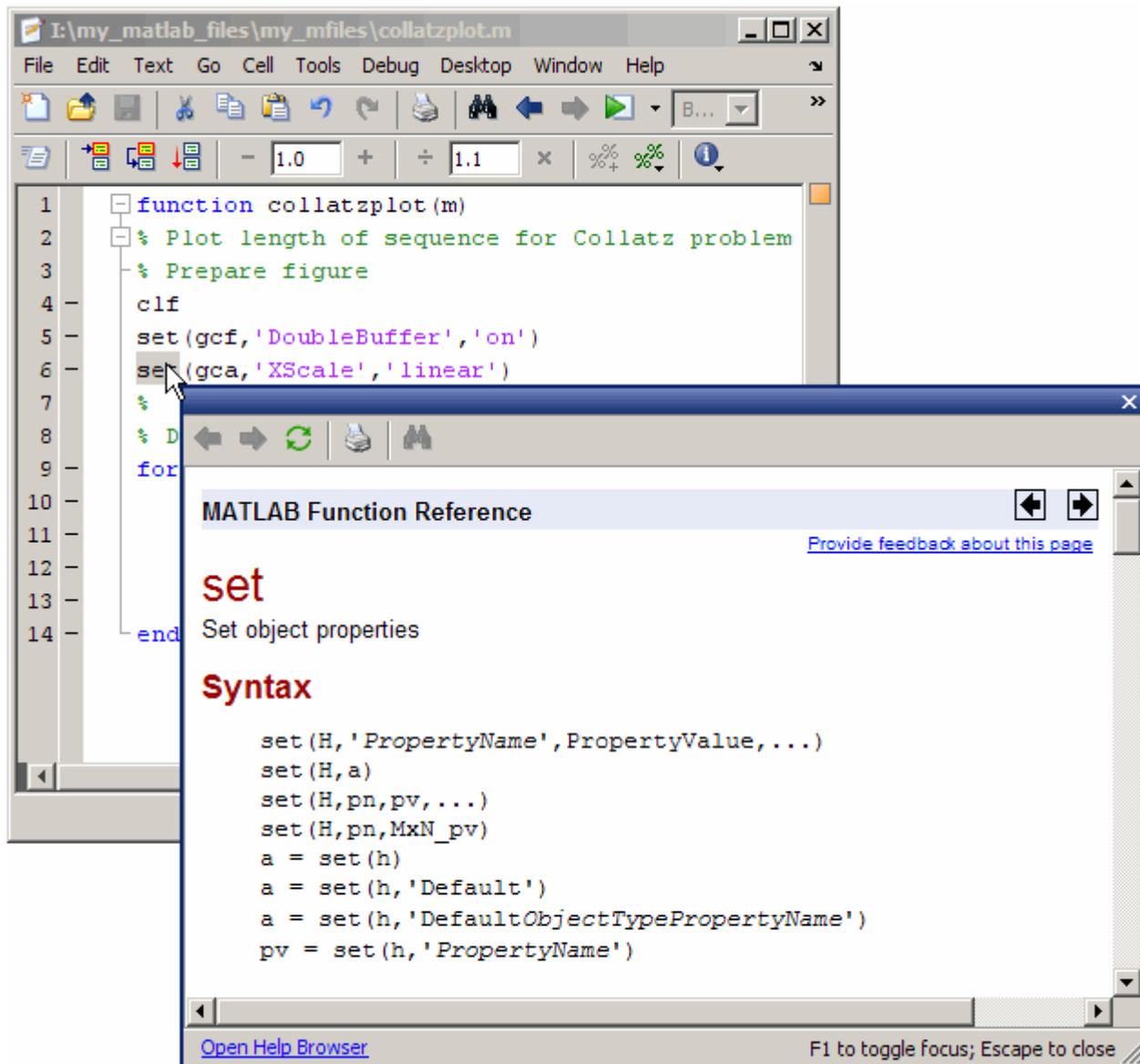
Other Forms of Help

In this section...
"Documentation for Other Products" on page 4-51
"Product-Specific Help Features" on page 4-51
"User-Contributed M-Files" on page 4-51
"Technical Support" on page 4-52
"Newsgroup for MathWorks Products" on page 4-52
"Other Resources for MATLAB Information" on page 4-53
"Version and License Information" on page 4-53
"Provide Feedback" on page 4-54

Documentation for Other Products

The Help browser provides access to documentation for all products installed on your system. To view the online documentation for all MathWorks products, use the MathWorks Web site at `http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml`.

Product-Specific Help Features

In addition to the Help browser and help functions, some products and tools allow other methods for getting help. You will encounter some methods in the course of using a product, such as entries in the **Help** menu, **Help** buttons in dialog boxes, and selecting **Help** from a context menu. These methods all display context-sensitive help. Other methods for getting help, such as pressing the **F1** key, are described in the documentation for the product or tool that uses the method.

User-Contributed M-Files

You can download M-files contributed by users and developers of MATLAB, Simulink, and related products from MATLAB Central. Before you write an M-file yourself, especially if it seems to be a generic utility, check the list of contributed files to see if someone has already written it. These files are freely contributed and can be used without charge by anyone

who downloads them. To view the files available to download, go to
the MATLAB Central File Exchange page on the MathWorks Web site,
`http://www.mathworks.com/matlabcentral/fileexchange/index.jsp`.
You can access this from any desktop component via **Help > Web Resources**.

If you write M-files that you think would be of use to others, consider
submitting them to the MATLAB Central File Exchange via the Web page.

Technical Support

Technical Support provides help for problems you have with MathWorks
products:

- Find specific Technical Support information using the Help browser
  Search feature. Run a search for a specified term. The end of the results
  list includes a link that runs the same search on the support database.
  This database, on the MathWorks Web site, provides the most up-to-date
  solutions, bug reports, and technical notes for questions posed by users.

- Select **Help > Web Resources > Support** to go to the Support Web
  page (`http://www.mathworks.com/support`). The page displays in your
  system's default Web browser. You can find out about other types of
  information such as third-party books, ask questions, make suggestions,
  view known bugs and workarounds, and report possible bugs.

- If you cannot access the Web site, e-mail Technical Support using the
  address `support@mathworks.com`. You must provide your license number
  to obtain support. It is helpful if you also provide your operating system
  and MATLAB version number. You can obtain the information by running
  the `ver` function or by selecting **Help > About**.

Newsgroup for MathWorks Products

The Usenet newsgroup for MATLAB and related products,
`comp.soft-sys.matlab`, (also known as `cssm`) is read by thousands of users
worldwide. Access the newsgroup to ask for or provide help or advice. You
can read and submit postings as well as view and search through a sizable
archive of postings using the MATLAB Central Newsgroup Access Web page
on the MathWorks Web site, `http://www.mathworks.com/matlabcentral`.
You can access this via **Help > Web Resources > MATLAB Newsgroup
Access** from any desktop component.

First-time users to the newsgroup should read the newsgroup FAQ, linked to from the MATLAB Central page. It is a good practice to try to solve your own problem using the documentation and Technical Support database before posting a question to the newsgroup. Be sure to post with a meaningful subject that briefly describes the nature of the issue.

Other Resources for MATLAB Information

Following are some additional resources for help with MATLAB and related products:

- Newsletters — The MathWorks publishes News and Notes twice a year, containing feature articles, technical notes, and product information for MATLAB users. More frequently, The MathWorks issues MATLAB Digest, an electronic bulletin consisting of technical notes, solutions, and timely announcements to the user community. Select **Help > Web Resources > MATLAB Newsletters** or see `http://www.mathworks.com/company/newsletters/`.

- Books — There are hundreds of MATLAB based books. For a list with descriptions, see `http://www.mathworks.com/support/books/`.

- Seminars and Training — The MathWorks regularly presents free seminars on special topics conducted in various locations. Webinars on special topics are presented via the Web. The MathWorks offers training classes for MATLAB and other products. For details, see `http://www.mathworks.com/company/events/`.

- `Mathtools.net` — This is a technical computing Web portal with links to many resources for MATLAB users. See `http://www.mathtools.net/`.

Version and License Information

If you need the version or license information for a product, select **About** from the **Help** menu for that product. The version is displayed in an About dialog box. If the product does not have a **Help** menu, use the `ver` function. To see the license number for MATLAB, type `license` in the Command Window. See also the `ver`, `version`, and `license` reference pages.

You can access information about your passcodes and licenses, as well get trial versions of products, using **Help > Web Resources > MathWorks Account**.

Provide Feedback

To report problems or provide comments or suggestions to The MathWorks about the documentation, help features, or products, use the **Provide feedback on this page** link on the top and bottom of every page in the Help browser.

5
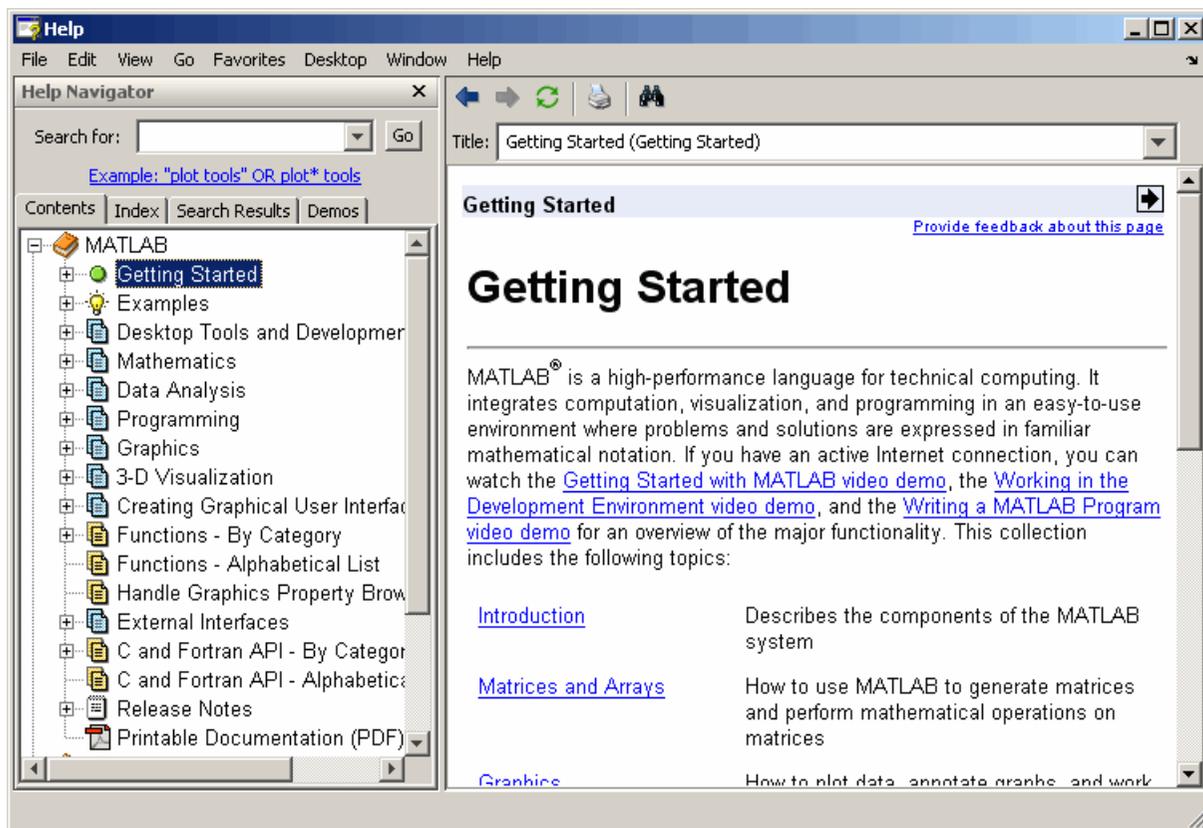
Workspace, Search Path, and File Operations

If you have an active Internet connection, you can watch the Workspace Browser video demo, the Array Editor video demo, and the Current Directory Browser video demo for an overview of the major functionality.

MATLAB Workspace (p. 5-2)	The workspace is the set of variables maintained in memory during a MATLAB session. Use the Workspace browser or equivalent functions to view the workspace.
Viewing and Editing Workspace Variables with the Array Editor (p. 5-12)	View and make changes to variables using the Array Editor.
Search Path (p. 5-23)	MATLAB uses a search path to find M-files and other MATLAB related files. View and change the path using the Set Path dialog box or equivalent functions.
File Management Operations (p. 5-35)	Search for, view, open, and make changes to MATLAB related directories and files, using the Current Directory browser or equivalent functions.

MATLAB Workspace

In this section...

About the MATLAB Workspace

The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces. For example, if you type

```
t = 0:pi/4:2*pi;
y = sin(t);
```

the workspace includes two variables, y and t, each having nine values.

You can perform workspace operations and related features using the Workspace browser. Equivalent functions are available and are documented with each feature of the Workspace browser. If you have an active Internet connection, you can watch the Workspace browser video demo for an overview of the major functionality:

Opening the Workspace Browser

To open the Workspace browser, select **Workspace** from the **Desktop** menu in the MATLAB desktop, or type `workspace` at the Command Window prompt.

The Workspace browser opens.
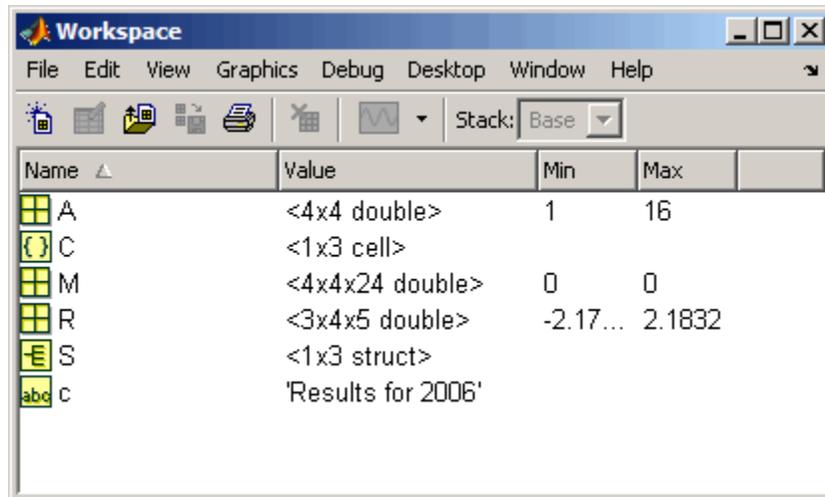
Viewing and Editing Values in the Current Workspace

The Workspace browser shows the name of each variable, its value, and the **Min** and **Max** calculations, which MATLAB computes using the `min` and `max` functions, and updates automatically. The icon for each variable denotes its class.

- You can display additional columns, including size (dimensions), size in bytes, class, and other common statistical calculations such as mode and standard deviation. To show or hide columns, select **View > Choose Columns** or right-click any column heading. To specify the size limit for calculations and how NaNs are considered, use "Preferences for the Workspace Browser" on page 5-10.

- To resize the columns of information, drag the column header borders. To reorder columns, drag a column header to a new position.

- You can select the column on which to sort as well as reverse the sort order of any column. Click a column heading to sort on that column. Click the column heading again to reverse the sort order in that column. For example, to sort on **Name**, click the column heading once. To change from ascending to descending, click the heading again. You cannot sort by the **Value** column in the Workspace browser. Alternatively, select **View > Sort By**

- You can edit variable values directly in the Workspace browser **Value** column. To edit a value, select the row to change in the **Value** column and type the new value.

This illustration shows the Workspace browser with all columns in view, sorted by the **Class** column, as indicated by the triangle in that column heading. Preferences are set to show statistical values only for arrays under 10,000 elements, so results for a, (1 x 20000), do not appear. The preference for treatment of NaNs is set to include them in calculations, as seen for n. With the preference set to ignore NaNs, values other than NaN would appear.

Function Alternative

Use who to list the current workspace variables. Use whos to list the variables and information about their size and class. For example:

```
>> who
```

```
Your variables are:

A  C  M  R  S  c

>> whos
 Name       Size              Bytes  Class      Attributes

  A         4x4                 128  double
  C         1x3                 348  cell
  M         4x4x24             3072  double
  R         3x4x5               480  double
  S         1x3                 826  struct
  c         1x16                 32  char
```

Use `exist` to see if the specified variable is in the workspace.

Saving the Current Workspace

The workspace is not maintained across MATLAB sessions. When you quit MATLAB, the workspace is cleared. You can save any or all of the variables in the current workspace to a MAT-file, which is a MATLAB specific binary file. You can then load the MAT-file at a later time during the current or another session to reuse the workspace variables. MAT-files use a `.mat` extension.

Note The `.mat` extension is also used by Microsoft Access.

Saving All Variables

To save all of the workspace variables using the Workspace browser,

1 From the **File** menu, select **Save Workspace As**, or click the Save button ▦ in the Workspace browser toolbar.

The **Save** dialog box opens.

2 Specify the location and **File name**. MATLAB automatically supplies the `.mat` extension.

3 Click **Save**.

The workspace variables are saved under the MAT-file name you specified.

You can also save the workspace variables from the desktop by selecting **Save Workspace As** from the **File** menu.

Saving Selected Variables

To save some but not all of the current workspace variables,

1 Select the variable in the Workspace browser. To select multiple variables, **Shift**+click or **Ctrl**+click.

2 Right-click and from the context menu, select **Save As**.

The **Save to MAT-File** dialog box opens.

3 Specify the location and **File name**. MATLAB automatically supplies the .mat extension.

4 Click **Save**.

The workspace variables are saved under the MAT-file name you specified.

To specify preferences for saving MAT-files, see "MAT-Files Preferences" on page 2-79.

Function Alternative

To save workspace variables, use the save function followed by the filename you want to save to. For example,

```
save('june10')
```

saves all current workspace variables to the file june10.mat.

If you don't specify a filename, the workspace is saved to matlab.mat in the current working directory. You can specify which variables to save, as well as control the format in which the data is stored, such as ASCII. For these and other forms of the function, see the reference page for save. For a related function, see genvarname. MATLAB provides additional functions for saving

information — see "Data Import and Export" in the MATLAB Programming documentation.

Loading a Saved Workspace and Importing Data

To load saved variables into the workspace,

1 Click the Import Data button 🖳 on the toolbar in the Workspace browser.

The **Open** dialog box opens.

2 Select the MAT-file you want to load and click **Open**.

The variables and their values, as stored in the MAT-file, are loaded into the current workspace. If any variables being loaded have the same names as variables in the current workspace, the values from the MAT-file replace the values in the current workspace. Any variables in the MAT-file that are not in the workspace are added to the workspace.

Function Alternative

Use `load` to open a saved workspace. For example,

```
load('june10')
```

loads all workspace variables from the file `june10.mat`.

Importing Data

MATLAB provides other methods and functions for loading information. You can use one of these methods, the Import Wizard, from the Workspace browser — select **Edit > Paste to workspace** or use **Ctrl+V** to import data to MATLAB using the Import Wizard. For more information on the Import Wizard and other methods for loading information, see the "Using the Import Wizard".

Viewing Variables in MAT-Files

Use the Current Directory browser to view the contents of a MAT-file without loading the file into MATLAB. For details, see "Finding Files and Content Within Files" on page 5-49.

Function Alternative. Use whos with the -file option.

Changing and Copying Variable Names

To rename a variable in the workspace, right-click the variable in the Workspace browser and select **Rename** from the context menu. Type the new variable name over the existing name and press **Enter** or **Return**.

To copy variable names to the clipboard, select the workspace variables and select **Edit > Copy**. You can then paste the names, for example, into the Command Window. Multiple variables are comma separated.

Deleting Workspace Variables

You can delete a variable, which removes it from the workspace. To delete a variable using the Workspace browser,

1 In the Workspace browser, select the variable, or **Shift**+click or **Ctrl**+click to select multiple variables. To select all variables, choose **Select All** from the **Edit** or context menus.

2 Press the **Delete** key on your keyboard or click the Delete button 🔳 on the Workspace browser toolbar.

3 A confirmation dialog box might appear. If it does, click **OK** to clear the variables.

   The confirmation dialog box appears if you selected that preference. For more information, see "Confirmation Dialogs Preferences" on page 2-81.

To delete all variables, select **Edit > Clear Workspace** from any desktop tool.

Function Alternative

Use the clear function to clear variables from the workspace. For example,

```
clear A M
```

clears the variables A and M from the workspace.

Viewing Base and Function Workspaces Using the Stack

When you run M-files, MATLAB assigns each function its own workspace, called the function workspace, which is separate from the MATLAB base workspace. To access the base and function workspaces when running or debugging M-files, use the **Stack** field in the Workspace browser. The **Stack** field is only available in debug mode and otherwise is grayed out. The **Stack** field is also accessible from the Array Editor and the Editor/Debugger. See "Finding Errors, Debugging, and Correcting M-Files" on page 6-84 for more information. See also the `dbstack` and `evalin` functions.

Creating Plots from the Workspace Browser

From the Workspace browser, you can generate a plot of a variable. To create a plot, click the Plot button ⊞ on the Workspace browser toolbar and select the plot type. The plot appears in a figure window. The button itself changes to reflect the currently selected style of plot, for example bar or stem.

This feature is only available for variables whose data types can be plotted, such as numeric. Open the variable in the Array Editor for additional plotting options.

In addition, you can right-click the variable you want to plot. From the context menu, choose the type of plot you want to create.

You can also **Shift**+click or **Ctrl**+click to select multiple variables to plot together. When one of the variables is named `time`, `t`, or `T`, MATLAB assumes it is the independent variable.

For more information about creating graphs in MATLAB, see the *MATLAB Graphics* documentation.

Opening Variables and Objects for Viewing and Editing

In the Workspace browser, double-click a variable and it opens in the Array Editor, where you can view and edit the contents of the variable. See "Viewing and Editing Workspace Variables with the Array Editor" on page 5-12 for more information about opening arrays.

Some toolboxes allow you to double-click an object in the Workspace browser to open a viewer or other tool appropriate for that object. For details, see the toolbox documentation for that object type.

Preferences for the Workspace Browser

The Workspace browser displays statistical calculations for variables. Use preferences to restrict the size of arrays on which calculations are performed and to specify if NaNs are included or ignored in calculations. Select **File > Preferences** to open the dialog box. Make changes and click **OK**.
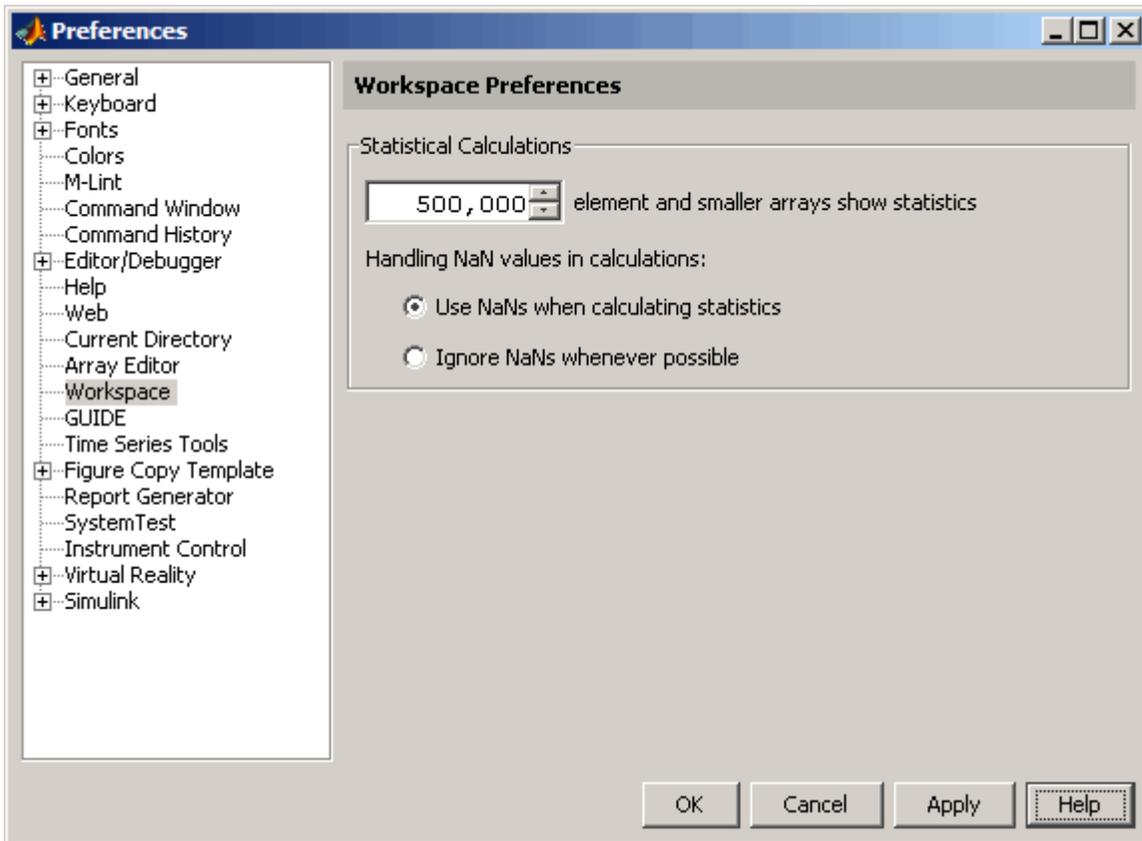
Specify Maximum Array Size on Which to Compute Statistics

If you show statistical columns in the Workspace browser, and if you work with very large arrays, you might experience performance issues when the data changes as MATLAB updates the statistical results. In that event, show only the columns of interest to you and hide those you do not need.

Another step you can take is specify via a preference that the Workspace browser *not* perform statistical calculations on the largest arrays. Use the arrows to change the value of the maximum array size for which you want the Workspace browser to perform statistical calculations. The default value is 500,000 elements. Any variable exceeding that size reports <Too many elements> instead of statistical results.

Handling NaN Values in Calculations

If your data includes NaNs, you can specify that the statistical calculations consider the NaNs, or ignore the NaNs. For example, if a variable includes a NaN and the preference is set to **Use NaNs when calculating statistics**, the values for **Min**, **Max**, **Var** and some others will appear as NaN, although **Mode**, for example, shows a numeric result. With the preference set to **Ignore NaNs whenever possible**, numeric results appear for most of the statistical columns including **Min** and **Max**, however, **Var** is still reported as NaN.

For more information about statistical values in the Workspace browser, see "Viewing and Editing Values in the Current Workspace" on page 5-3.

Viewing and Editing Workspace Variables with the Array Editor

In this section...
"About the Array Editor" on page 5-12
"Opening the Array Editor" on page 5-12
"Viewing and Editing Cell Arrays, Structures, and Multidimensional Arrays" on page 5-14
"Navigating and Editing Shortcut Keys for the Array Editor" on page 5-16
"Changing Array Size, Content, and Format of Elements in the Array Editor" on page 5-17
"Cut, Copy, Paste, and Clear Contents in the Array Editor" on page 5-18
"Insert and Delete in the Array Editor" on page 5-21
"Undo and Redo in the Array Editor" on page 5-21
"Exchanging Data with the Command Window" on page 5-21
"Exchanging Data with Excel" on page 5-21
"Creating Graphs and Variables from the Current Selection" on page 5-21
"Preferences for the Array Editor" on page 5-22

About the Array Editor

You use the Array Editor to view and edit a visual representation of one or two-dimensional numeric arrays, strings, cell arrays of strings, and structures. You can also view the contents of multidimensional arrays. If you have an active Internet connection, watch the Array Editor video demo for an overview of the major functionality.

Opening the Array Editor

To open the Array Editor from the Workspace browser,

1 In the Workspace browser, select the variable you want to open. **Shift**+click or **Ctrl**+click to select multiple variables, or use **Ctrl+A** to select all variables to open.

2 Click the Open Selection button ☑ on the toolbar. For one variable, you can also open it by double-clicking it.

The Array Editor opens, displaying the values for the selected variable.
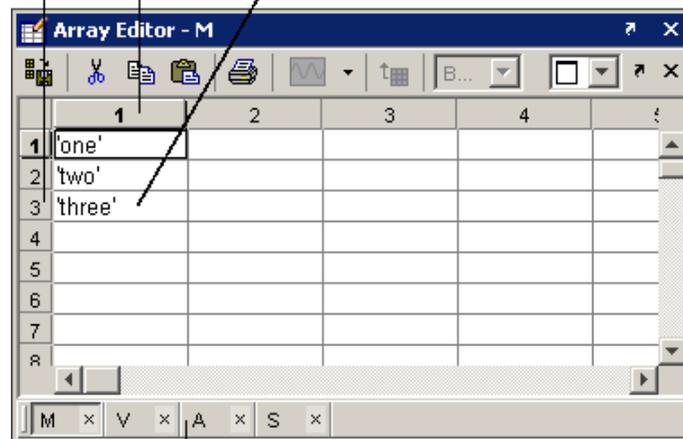
Repeat the steps to open additional variables in the Array Editor. Access each variable via its tab at the bottom of the window, or use the **Window** menu.

Changes you make to variables via the Command Window or other operations automatically update the information for those variables in the Array Editor.

Note The maximum array size that you can open in the Array Editor is not limited by MATLAB, but is based on your operating system or the amount of physical memory installed on your system.

Function Alternatives

To open a variable in the Array Editor, use `openvar` with the name of the variable you want to open as the argument. For example, type

```
openvar('M')
```

MATLAB opens `M` in the Array Editor.

To see the contents of a variable in the workspace, just type the variable name at the Command Window prompt. For example, type

```
M
```

and MATLAB returns

```
M =
'one'
'two'
'three'
```

Viewing and Editing Cell Arrays, Structures, and Multidimensional Arrays

Cell Arrays and Structures in the Array Editor

You can view and edit the content of cell arrays and structures in the Array Editor.

In the Array Editor, double-click an element of a structure to open it as its own Array Editor document. You can then view and edit the contents of that element.

Similarly, double-click a cell in a cell array to view and edit its contents. The following illustration shows an 8-by-1 cell array, M, and the contents of M{4,1}.

Double-click cell in cell array or element of structure to view its contents.

In cell array M, double-click cell {4,1} to view its contents.   Contents of M{4,1}.
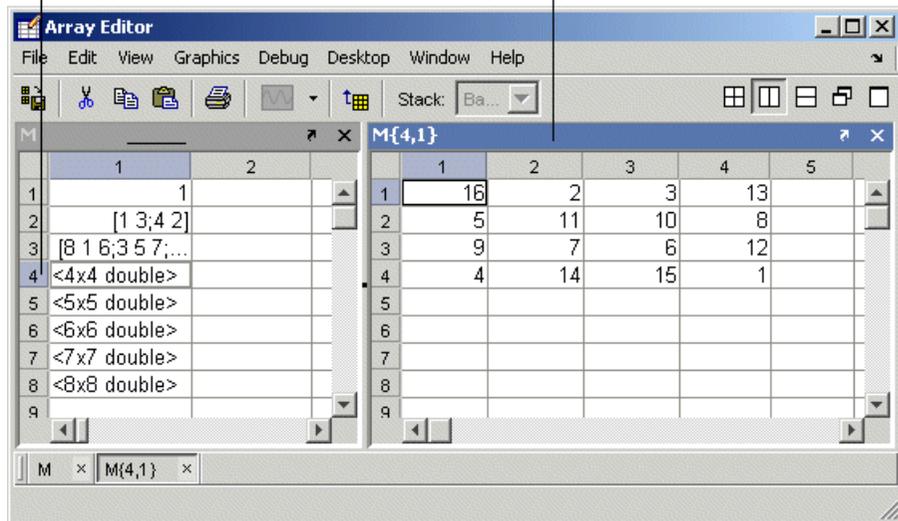
Multidimensional Arrays in the Array Editor

You can view the contents of multidimensional arrays in the Array Editor. When you open a multidimensional array in the Array Editor, it does not have usual grid structure, because multidimensional arrays do not fit that format. You cannot double-click an element in a multidimensional array to edit it. The following illustration shows R = rand(1,2,3).

You can view but cannot edit the contents of a
multidimensional array in the Array Editor.
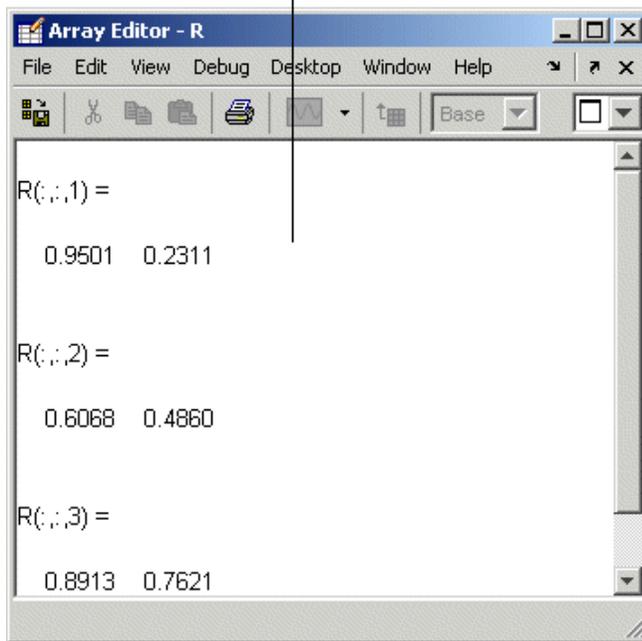
Navigating and Editing Shortcut Keys for the Array Editor

Use the following shortcut keys (sometimes called hot keys) to move among
elements in the Array Editor. Navigating in the Array Editor is much like
navigating in Microsoft Excel.

Key	Result
Enter	Commit any changes to the element and move to next element, where next element is specified using "Preferences for the Array Editor" on page 5-22 (default is down)

Key	Result
Tab	Move right
	Within a selection, also moves from the last column to the first column in the next row
Shift+Enter or **Shift+Tab**	Move in opposite direction of **Enter** or **Tab**
Page Up	Move up m rows, where m is the number of visible rows
Page Down	Move down m rows, where m is the number of visible rows
Home	Move to column 1
Ctrl+Home	Move to row 1, column 1
Shift+Home	Select to column 1
End	Move to last column in current row
F2 (**Ctrl+U** on Macintosh)	Edit current element, positioning cursor at the end of the element

Changing Array Size, Content, and Format of Elements in the Array Editor

To increase the size of an array, scroll to the desired location in the array and enter a value. The array will automatically expand to accommodate the new value. Empty cells are filled with zeros, if numeric, or empty arrays, if a cell array. To decrease the size of an array, select the rows or columns that you want to remove by clicking in the row or column header to select the entire row, right-clicking, and selecting **Delete**.

To change the value of an element in the Array Editor, click in that element and type a new value. Press **Enter** or **Return**, or click in another element to make the change take effect. You can specify where the cursor moves to after you press **Enter** — see "Preferences for the Array Editor" on page 5-22.

If you want to change the display format for the Array Editor, select the **View** menu and choose a format. To change the default format for future use, use

the **Preferences** dialog. For more information, see "Preferences for the Array Editor" on page 5-22.

If you opened an existing MAT-file and made changes to it using the Array Editor, save that MAT-file if you want the changes to be saved. For instructions, see "Saving the Current Workspace" on page 5-5.

Cut, Copy, Paste, and Clear Contents in the Array Editor

You can cut or copy selected elements, rows, and columns in an array and paste them to another position in that or another open array. To select a column or row, click in the row or column heading (the element that shows the row or column number). **Shift**+click to choose contiguous elements, rows, or columns in the array, or **Ctrl+A** to select all elements. For the cut, copy, and paste operations, use the **Edit** menu, the context menu, or the toolbar buttons. You can undo the last operation you performed in the Array Editor.

When you cut elements, the value of each element you cut becomes 0 if numeric or [ ] if a cell array. After cutting, select the elements whose value you want to replace with the cut elements and then choose **Paste**. If the shape of the elements you cut differs from the shape of the elements into which you are pasting, the Array Editor pastes all the elements, either by expanding the selection to be pasted into, or by expanding the array size to allow all the elements to be pasted. Pasting copied elements is the same as pasting cut elements, but the elements copied maintain their value rather than becoming 0.

Select elements, rows, or columns and then select **Edit > Clear Contents**. The value of the selected elements becomes 0. It differs from a performing a **Cut** because the data from the selected elements does not move to the clipboard; any clipboard content is unaffected by **Clear Contents**.

Example Copying and Pasting Array Elements

In this example, two elements are copied but the selected area for pasting is only one element, so the Array Editor expands the selected area for pasting.

Two elements are selected and copied.

One element is selected as the paste area.

The Array Editor pastes all of the copied elements.
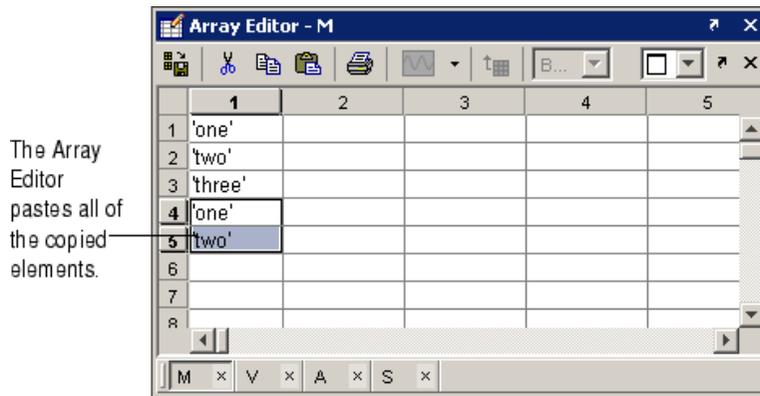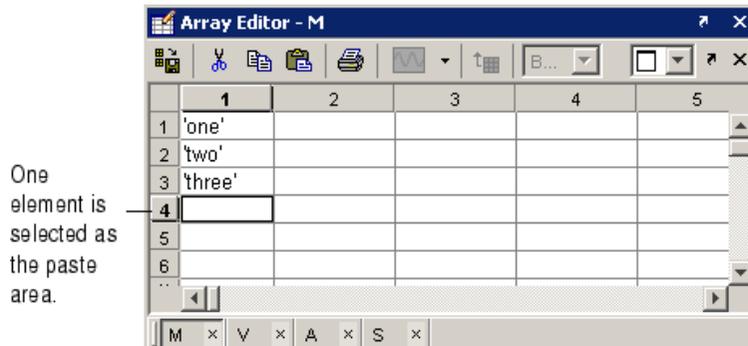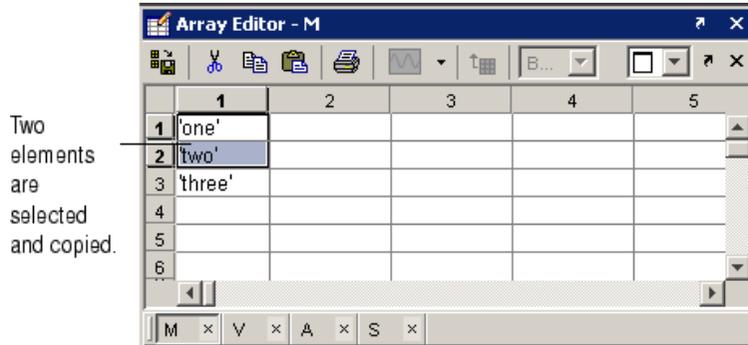
Example Cutting and Pasting Array Elements

In this example, the area selected for pasting requires the Array Editor to expand the array size in order for all cut elements to be pasted.

Two elements are selected and cut.

The values of the elements are cleared.

One element is selected as the paste area.

The Array Editor adds rows so that both cut elements can be pasted.
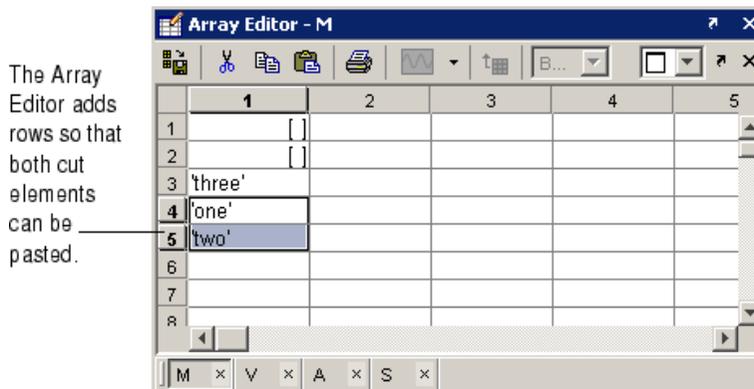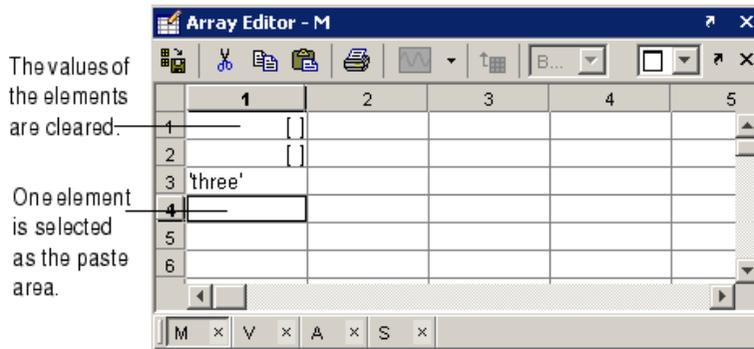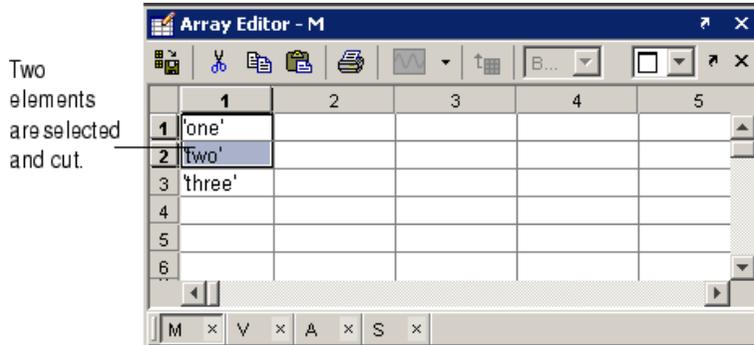
Insert and Delete in the Array Editor

You can insert and delete elements, rows, and columns in the Array Editor. When you select **Edit > Insert**, or **Edit > Delete**, a dialog box appears in which you specify cells, rows, or columns, and for cells, the direction for shifting.

Undo and Redo in the Array Editor

You can undo the last action you performed in the Array Editor, or redo a change after choosing undo. Select **Edit > Undo** or **Edit > Redo**. The actions supported are a change to a value you make by editing it in the Array Editor, cutting, pasting, inserting, deleting, clearing contents, and pasting Excel data.

Exchanging Data with the Command Window

You can copy data from the Array Editor and paste it into the Command Window. You can also copy a value from the Command Window and paste it into an element in the Array Editor. Be sure the data types are compatible. For example, you cannot paste text from the Command Window into a numeric array in the Array Editor.

Exchanging Data with Excel

You can cut or copy cells from Microsoft Excel and paste them into the Array Editor—use **Edit > Paste from Excel**. You can also cut or copy elements from the Array Editor and paste them into Excel.

Be sure the data types are compatible. For example, you cannot paste text from Excel into a numeric array in the Array Editor.

Creating Graphs and Variables from the Current Selection

You can create graphs and new variables from the Array Editor. To create a graph, select a cell, row, or column, and in the right-click context menu, choose the graph type. MATLAB presents allowable options for the selected data. In some cases, MATLAB makes assumptions, such as using `cell2mat` to convert selected cell array data, which cannot be plotted directly.

To create a new variable, select a cell, row, or column in the Array Editor, right-click, and from the context menu, select **Create Variable from Selection**.

Preferences for the Array Editor

To set preferences for the Array Editor, select **File > Preferences**. The Preferences dialog box opens showing **Array Editor Preferences**.

Format

Specify the default array output format of numeric values displayed in the Array Editor. This affects only how numbers are displayed, not how MATLAB computes or saves them. For more information, see the reference page for `format`.

Editing

You can specify where the cursor moves to after you type in an element and press **Enter**:

- If you want the cursor to remain at the element where you just typed, clear the **Move selection after Enter** check box.

- If you want the cursor to move to another element, select the **Move selection after Enter** check box, and then use **Direction** to specify how you want the cursor to move. For example, if you want the cursor to move right one element after you press **Enter**, select **Right**.

International Number Handling

You can specify how you want decimal numbers to be formatted when you cut or copy cells from the Array Editor and paste them into text files or other applications. The **Decimal separator to use when copying** edit field is by default "." (period). If you are working in or providing data to a locale that uses a different character to delimit decimals, type that character in this preference and click **OK** or **Apply**.

Search Path

In this section...

About the Search Path

MATLAB uses a *search path* to find M-files and other MATLAB related files, which are organized in directories on your file system. By default, the files supplied with MATLAB and MathWorks products are included in the search path. This default search path includes many of the directories and files under *matlabroot*/toolbox, where matlabroot is the directory in which MATLAB was installed, as returned by running the matlabroot function.

Any file you want to run or debug in MATLAB must reside in a directory that is on the search path, or in the current directory. If you create any MATLAB related files, add the directories containing the files to the MATLAB search path.

If you try to run or debug a file that is in a directory not on the path, the action fails. When this occurs in the Command Window, MATLAB errors. In the Editor/Debugger, a dialog box prompts you to

- **Change Directory** — Makes the directory containing the file you want to run become the current directory

- **Add to Path** — Adds the directory containing the file to the top of the search path

If there is already another file of the same name on the search path, the file you want to run is considered shadowed. You might need to perform some

action so that MATLAB runs the file you want—for more information, see "How the Search Path Determines Which Function to Use" on page 5-24.

The search path is also referred to as the *MATLAB path*. Directories included are considered to be *on the path*. When you include a directory in the search path, you *add it to the path*. Subdirectories must be explicitly added to the path; they are not on the path just because their parent directories are. Adding directories to the path is similar to performing an include or import in some other applications.

For instructions to view the search path and add directories to it, see "Viewing and Setting the Search Path" on page 5-26, including "Caution Against Saving Files in matlabroot/toolbox" on page 5-31.

How the Search Path Determines Which Function to Use

The order of directories on the path is relevant. MATLAB looks for a named element, for example, foo, as described here. If you enter foo at the MATLAB prompt, MATLAB performs the following actions:

1 Looks for foo as a variable.

2 Looks in the current directory for a file named foo.m.

3 Searches the directories on the MATLAB search path, in order, for foo as a built-in function, followed by foo.m which is not built-in.

If there is more than one function with the same name, the order of directories on the path determines which of those functions MATLAB uses. When MATLAB looks for that function, it uses the first one found in the search path:

• To use a function with the same name that is located in a directory further down on the search path, called a *shadowed* function, make its location the current directory. For M-file scripts, you can use run with the full pathname for the M-file. For example, use run d:/mymfiles/foo.m to ensure that version of foo runs. Another option is to move the directory containing the shadowed function to a position in the search path ahead of the directory containing the file of the same name. For example, add it to the top of the search path.

- If you are not sure of the function MATLAB is using, run `which` for a specified function and MATLAB returns the full path to the function.

Although the actual search path rules are more complicated because of the restricted scope of private functions, subfunctions, object-oriented functions, P-files, and MAT-files, this simplified perspective is accurate for the ordinary M-files you usually work with. For more information, see "Determining Which Function Is Called" in the MATLAB Programming documentation.

How MATLAB Finds the Search Path, pathdef.m

The search path is stored in the file `pathdef.m`, which by default, is located in *matlabroot*`/toolbox/local`. You can store it in the MATLAB startup directory, and modify it for the current session or for all future sessions.

When MATLAB starts, it looks for a `pathdef.m` file in its startup directory. If none is found, it uses `pathdef.m` in `matlabroot/toolbox/local`. MATLAB modifies the path based on any path statements in a `startup.m` file. During a session, you can save changes to the path using the **Set Path** dialog box or the `savepath` function, and MATLAB uses the path you saved to for the remainder of the session.

If MATLAB finds a `pathdef.m` in the current directory, it uses that version instead. To avoid problems, do not maintain a `pathdef.m` file in a directory other than the MATLAB startup directory or *matlabroot*`/toolbox/local`.

Viewing and Setting the Search Path

Use the **Set Path** dialog box to view and modify the MATLAB search path. Equivalent functions are documented for each feature of the **Set Path** dialog box. Select **Set Path** from the **File** menu, or type `pathtool` at the Command Window prompt. The **Set Path** dialog box opens.

Directories on the current
MATLAB search path.

Make changes
to the search
path.

Save changes
for use in future
MATLAB
sessions.

Use the changes for the current
session, but do not save the
changes for use in future MATLAB
sessions.

Replace current path with all
directories installed with
MATLAB and related products.
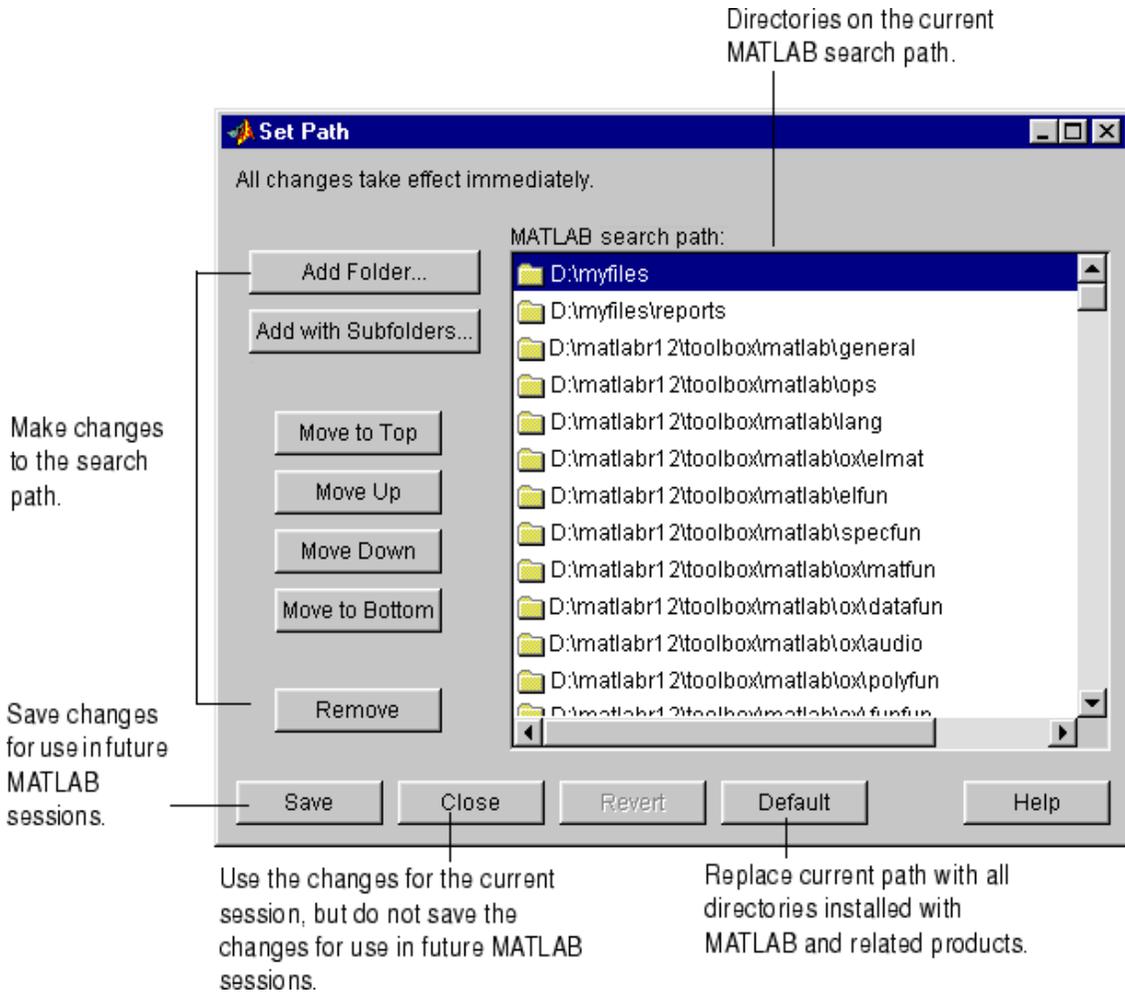
Use the **Set Path** dialog box for the following actions. Equivalent functions are listed as well:

See also

Viewing the Search Path
The **MATLAB search path** field in the **Set Path** dialog box lists all of the directories on the search path. The top of the list is the start of the search path, while the bottom of the list is the end.

Function Alternative. Use the `path` function to view the search path.

Adding Directories to the Search Path
Add directories to the search path when you want to run M-files in those directories.

To add directories to the MATLAB search path using the **Set Path** dialog box,

1 Click **Add Folder** or **Add with Subfolders**.

- If you want to add only the selected directory but do not want to add all of its subdirectories, click **Add Folder**.
- If you want to add the selected directory and all of its subdirectories, click **Add with Subfolders**.

The **Browse for Folder** dialog box opens.

2 In the Browse for Folder dialog box, use the view of your file system to select the directory to add, and then click **OK**.

The selected directory, and subdirectories if specified in step 1, are added to the top of the search path.

3 To use the newly modified search path in future sessions, click **Save**. For more information about saving the path, see "Saving Settings to the Path" on page 5-30.

4 Click **Close**. If you did not save the changes in the previous step, the directories you added remain on the search path until you end the current MATLAB session.

You cannot add method directories (directories that start with @) and private directories to the MATLAB search path.

Adding Directories to the Path from the Current Directory Browser.
In the Current Directory browser, select a directory, right-click, and select **Add to Path** from the context menu. Then select one of the submenus, for example, **Selected Folder and Subfolders**.

Function Alternative. To add directories to the top or the end of the search path, use addpath. The addpath function offers an option to get the path as a string and to concatenate multiple strings to form a new path.

You can include addpath statements in your startup M-file to automatically modify the path when MATLAB starts. For details, see "Modifying the Path in a startup.m File" on page 5-31.

Moving Directories Within the Search Path

The order of files on the search path is relevant — for more information, see "How the Search Path Determines Which Function to Use" on page 5-24.

To modify the order of directories within the search path,

1 Select the directory or directories you want to move.

2 Click one of the **Move** buttons, such as **Move to Top**. The order of the directories changes.

3 To use the newly modified search path in future sessions, click **Save**. For more information about saving the path, see "Saving Settings to the Path" on page 5-30.

4 Click **Close**. If you did not save the changes in the previous step, the new order of files on the search path remains in effect until you end the current MATLAB session.

Function Alternative. While there is not a specific function to move directories, you can edit the `pathdef.m` file with any text editor to change the order of the directories. Use caution when editing the file so that you do not make MATLAB and toolbox functions unusable.

Removing Directories from the Search Path

To remove directories from the MATLAB search path using the **Set Path** dialog box,

1 Select the directories to remove.

2 Click **Remove**. The directories are removed from the path.

3 To use the newly modified search path in future sessions, click **Save**. For more information about saving the path, see "Saving Settings to the Path" on page 5-30.

4 Click **Close**. If you did not save the changes in the previous step, the directories are removed from the search path until you end the current MATLAB session.

Function Alternative. To remove directories from the search path, use `rmpath`.

You can include `rmpath` statements in your `startup` M-file to automatically modify the path when MATLAB starts. For details see "Modifying the Path in a startup.m File" on page 5-31.

Restoring the Default Search Path

To restore the default search path, click **Default** in the **Set Path** dialog box. This changes the search path so that it includes only the directories installed with MATLAB and related products.

Reverting to the Previous Path

To restore the previous path, click **Revert** in the **Set Path** dialog box. This cancels any unsaved changes you have made in the **Set Path** dialog box.

Saving Settings to the Path

When you make changes to the search path, they remain in effect during the current MATLAB session. To keep the changes in effect for subsequent sessions, you need to save them. To save changes using the **Set Path** dialog box, click **Save**.

If you want to automatically use this search path in future sessions, save the path to your MATLAB startup directory, which saves `pathdef.m` to that location. You can save the changes to the default `pathdef.m` file, in *matlabroot*`/toolbox/local` if you have write permission for that directory but see the following caution. Alternatively, you can include `addpath` and `rmpath` statements in a `startup.m` file, which avoids some problems you might have with saving the path, for example, using the same path with both Windows and UNIX platforms. For more information, see "Using the Path in Future Sessions" on page 5-31.

Note When MATLAB starts up, it may provide in the Command Window a list of invalid directories. These directories were previously in `pathdef.m` but were subsequently deleted. The directories do not now appear on the search path listing in the **Set Path** dialog box. Click **Save** to overwrite the `pathdef` M-file, thereby eliminating future reporting of these nonexistent directories.

Caution Against Saving Files in matlabroot/toolbox. Save any M-files you create and any MathWorks supplied M-files that you edit in a directory that is not in the *matlabroot*/toolbox directory tree. If you keep your files in *matlabroot*/toolbox directories, they can be overwritten when you install a new version of MATLAB. Also note that locations of files in the *matlabroot*/toolbox directory tree are loaded and cached in memory at the beginning of each MATLAB session to improve performance. If you save files to *matlabroot*/toolbox directories using an external editor or add or remove in from these directories using file system operations, run rehash toolbox before you use the files in the current session. If you make changes to existing files in *matlabroot*/toolbox directories using an external editor, run clear functionname before you use the files in the current session. For more information, see rehash or "Toolbox Path Caching in MATLAB" on page 1-17.

Function Alternative. Use savepath to save the current path to pathdef.m. Locate pathdef.m in your MATLAB startup directory to automatically use it in future sessions. Consider using savepath in your finish.m file. To modify the default path upon startup, include addpath and rmpath functions in your startup.m file. For more information, see "Modifying the Path in a startup.m File" on page 5-31.

Using the Path in Future Sessions

There are three basic ways for MATLAB to automatically use a search path you specify, each with advantages and disadvantages:

- "Modifying the Path in a startup.m File" on page 5-31
- "Saving the Path in the MATLAB Startup Directory" on page 5-32
- "Saving the Path in matlabroot/toolbox/local" on page 5-32

For background information, see "How MATLAB Finds the Search Path, pathdef.m" on page 5-25.

Modifying the Path in a startup.m File

Put addpath and rmpath statements in a startup.m file, and include the startup file in MATLAB's startup directory. When MATLAB starts, it uses the search path defined in pathdef.m in *matlabroot*/toolbox/local and modifies it based on the commands in the startup.m file.

By maintaining an unaltered `pathdef.m` in *matlabroot*/`toolbox/local`, you avoid inadvertently removing directories supplied by The MathWorks from the path. This method continues working even when you update to a new version of MATLAB. If you run MATLAB on both Windows and UNIX platforms, this method works well — for example, for each platform, include separate `addpath` sections in the `startup.m` file, with each section preceded by an `ispc` or `isunix` statement.

One disadvantage of this method is that changes you make to the path using the **Set Path** dialog box are not incorporated in the `startup.m` file.

Saving the Path in the MATLAB Startup Directory

Copy `pathdef.m` from *matlabroot*/`toolbox/local` to the MATLAB startup directory. Make changes to the path using the **Set Path** dialog box, and with `addpath` and `rmpath` functions — choose whichever suits your needs. You can use this method if you do not have write access to *matlabroot*/`toolbox/local`.

There are some disadvantages to this method. You might inadvertently remove directories supplied by The MathWorks from the path. When you update to a new version of MATLAB, you cannot use the `pathdef.m` file in the startup directory, but must delete it and create a new version. If you run MATLAB on both Windows and UNIX platforms, you need to maintain a separate `pathdef.m` file for each.

Saving the Path in matlabroot/toolbox/local

If you have write access to *matlabroot*/`toolbox/local`, make and save changes to the path using the **Set Path** dialog box, and with `addpath` and `rmpath` functions — choose whichever suits your needs.

There are some disadvantages to this method. You cannot maintain this file when you update to a new version of MATLAB, but will need to use the new default `pathdef.m` and make changes to it. If you run MATLAB on both Windows and UNIX platforms, you need to maintain a separate `pathdef.m` file for each.

Recovering from Problems with the Search Path

If you get unexpected results that are related to the search path, you can try to correct the path file or restore the default path. You might experience path problems if you save the path on a Windows platform and then try to use the same `pathdef.m` file on a UNIX platform. Similarly, you might experience problems if you edit the `pathdef.m` file directly and make it invalid, or if the file becomes corrupt, renamed, or lost.

For example, if an error message similar to the following appears when you start MATLAB

```
Warning: MATLAB did not appear to successfully set the search
path...
```

it indicates a problem with the search path and you will not be able to use MATLAB successfully.

To recover from problems with the search path, try the following, in order, proceeding to the next step only if needed:

1 View the `pathdef.m` and `startup.m` files, looking for obvious problems. Make changes and save them. If path problems appear to be resolved, start MATLAB again to be sure the problem does not reappear. Depending on the problem, you might not be able to even view the `pathdef.m` file.

2 Use the default path for MathWorks products. In the **Set Path** dialog box, select **Default**, then **Save**, then **Close**. Depending on the problem, you might not be able to even open the dialog box.

3 Run `restoredefaultpath`. This sets the search path to include only installed products from The MathWorks. If that seems to have corrected the problem, run `savepath`. Start MATLAB again to be sure the problem does not reappear.

Depending on the problem, this might generate a message such as

```
The path may be bad. Please save your work (if desired), and quit.
```

If so, perform step 4.

4 Perform these steps after trying step 3.

a Run

```
restoredefaultpath; matlabrc
```

This might run for a few minutes. It sets the search path to include only installed products from The MathWorks and corrects path problems encountered during startup.

b If there is a `pathdef.m` in your startup directory for MATLAB, it caused the problem. So either remove the bad `pathdef.m` file or replace the with a good `pathdef.m` file, for example, one you can generate at this point with

```
savepath('path_to_your_startup_directory/pathdef.m')
```

c Start MATLAB again to be sure the problem does not reappear.

File Management Operations

In this section...

Note You generally cannot perform operations on files and directories for which you do not have proper permission. For example, you cannot copy a file to a read-only directory using the Current Directory browser, however, you can do so using `movefile` with the appropriate option.

About MATLAB File Operations

MATLAB file operations use the current directory and the MATLAB search path as reference points. Any file you want to run must either be in the current directory or on the search path. The key tools for performing file operations are

Current Directory Field

A quick way to view or change the current directory is by using the current directory field in the desktop toolbar.

D:\mymfiles

To change the current directory from this field, do one of the following:

- In the field, type the path for the new current directory.

- Click the down arrow to view a list of previous working directories, and select an item from the list to make that directory become the MATLAB current working directory. The directories are listed in order, with the most recently used at the top of the list. You can clear the list and set the number of directories saved in the list — see "Preferences for the Current Directory Browser" on page 5-54.

- Click the Browse for Folder button (**...**) to set a new current directory.

- Use the Go Up One Level button to move the current directory up one level.

The current directory field in the desktop also appears in the Current Directory browser, when the Current Directory browser is undocked. Consider it to be one tool with two different means of accessing it.

Current Directory Browser

To search for, view, open, find, and make changes to MATLAB related directories and files, use the MATLAB Current Directory browser. Most features of the Current Directory browser have equivalent functions that perform similar actions. If you have an active Internet connection, you can watch the Current Directory Browser video demo for an overview of the major functionality.

In addition to the features described here, the Current Directory browser includes tools to help you manage your M-files — see "Directory Reports in Current Directory Browser" on page 7-2.

To open the Current Directory browser, select **Desktop > Current Directory** from the MATLAB desktop, or type filebrowser at the Command Window prompt. The Current Directory browser opens.

Change the pathname in the edit box to view a directory and its contents. This field only appears when the Current Directory browser is undocked from the desktop.

Click the Find Files button to search for M-files and content within M-files.

For Directory Reports.

Double-click a file to open it in an appropriate tool.

With the **Show descriptions** preferences setting on, the help portion of the selected M-file or Simulink model, or contents of a MAT-file is displayed.
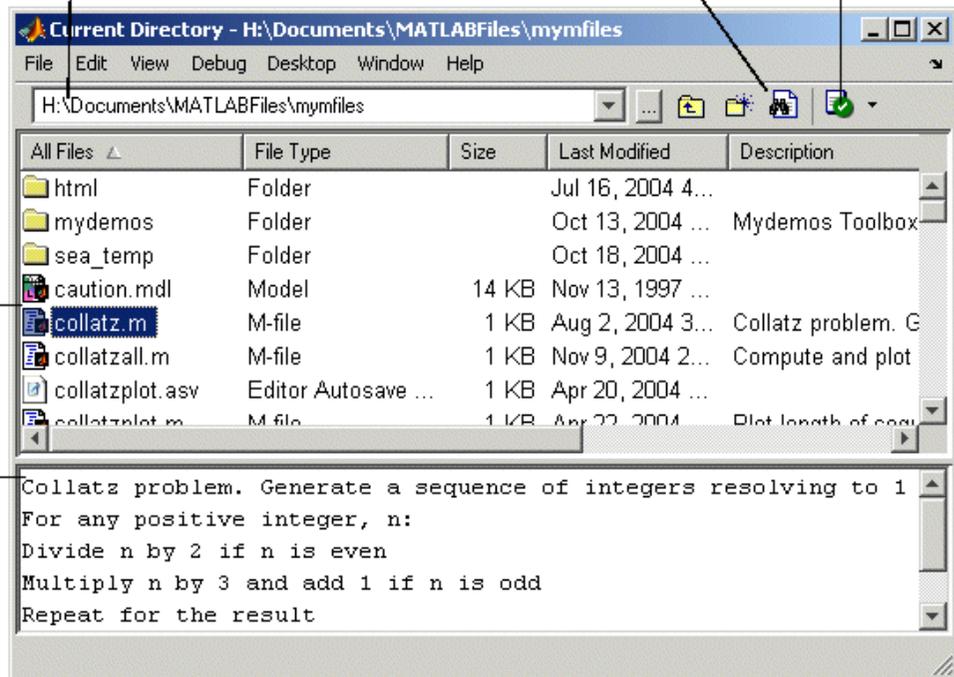
The main tasks you perform with the Current Directory browser are

- "Viewing and Making Changes to Directories" on page 5-38
- "Creating, Renaming, Copying, and Removing Directories and Files" on page 5-43
- "Opening and Running Files" on page 5-47
- "Finding Files and Content Within Files" on page 5-49
- "Accessing Source Control Features" on page 5-54
- Setting "Preferences for the Current Directory Browser" on page 5-54

Viewing and Making Changes to Directories

You can change the current directory, view its contents, add directories to the MATLAB search path, and change the way the Current Directory browser presents entries.

- "Changing the Current Working Directory and Viewing Its Contents" on page 5-38
- "Searching in the Current Directory Browser" on page 5-38
- "Changing the Display" on page 5-41
- "Adding Directories to the MATLAB Search Path" on page 5-43

Changing the Current Working Directory and Viewing Its Contents

To change the current directory, use the current directory field. The Current Directory browser lists the files and directories in the current directory.

To view the contents of a subdirectory, double-click it, or select the subdirectory and press **Enter** or **Return**.

To move up one level in the directory structure, press the backspace (**<-**) key.

Function Alternative. Use dir to view the contents of the current working directory or another specified directory. Use a return argument with dir to get a structure containing information including the names of the files in the directory and their last modified date and time.

Use what with no arguments to display the MATLAB related files in the current working directory. Use which to display the pathname for a specified function. Use exist to see if a directory or file exists. Use fileattrib to see or set file attributes, much like attrib in DOS or chmod in UNIX.

Searching in the Current Directory Browser

You can search the Current Directory browser for files and directories by typing directly in the window. As you type the Current Directory browser searches downward from the top of the window to find an entry that matches what you have typed. For example:

1 Set `C:\Climate` as your current directory.
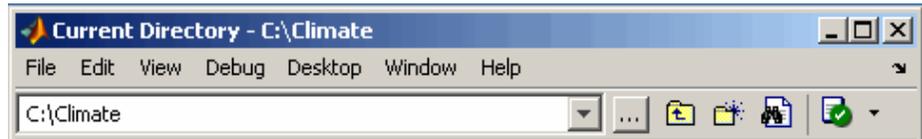
2 Assume that you want to search the Current Directory for the file named `standard.m`. Begin your search by positioning the cursor anywhere within the Current Directory browser.
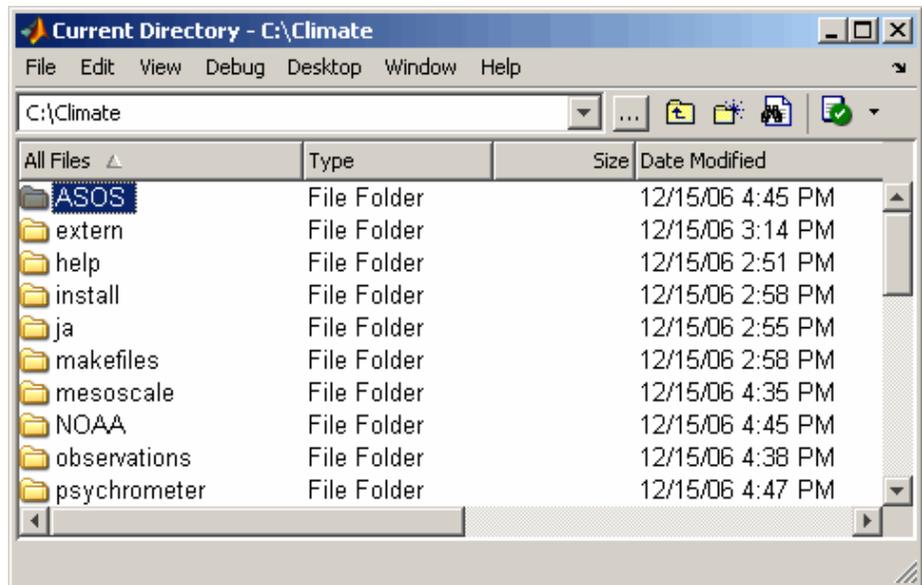
3 Type the letter `s`. The Current Directory browser searches to find the first entry beginning with the letter `s`. In this example it stops at the directory named `standalone`. Note as you begin typing that a small yellow-background `Search filenames for:` dialog box appears at the top of the Current Directory browser window. This dialog box keeps track of your search target as you type additional letters to narrow the focus of your search.

5-39
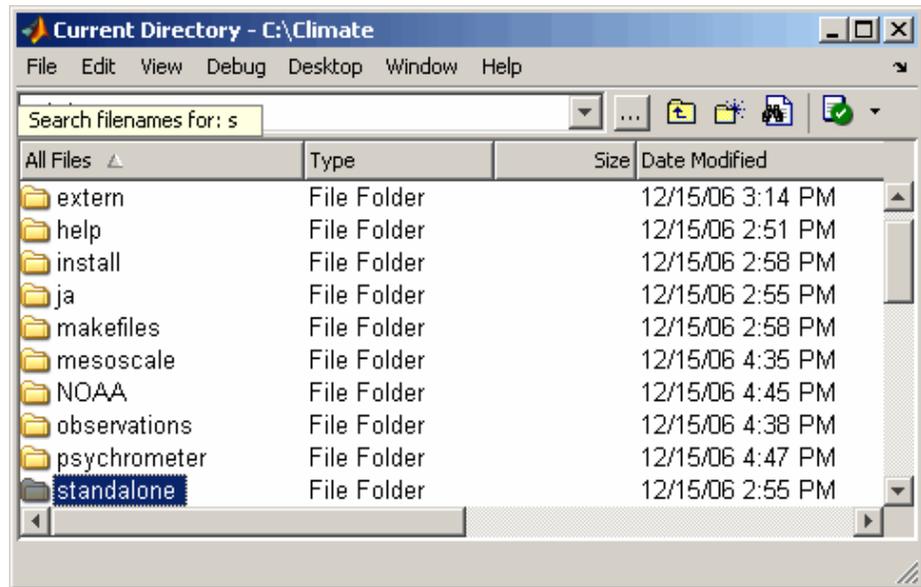
4 Since the `standalone` directory is not your intended search target, continue typing additional letters that identify your search target, eventually entering the letters `standar`.

5 When you have entered the letters `standar`, the search resumes, stopping this time at the `standard.m` file, your intended search target.
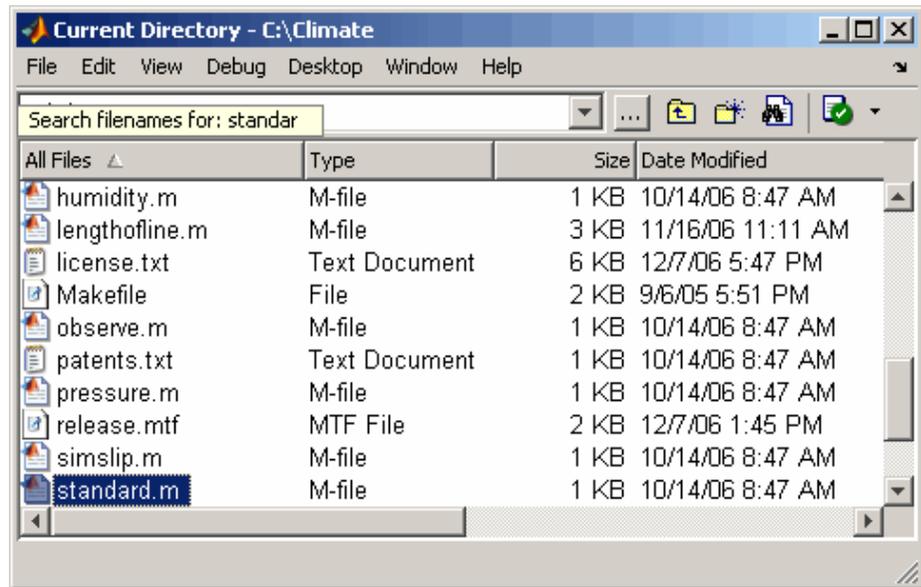
Changing the Display

Types of Files. To specify the types of files shown in the Current Directory browser, use the **View** menu. For example, you can show only M-files. If **All Files** is selected and you want to see specific file types, first clear the selection for **All Files** and then select the specific file types.

Columns. To show or hide columns, use preferences for the Current Directory browser. Select **File > Preferences > Current Directory** and select or clear the check boxes for **Browser display options**. For more information, see "Browser Display Options" on page 5-55.

You can sort the information shown in the Current Directory browser by column. Click the title of column on which you want to sort. The display is sorted, with the information in that column shown in ascending order, and an up arrow icon indicating the direction. Click a second time on the column title to sort the information in descending order.

Contents. In the Current Directory browser, select a file and then view information about the file in the Current Directory browser's lower pane. To view this, you must first select **File > Preferences > Current Directory** and under **Browser display options**, select the check box **Show M, MDL and MAT file contents**.

For an M-file, it shows the M-file help. For a Simulink model, it shows the complete description, allowing you to view information about a model without having to start Simulink. For a MAT-file, it displays the names of its variables along with their size, bytes, and class, allowing you to view the content of a MAT-file without loading it.

View variables in selected MAT-file without loading the file.
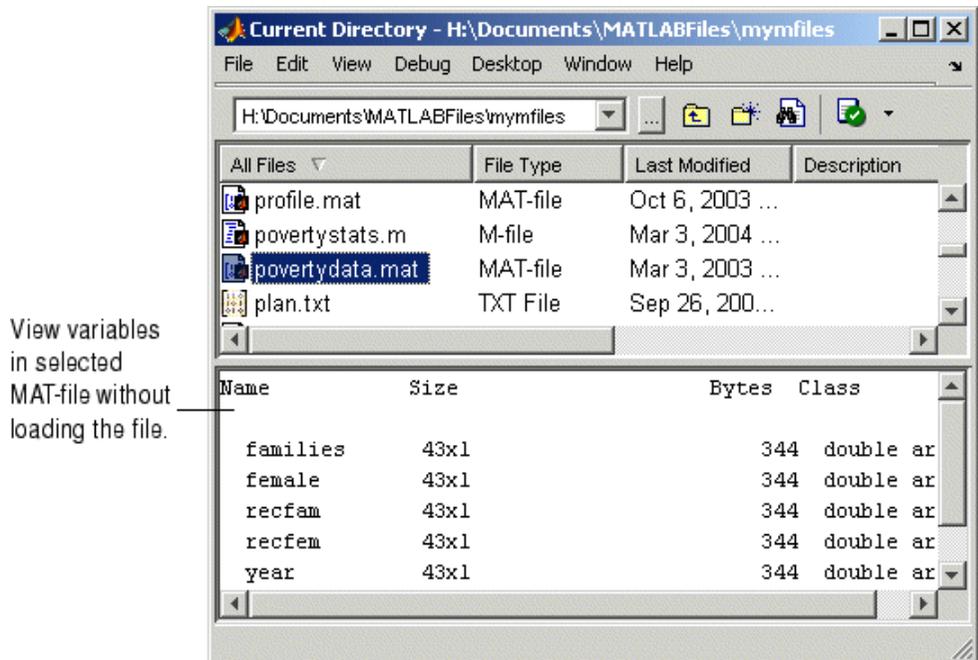
You can view more extensive help for the M-file selected in the Current Directory browser. From the context menu, select **View Help**. The reference page for that function appears in the Help browser.

Adding Directories to the MATLAB Search Path

From the Current Directory browser, you can add directories to the MATLAB search path. Right-click and from the context menu, select **Add to Path**. Then select one of the options:

• **Current Directory** — Adds the current directory to the path.

• **Selected Folders** — Adds the directories selected in the Current Directory browser to the path.

• **Selected Folder and Subfolders** — Adds the directory selected in the Current Directory browser to the path, and adds all of its subdirectories to the path.

Creating, Renaming, Copying, and Removing Directories and Files

• "General Notes" on page 5-43

• "Creating New Files" on page 5-44

• "Creating New Directories" on page 5-44

• "Renaming Files and Directories" on page 5-45

• "Cutting or Deleting Files and Directories" on page 5-45

• "Copying and Pasting Files and Directories" on page 5-46

General Notes

If you have write permission, you can create, copy, remove, and rename MATLAB related files and directories for the directory shown in the Current Directory browser. If you do not have write permission, you can still copy files and directories to another directory, or you can use equivalent functions, such as `movefile`.

To run functions whose arguments require the use of a pathname or filename, use the function form rather than the unquoted or command form of the syntax when the pathname or filename includes spaces. For example, the command form

```
delete my file.m
```

generates a warning and does not delete `my file.m`. Instead use the function form of the syntax:

```
delete('my file.m')
```

Creating New Files

To create a new file in the current directory,

1 Select **New** from the context menu or **File** menu and then select the type of file to create.

An icon for that file type, for example, an M-file icon , with the default name `Untitledn`, appears at the end of the list of files shown in the Current Directory browser.

2 Type over `Untitledn` with the name you want to give to the new file.

3 Press **Enter** or **Return**.

The file is added.

4 To enter the contents of the new M-file, open the file—see "Opening and Running Files" on page 5-47. If you created the file using the context menu, the new file opens in the Editor/Debugger with a template for writing an M-file function.

Function Alternative. Use the `edit` function to create a new M-file or other type of text file in the Editor/Debugger.

Creating New Directories

To create a new directory in the current directory,

1 Click the New Folder button in the Current Directory browser toolbar, or select **New > Folder** from the context menu.

An icon, with the default name `NewFoldern` appears at the end of the list of files shown in the Current Directory browser.

2 Type over `NewFoldern` with the name you want to give to the new directory.

3 Press the **Enter** or **Return** key.

The directory is added.

Function Alternative. To create a directory, use the `mkdir` function. For example,

```
mkdir newdir
```

creates the directory `newdir` within the current directory.

Renaming Files and Directories

To rename a file or directory, select the item, right-click, and select **Rename** from the context menu. Type over the existing name with the new name for the file or directory, and press **Enter** or **Return**. The file or directory is renamed.

Function Alternative. You can use `movefile` to rename a file or directory. For example,

```
movefile('myfile.m','projectresults.m')
```

renames `myfile.m` to `projectresults.m`.

Cutting or Deleting Files and Directories

To cut or delete files and directories,

1 Select the files and directories to remove. Use **Shift**+click or **Ctrl**+click to select multiple items.

2 Right-click and select **Cut** or **Delete** from the context menu.

The files and directories are removed.

Files and directories you delete from the Current Directory browser go to the Recycle Bin on Windows (or the Trash Can on Macintosh platforms). If you do not want the selected items to go to the Recycle Bin, press **Shift+Delete**. A confirmation dialog box displays before the items are deleted if you have set that option in your operating system. For example, on Windows, right-click

the Recycle Bin, select **Properties** from the context menu, and then, under the **Global** tab, select the check box to **Display delete confirmation dialog**.

Function Alternative. To delete a file, use the `delete` function. For example,

```
delete('d:/mymfiles/testfun.m')
```

deletes the file `testfun.m`. You can recover deleted files if you use the `recycle` function or the equivalent preference described in "Default Behavior of the Delete Function" on page 2-78.

To delete a directory and optionally its contents, use `rmdir`. For example,

```
rmdir('myfiles')
```

removes the directory `myfiles` from the current directory.

Copying and Pasting Files and Directories

Use the Current Directory browser, to copy (or cut) and paste files and directories:

1 Select the files or directories to copy. Use **Shift**+click or **Ctrl**+click to select multiple items. For a directory, the entire contents are copied, including all subdirectories and files.

2 Right-click and select **Copy** from the context menu.

3 Navigate to the file or directory where you want to paste the items you just copied.

4 Right-click and select **Paste** from the context menu.

You can also copy and paste files and directories to and from tools outside of MATLAB, such as Windows Explorer. You can use Current Directory browser menu items or keyboard shortcuts, or you can drag the items.

Function Alternative. Use `movefile` or `copyfile` to cut and paste or to copy and paste files or directories. For example, to make a copy of the file `myfun.m` in the current directory, assigning it the name `myfun2.m`, type

```
copyfile('myfun.m','myfun2.m')
```

Opening and Running Files

- "Opening Files" on page 5-47
- "Running M-Files" on page 5-49

Opening Files

You can open a file from the Current Directory browser and the file opens in the tool associated with that file type.

To open a file, select one or more files and perform one of the following actions:

- Press the **Enter** or **Return** key.
- Right-click and select **Open** from the context menu.
- Double-click the file(s).

The file opens in the appropriate tool, provided that the tool has been installed on your system. For example, the Editor/Debugger opens for M-files, and Simulink opens for model (`.mdl`) files.

To open a file in the Editor/Debugger, no matter what type it is, select **Open as Text** from the context menu. One exception is P-files (`.p`), which you cannot open.

To open a file using an external application, select **Open Outside MATLAB** from the context menu. For example, if you select `myfile.doc`, **Open Outside MATLAB** opens `myfile.doc` in Microsoft Word, assuming you have the `.doc` file association configured to start Word. This is also useful for file types associated with MATLAB that are also associated with an external application in Windows. For example, `.mat` is the extension for MATLAB data files as well as Microsoft Access files. When you double-click a `.mat` file in the Current Directory browser, it loads the MATLAB data file into the workspace. If instead you want to open the `.mat` file in Access, right-click it and select **Open Outside MATLAB**. MATLAB opens the file using the applications you associated with that file type in Windows. For more information, see "Changing File Associations for MATLAB from Windows" on page 1-5.

You can also import data from a file. Select the file, right-click, and select **Import Data** from the context menu. The Import Wizard opens. See the Import Wizard documentation for instructions to import the data.

You can run a Windows shortcut directly from the Current Directory browser. Double-click the shortcut icon in the Current Directory browser to perform the Windows operation.

Function Alternative. Use the open function to open a file in the tool appropriate for the file, given its file extension. Default behavior is provided for standard MATLAB file types. You can add other file types and override the default behavior for the standard files. For name.ext, open performs the following actions.

File Type	Extension	Action
Figure file	fig	Opens figure name.fig in a figure window.
HTML file	html	Opens HTML file name.html in the MATLAB Web browser.
M-file	m	Opens M-file name.m in the Editor/Debugger.
MAT-file	mat	Opens MAT-file name.mat in the Import Wizard.
Model	mdl	Opens model name.mdl in Simulink.
PDF file	pdf	Opens the PDF file name.pdf in the installed PDF reader, for example, Adobe Acrobat.
Project file	prj	Opens the project file name.prj in the MATLAB Compiler Deployment Tool. If the MATLAB Compiler or Deployment Tool is not installed, opens the project file in a text editor.

File Type	Extension	Action
Variable	none	Opens the numeric or string array `name` in the Array Editor; `open` calls `openvar`.
Other	`custom`	Opens `name.custom` by calling the helper function `opencustom`, where `opencustom` is a user-defined function.

Use `winopen` to open a file using an external application on Windows platforms.

To view the content of an ASCII file, such as an M-file, use the `type` function. For example

```
type('startup')
```

displays the contents of the file `startup.m` in the Command Window.

Running M-Files

To run an M-file from the Current Directory browser, select it, right-click, and select **Run** from the context menu. The results appear in the Command Window.

Finding Files and Content Within Files

Use the Find Files tool to search for files or for specified text within files.

Results of find. Click a column heading to change sort order.

Close current tab pane.

Type filename or text (or both) you want to find.

Restrict file types.

Select directories to search in.

Start the find operation.

Option to specify exact match.

View results of previous searches in their own tab panes.

Close **Find Files** tool.
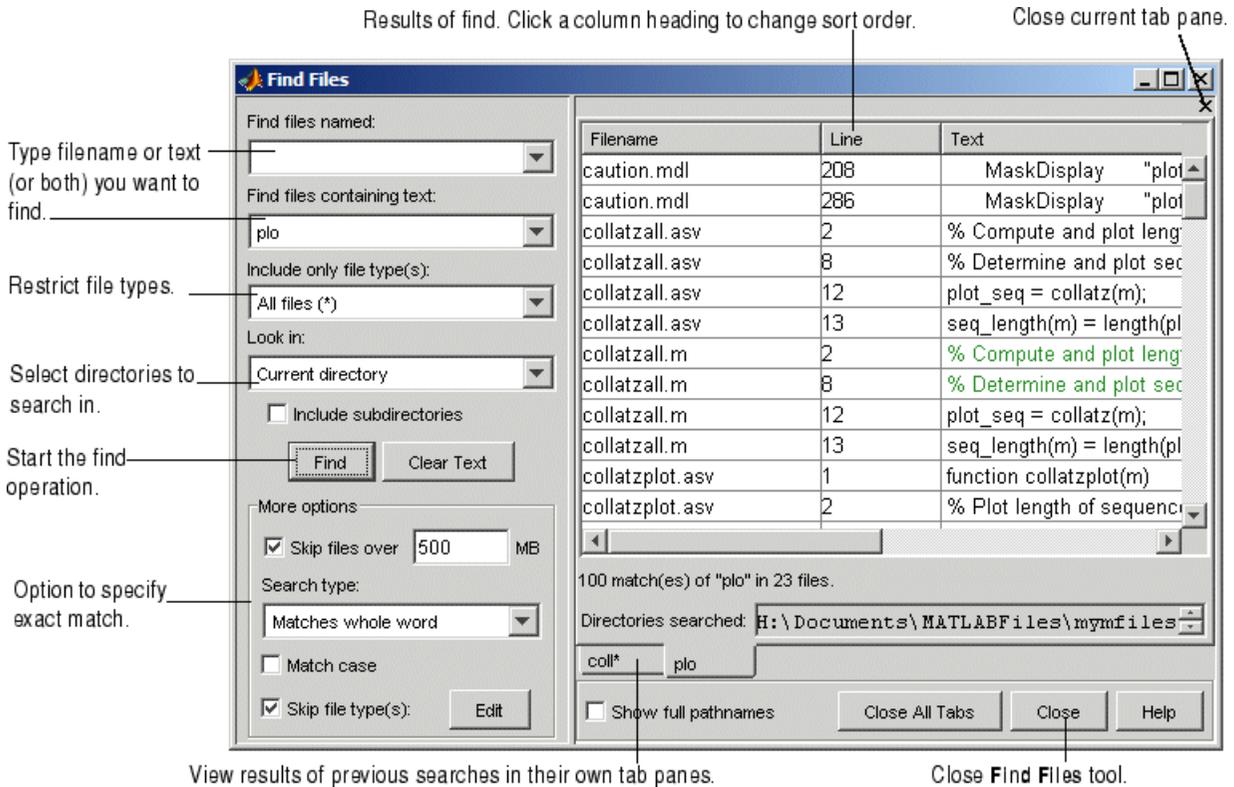
To search for files in one or more directories, or to search for specified text in files, follow these instructions:

1 Open the **Find Files** tool by clicking the Find Files button 🔍 in the Current Directory browser toolbar, or by selecting **Edit > Find Files** from any desktop tool, such as the Current Directory browser or the Editor/Debugger.

The **Find Files** dialog box opens.

2 Type the filename and/or text you are searching for:

- To search for files, type the filename in the **Find files named** field. You can use the wildcard character (*) in the filename. For example, type coll* to search for filenames that start with coll.

- To search for text within files, type the text in the **Find files containing text** field. For example, search for plot. Alternatively, you can select text in the Command Window or Editor/Debugger and that text appears in the **Find files containing text** field.

  Under **More options**, use the **Search type** to specify **Matches whole word**, or specify a partial match by selecting **Contains text**.

- To search for text in specified filenames only, type entries in both fields. Use the **Clear Text** button to clear the entries in both fields.

Click the down arrow next to each field to select previous entries from the current MATLAB session.

3 You can restrict the types of files to search by selecting an option in **Include only file type(s)**. For example, select *.m to limit the search to M-files only.

With **All files (*)** selected, use **Skip file types** (under **More options**) to ignore files of the specified type. For details, see "Skip File Types in Find Files" on page 5-52.

4 From the **Look in** list box, select the directories to search in. Select the MATLAB current directory or MATLAB search path, or use the Browse option to select another directory. You can instead type the full pathname for one or more directories into this field, with each pathname separated by a semicolon (;). To include subdirectories in the search, select the **Include subdirectories** check box.

5 Use additional entries under **More options** to further restrict the search:

- **Skip files over** the specified size to ignore large files that might take a long time to search through. This option is only available when you are searching for text within files.

- **Match case** when lower or upper case is relevant.

6 To execute the search, click **Find**. While the search is in progress, the **Find** button label changes to **Stop Find**. To abort a search, click **Stop Find**.

Search results appear in the pane on the right side of the **Find Files** dialog box, with a summary of the results at the bottom of the pane. For text searches, the line number and line of code are shown. To see the full pathnames for the files, select the **Show full pathnames** check box.

7 Click a column heading to sort the results based on that column. Click the column heading again to reverse the sort order for that column. For example, click **Line** to sort results by line number.

Opening Files from Find Files

To open files shown in the results list, do one of the following:

- Double-click the file

- Select the files and press **Enter** or **Return**

- Right-click the selected files and select **Open** from the context menu

The files open in the Editor/Debugger. For text searches, the file opens scrolled to the line number shown in the results section of the **Find Files** dialog box. Once in the Editor/Debugger, you can use the Find & Replace tool to change specified text.

Previous Results of Find Files

To see the results of a previous search, select its tab at the bottom of the results pane. **Find Files** shows up 10 search result tabs while the tool is open, but does not maintain the results after you close the tool.

MATLAB maintains the state for options in the Find Files tool even after you end the session.

Skip File Types in Find Files

In the Find Files tool, you can restrict the search to look in all file types except those you specify:

1 For **Include only file type(s)**, select **All files (*)**.

2 Select the **Skip file type(s)** check box.

3 Click **Edit** to view or change the list of file types the search ignores.

  The **Edit Skipped File Extensions** dialog box opens.

To skip a file type not shown in this list, enter the extension and click **Add**. The extension then appears in the list. Be sure its **State** is selected.

When **Skip file type(s)** in the Find Files tool is selected, the search ignores file types in this list whose **State** is selected. For the example shown, the search would ignore P-files because **State** is selected, but would look in TAR-files because **State** is cleared.

Edit Skipped File Extensions

To exclude a new file type from searches (e.g. .foo), type the extension and click Add.

Add

State	Extension
☑	.mat
☑	.mex
☑	.p
☑	.pdf
☑	.ppt
☑	.pub
☑	.scc
☑	.sl
☑	.so
☐	.tar
☐	.zip

Remove   Restore Defaults

Select an extension in the list to exclude files of that type from searches. Clear it to include files of that type in searches.

OK   Help

If you do not want a file type to appear in this list, select the name of the extension, and then click **Remove**.
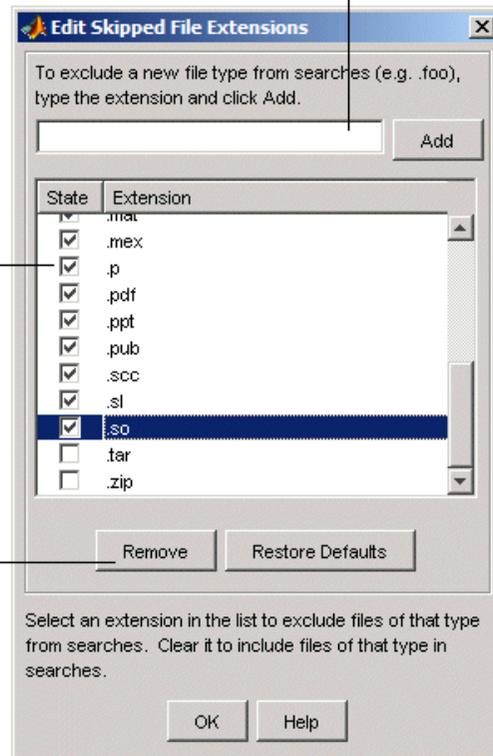
4 Find Files will not look in any file type in the list whose **State** check box is selected. It will look in any file type in the list whose **State** check box is cleared.

a Clear or select the **State** check box as needed to instruct Find Files about file types to skip.

b If you want Find Files to skip a file type not shown in the list, enter the file extension in the field at the top of the dialog box and click **Add**. The type appears in the list. Be sure its **State** check box is selected. For the example shown, the scc file type was added.

c You can reduce the size of the list by removing any file extensions. Select the name of the extension and click **Remove**.

5 Click **OK** to accept the changes and close the **Edit Skipped File Extensions** dialog box.

6 When you click **Find** in the Find Files tool, the search ignores the selected file types.

Function Alternative

Use `lookfor` to search for the specified text in the first line of help for all M-files on the search path.

Comparing Files

The MATLAB File Comparison tool highlights the line-by-line differences between two files. Access it by selecting a file in the Current Directory browser, then right-click, and from the context menu, select **Compare Against > Browse** and navigate to the file to compare. Select it and click **Open**.

For two files in the same directory, select the files, right-click, and from the context menu, select **Compare Selected Files**.

For information about using the results and other options, see "Comparing Files — File Comparison Tool" on page 6-54.

Accessing Source Control Features

Select a file or files in the Current Directory browser and right-click to view the context menu. From there you can access features for Source Control. For details on these features, see Chapter 9, "Source Control Interface".

Preferences for the Current Directory Browser

Using preferences, you can specify the number of recently used current directories to maintain in the history list as well as the type of information to display in the Current Directory browser.

From the Current Directory browser, select **File > Preferences**. The **Current Directory Preferences** pane appears in the Preferences dialog box.
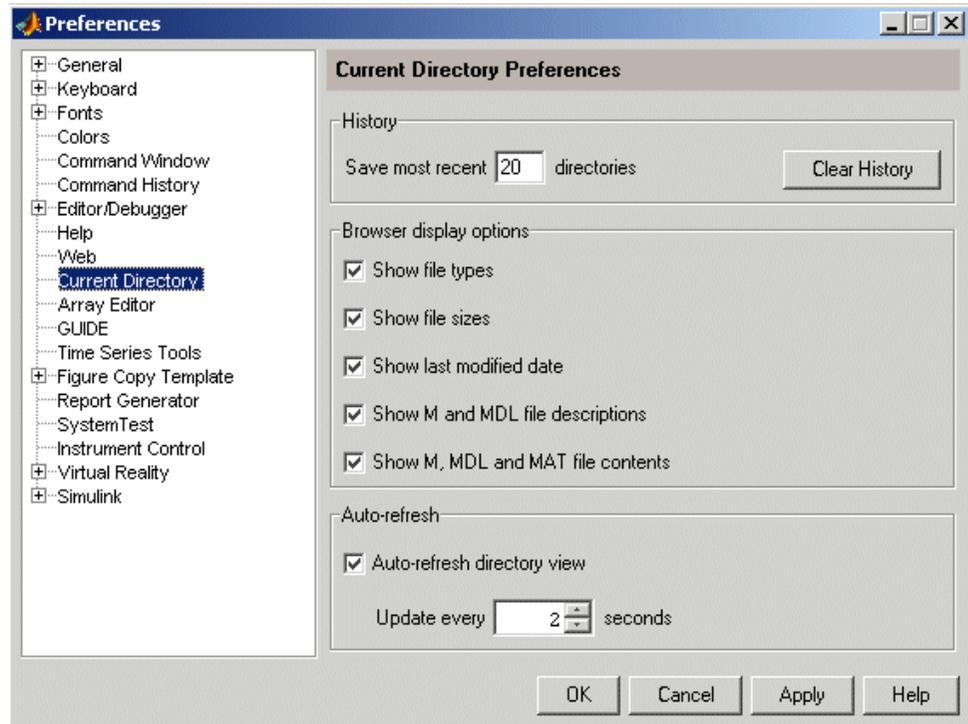
History

The drop-down list in the current directory field shows the history of current directories, that is, the most recently used current directories.

Saving Directories. When the MATLAB session ends, the list of directories will be maintained. Use the **Save most recent directories** field to specify how many directories will appear on the list at the start of the next MATLAB session.

Removing Directories. To remove all entries in the list, click **Clear History**. The list is cleared immediately.

Browser Display Options

In the Current Directory browser, you can view or hide the following information by selecting the appropriate **Browser display options**:

- File type

- File size

- Last modified date

- M-file descriptions (the first comment line in the M-file, also called the H1 line) and the start of MDL file descriptions (approximately the first 128 characters)

- M-file help, MDL complete descriptions, and MAT-file contents

For more information, see "Changing the Display" on page 5-41.

Auto-Refresh

By default, the **Auto-refresh directory view** check box is selected, with an update time of 2 seconds. This means that every 2 seconds, the Current Directory browser checks for and reflects any changes you made to files and directories in the current directory using other applications.

In some cases when the current directory is on a network, MATLAB becomes slow because of the auto-refresh feature in the Current Directory browser. If you experience general slowness in MATLAB and have the Current Directory browser open, try increasing the default update time to alleviate this problem. For extremely slow performance situations, clear the check box to turn auto-refresh off. You can then right-click and select **Refresh** from the context menu to update the Current Directory browser display.

Editing and Debugging M-Files

MATLAB provides powerful tools for creating, editing, and debugging files, as detailed here. For information about the MATLAB language and writing M-files, see the MATLAB Programming documentation.

Begin with Existing Code

In this section...
"Create M-Files from Command Window and History" on page 6-3
"Use Existing M-Files and Examples" on page 6-3

Create M-Files from Command Window and History

Before you begin writing MATLAB code in a blank file, consider starting with existing resources for the code, and then use the MATLAB Editor/Debugger to modify the code.

In many cases, you create and run MATLAB statements in the Command Window, modify those statements to your satisfaction, and then create an M-file that includes the statements. To facilitate this process, in the Command History, select the MATLAB statements you want to include in the M-file. Right-click and select **Create M-File**. The Editor/Debugger opens a new file that includes the statements you selected from the Command History. You can also copy the statements from the Command History and paste them into an existing M-file.

Use Existing M-Files and Examples

If you can find existing M-files that accomplish what you want to do, copy and use the code in your own M-file, assuming you have legal permission to do so. Following are some resources you can use.

MATLAB and Toolbox M-Files

You can access and reuse the code in most MATLAB and toolbox functions that have a `.m` file extension. You cannot use MATLAB and toolbox functions that are built-in. They are efficient but their code is not accessible.

If there is a MATLAB function that is similar to what you need to do and it is not built-in, open the file in the Editor/Debugger and use it as a basis for your file. Be sure to save the file using a different name and in a directory that is not in `matlabroot/toolbox`. See "Saving Files" on page 6-62 for details.

Demos and Examples

MATLAB and its toolboxes include demonstration programs. You can view the code in the demos and copy it for use in your own M-files. To see the demos, type demo, which opens the Help browser to the **Demos** pane. For more information about demos, see "Demos in the Help Browser" on page 4-31.

There are also code examples in the online documentation. To see a list of examples for a product, type helpbrowser to open the Help browser. In the **Contents** pane, click **+** for a product to view the help topics, and then select the 💡 **Examples** entry.

File Exchange

The MathWorks Web site features a user-contributed code library, from which you can download free M-files contributed by users and developers of MATLAB, Simulink, and related products. To view the files available to download, go to the MATLAB Central File Exchange page on the MathWorks Web site, http://www.mathworks.com/matlabcentral/fileexchange/index.jsp, or access it via the **Help > Web** menu in any desktop component.

Ways to Edit, Evaluate, and Debug M-Files

There are several methods for creating, editing, evaluating, and debugging files with MATLAB.

Creating and Editing Files — Options	Instructions
MATLAB Editor/Debugger	See "Starting, Customizing, and Closing the Editor/Debugger" on page 6-7, and "Saving, Printing, and Closing Files in the Editor/Debugger" on page 6-62. You can create, open, edit and save M-files as well as other file types in the MATLAB Editor/Debugger — see "Creating and Editing Other Text File Types" on page 6-12.
Any text editor, such as Emacs or vi	To specify another editor as the default for use with MATLAB, select **File > Preferences > Editor/Debugger**, and for **Editor**, specify the **Text editor**. Click the **Help** button in the Preferences dialog box for details. Use that editor by default, or use any other editor you open. Regardless of the editor you use, you can debug M-files using the MATLAB Editor/Debugger or debugging functions.

Debugging M-Files – Options	Instructions
General debugging tips	See "Finding Errors, Debugging, and Correcting M-Files" on page 6-84.
MATLAB Editor/Debugger	See • "M-Lint Code Analyzer" on page 6-87 to identify errors and make improvements. • "Debugging Process and Features" on page 6-103 to help you isolate run-time problems.
MATLAB debugging functions (for use in the Command Window)	See function alternatives in "Debugging Process and Features" on page 6-103.

Use preferences for the Editor/Debugger to set up the editing and debugging environment to best meet your needs.

For information about the MATLAB language and writing M-files, see the MATLAB Programming documentation.

Starting, Customizing, and Closing the Editor/Debugger

In this section...
"Starting the Editor/Debugger" on page 6-7
"Creating a New File in the Editor/Debugger" on page 6-8
"Opening Existing Files in the Editor/Debugger" on page 6-9
"Arranging Editor/Debugger Documents" on page 6-11
"Preferences for the Editor/Debugger" on page 6-11
"Creating and Editing Other Text File Types" on page 6-12
"Closing the Editor/Debugger" on page 6-13

Starting the Editor/Debugger

The MATLAB Editor/Debugger provides a graphical user interface for basic text editing features for any file type, as well as for M-file debugging. The Editor/Debugger is a single tool that you can use for editing, debugging, or both. There are various ways to start the Editor/Debugger. The Editor/Debugger automatically starts when you open a document or create a new one. Once started, you can customize the Editor/Debugger to suit your needs.

This figure shows an example of the Editor/Debugger outside of the desktop opened to an existing M-file, and calls out some of the tool's useful features.

The * after the filename indicates the file has not been saved since it was last modified.

Set the right-hand text limit (red line) using **Preferences -> Display**.

Toolbar for cell features.

Set breakpoints in the alley.

Colors highlight syntax elements.

```
D:\mymfiles\collatz.m*
File  Edit  Text  Go  Cell  Tools  Debug  Desktop  Window  Help

 1    function sequence=collatz(n)
 2    % Collatz problem. Generate a sequence of integers resolving to 1
 3    % For any positive integer, n:
 4    %    Divide n by 2 if n is even
 5    %    Multiply n by 3 and add 1 if n is odd
 6    %    Repeat for the result
 7    %    Continue until the result is 1
 8
 9    sequence = n;
10    next_value = n;
11    while next_value > 1
12        if rem(next_value,2)==0
13            next_value = next_value/2;
14        else
```

collatz   Ln 9   Col 12   OVR

Status bar information includes cursor line and column numbers.

Creating a New File in the Editor/Debugger

To create a new text file in the Editor/Debugger, either click the New M-file button ☐ on the MATLAB desktop toolbar, or select **File > New > M-File** from the MATLAB desktop. The Editor/Debugger opens, if it is not already open, with an untitled file in the MATLAB current directory, in which you can create an M-file or another type of text file.

The location of the new file and the Editor/Debugger are determined by document positioning guidelines. You can rearrange the documents to suit your needs. For details, see "Opening and Arranging Documents" on page 2-8.

If the Editor/Debugger is open, create more new files by using the New M-file button ☐ on the toolbar, or select **File > New > M-File**.

Other tools also provide features for creating new M-files. For example, in the Command History, select statements, right-click, and select **Create M-File** from the context menu. Similarly, create a new file from the context menu in the Current Directory browser — see "Creating New Files" on page 5-44.

Function Alternative

Type `edit` in the Command Window to create a new file in the Editor/Debugger.

Type `edit filename.ext` to create the file `filename.ext`. If `filename.ext` already exists in the current directory or on the MATLAB search path, this opens the existing file. If `filename.ext` does not exist in the current directory or on the MATLAB search path, a confirmation dialog box might appear asking if you want to create a new file titled `filename.ext`:

- If you click **Yes**, the Editor/Debugger creates a blank file titled `filename.ext`. If you do not want the dialog to appear in this situation, select that check box in the dialog. Then, the next time you type `edit filename.ext`, the file is created without first prompting you.

- If you click **No**, the Editor/Debugger does not create a new file. If you do not want the dialog to appear in this situation, select that check box in the dialog. In that case, the next time you type `edit filename.ext`, a "file not found" message appears.

For more information about the confirmation dialog box, see preferences for "Confirmation Dialogs Preferences" on page 2-81.

Opening Existing Files in the Editor/Debugger

To open an existing file in the Editor/Debugger, click the Open file button on the desktop or Editor/Debugger toolbar, or select **File > Open**.

The **Open** dialog box appears, listing all M-files. You can see different files by changing the selection for **Files of type** in the dialog box. Type or select a filename, and click **Open**. If you access the **Open** dialog box from the desktop, the current directory files are shown, but if you access it from the Editor/Debugger, the files in the directory for the current file are shown.

The Editor/Debugger opens, if it is not already open, with the file displayed. You can have multiple Editor/Debugger files open at once, and the location of the files and the Editor/Debugger are determined by document positioning guidelines. You can rearrange the documents to suit your needs. For details, see "Opening and Arranging Documents" on page 2-8.

To make a document in the Editor/Debugger become the current document, click it, or select it from the **Window** menu or document bar.

M-File Cells

If you open an M-file that contains M-file cells, yellow highlighting and gray horizontal lines might appear in the M-file, along with an information toolbar. Cell mode is used for publishing results and rapid code iteration. An M-file cell is denoted by a %% at the start of a line. MATLAB interprets any M-file that contains %% at the start of a line as including cells and the Editor/Debugger reflect the cell toolbar state and the cell display preferences, such as yellow highlighting of the current cell and gray lines between cells.

The first time you open an M-file that contains cells, an information bar appears below the cell toolbar, providing links for details about cell mode. To dismiss the information bar, click the close box on the right side of the bar. The information bar does not appear again, but you can get the same quick access to the information about M-file cells from the information button 🛈 on the cell toolbar.

To hide the cell toolbar, right-click in the toolbar and select **Cell Toolbar** from the context menu. If you do not want cell mode enabled, select **Cell > Disable Cell Mode**. Because MATLAB remembers the cell mode between sessions, if cell mode is disabled when you quit MATLAB, it is disabled the next time you start MATLAB, and the converse is true.

Other Methods for Opening Files in the Editor/Debugger

These are other ways to open files in the Editor/Debugger:

- Drag a file from another MATLAB desktop tool or a Windows tool into the Editor/Debugger. For example, drag files from the Current Directory browser, or from Windows Explorer.

- Open files from the Current Directory browser — see "Opening Files" on page 5-47.

- Select a file to open from the most recently used files, which are listed at the bottom of the **File** menu in the Editor/Debugger and all other desktop tools. You can change the number of files appearing on the list — select

File > Preferences > Editor/Debugger and in the **Most recently used file list**, specify the **Number of entries**.

- In the Editor/Debugger or another desktop tool such as the Command Window, select a filename, right-click, and select **Open Selection** from the context menu to open that file. For details, see "Opening a Selection in an M-File" on page 6-48.

- Set a preference that instructs MATLAB, upon startup, to automatically open the files that were open when the previous MATLAB session ended. Select **File > Preferences > Editor/Debugger** and in the **Opening files in editor area**, select the check box for **On restart reopen files from previous MATLAB session**.

Function Alternative for Opening an M-File. Use the `edit` or `open` function to open an existing file in the Editor/Debugger. For example, type

```
edit collatz.m
```

to open the file `collatz.m` in the Editor/Debugger, where `collatz.m` is on the search path or in the current directory. Use the relative or absolute pathname for the file you want to open if it is not on the search path or in the current directory.

Arranging Editor/Debugger Documents

You can arrange the size and location of M-files and other text documents you open in the Editor/Debugger. Editor/Debugger documents follow the same arrangement practices as other desktop documents. For details, see "Opening and Arranging Documents" on page 2-8.

Preferences for the Editor/Debugger

Using preferences, you can specify the default behavior for various aspects of the Editor/Debugger.

To set preferences for the Editor/Debugger, select **File > Preferences**. The Preferences dialog box opens showing **Editor/Debugger Preferences**.

Appears only if EmacsLink is registered with MATLAB.
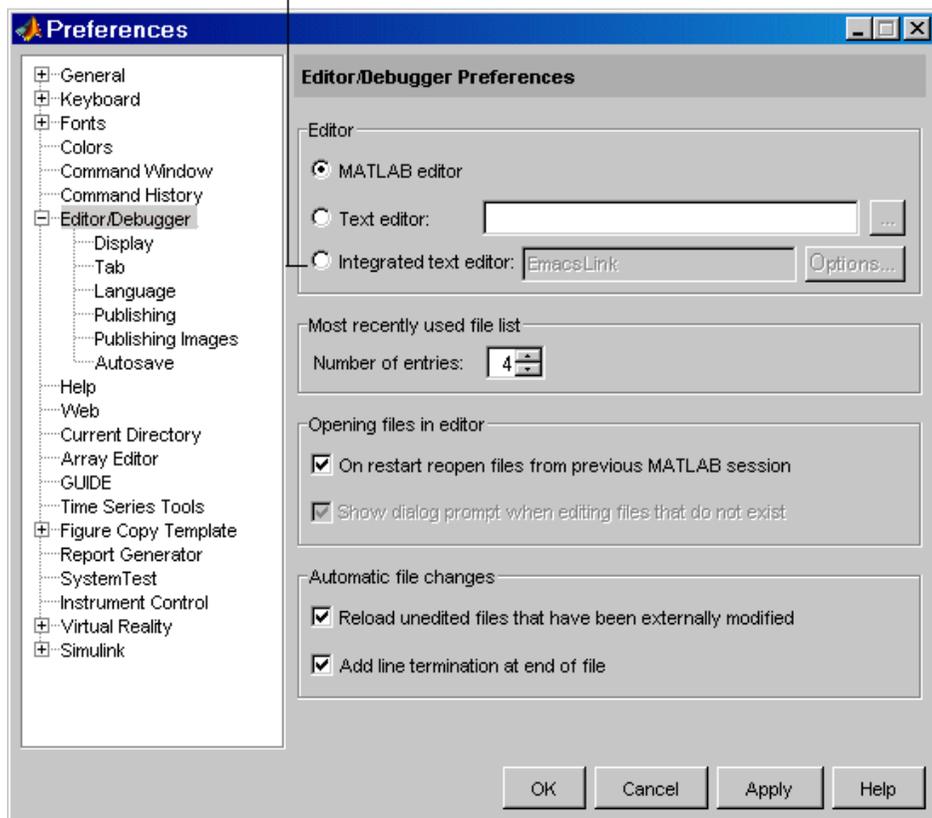
Click the **+** next to Editor/Debugger in the left pane to view all categories of Editor/Debugger preferences. Select a category and that preference pane displays. Make changes and click **Apply** or **OK**.

Click the **Help** button in the Preferences dialog box for details about Editor/Debugger preferences.

Creating and Editing Other Text File Types

You can edit any type of text file using the MATLAB Editor/Debugger. For example, you can open and edit an HTML file. Note that you can run or debug only M-files from the Editor/Debugger.

When working with files created for C/C++, Java, and HTML, you can specify syntax highlighting and indenting preferences appropriate to those languages. Select **File > Preferences > Editor/Debugger > Language**. See details in the online documentation for Editor/Debugger language preferences, or click the **Help** button in the dialog box.

Closing the Editor/Debugger

To close the Editor/Debugger, click the **Close** box in the title bar of the Editor/Debugger. This is different from the **Close** box in the menu bar of the Editor/Debugger, which closes the current file when multiple files are open in a single window.

Close box for Editor/Debugger.

Close box for current file.
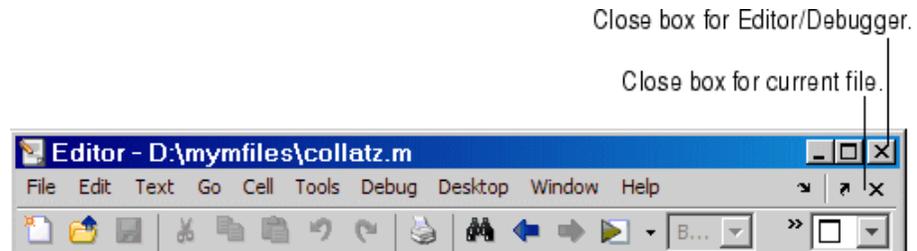
If multiple files are open, with each in a separate window, close each window separately. To close all files at once, select **Close All Documents** from the **Window** menu. Note that this will close other desktop documents as well, such as arrays in the Array Editor, and it will close the tools as well, that is, the Editor/Debugger and Array Editor, for example.

When you close the Editor/Debugger and any of the open files have unsaved changes, you are prompted to save the files.

Entering Statements in the Editor/Debugger

Use Command Window Features in the Editor/Debugger

After opening an existing file or creating a new file in the Editor/Debugger, enter statements in the file. Follow the same rules you would use for entering statements in the Command Window as described in Chapter 3, "Running Functions — Command Window and History":

- "Case and Space Sensitivity" on page 3-15
- "Matching Delimiters (Parentheses)" on page 3-17
- "Entering Multiple Functions in a Line" on page 3-18
- "Entering Long Statements (Line Continuation)" on page 3-18
- "Suppressing Output" on page 3-30
- "Formatting and Spacing Numeric Output" on page 3-31

In addition, utilize the Editor/Debugger features described in the remaining parts of this section.

Changing the Case of Selected Text

To change the case of text in the Editor/Debugger, select the text and then from the **Text** menu, select one of the following:

- **Change to Upper Case** to change all text to uppercase
- **Change to Lower Case** to change all text to lowercase

• **Reverse Case** to change the case of each letter

This is useful, for example, when copying syntax from help in an M-file, where function and variable names are distinguished by the use of uppercase. But because of that, the code will not run in MATLAB. In this example, the text was copied and pasted from the output of help get.

```
V = GET(H, 'Default')
```

Select all of the text. Select **Text > Change to Lower Case**. The text becomes

```
v = get(h, 'default')
```

If instead you select **Reverse Case** for

```
V = GET(H, 'Default')
```

the case changes to

```
v = get(h, 'dEFAULT')
```

Undo and Redo

You can undo many of the Editor/Debugger actions listed in **Edit** and **Text** menus. Select **Edit > Undo**. You can undo multiple times in succession until there are no remaining actions to undo. Select **Edit > Redo** to reverse an undo.

Adding Comments

Comments in an M-file are strings or statements that do not execute. Add comments in an M-file to describe the code or how to use it. Comments determine what text displays when you run help for a filename. Use comments when testing your files or looking for errors—temporarily turn lines of code into comments to see how the M-file runs without those lines. These topics provide details:

• "Commenting in M-Files Using the MATLAB Editor/Debugger" on page 6-16

- "Commenting in Java and C/C++ Files Using the MATLAB Editor/Debugger" on page 6-17
- "Commenting in M-File Using Any Text Editor" on page 6-17
- "Commenting Out Part of a Statement" on page 6-19
- "Formatting Comments in M-Files" on page 6-20

Commenting in M-Files Using the MATLAB Editor/Debugger

You can comment the current line or a selection of lines in an M-file:

1 For a single line, position the cursor in that line. For multiple lines, click in the line and then drag or **Shift**+click to select multiple lines.

2 Select **Comment** from the **Text** menu, or right-click and select it from the context menu.

A comment symbol, %, is added at the start of each selected line, and the color of the text becomes green or the color specified for comments — see "Syntax Highlighting" on page 6-28.

To uncomment the current line or a selected group of lines, select **Uncomment** from the **Text** menu, or right-click and select it from the context menu.

Click in the area to the left of a line to select that line.
To select multiple lines, click+drag or **Shift**+click.

Select **Text -> Comment** to make all the selected lines comments.
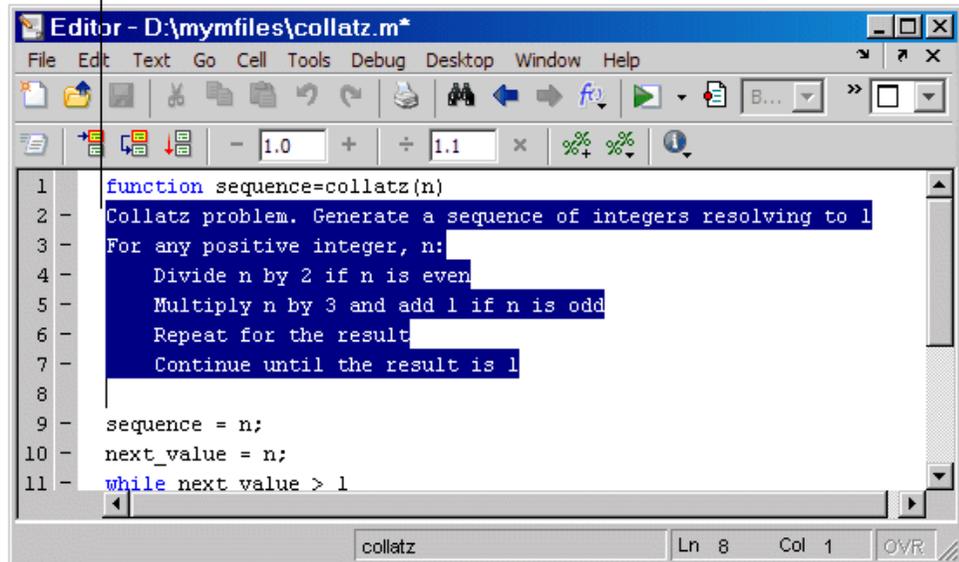
Commenting in Java and C/C++ Files Using the MATLAB Editor/Debugger

For Java and C/C++ files, selecting **Text > Comment** adds the // symbols at the front of the selected lines. Similarly, **Text > Uncomment** removes the // symbols from the front of selected lines in Java and C/C++ files.

Commenting in M-File Using Any Text Editor

You can make any line in an M-file a comment by typing % at the beginning of the line. To put a comment within a line, type % followed by the comment text; MATLAB treats all the information after the % on a line as a comment.

% This is a comment.————MATLAB ignores this comment line when you run the M-file.
This is not a comment.————This line produces an error when you run the M-file.

To uncomment any line, delete the comment symbol, %.

To comment a contiguous group of lines, type %{ before the first line and %} after the last line you want to comment. This is referred to as a block comment. The lines that contain %{ and %} can contain spaces, but not contain any other text. After typing the opening block comment symbol, %{, all subsequent lines assume the syntax highlighting color for comments until you type the closing block comment symbol, %}. Remove the block comment symbols, %{ and %}, to uncomment the lines.

This examples shows some lines of code commented out. When you run the M-file, the commented lines will not execute. This is useful when you want to identify the section of a file that is not working as expected.

Comment a block of code by adding %{ before
the first line and %} after the last line.

```
a = magic(3)
%{
sum(a)
diag(A)
sum(diag(a))
%}
sum(diag(fliplr(a)))
```

You can easily extend a block comment without losing the original block comment, that is, create a nested block comment, as shown in the following example.

Create a nested comment, that is, a block comment
within a block comment.

```
                              %{
                              a = magic(3)
          Original            %{
          comment             sum(a)
                              diag(A)
                              sum(diag(A))
Extended                      %}
comment                       sum(diag(fliplr(a)))
                              %}
```
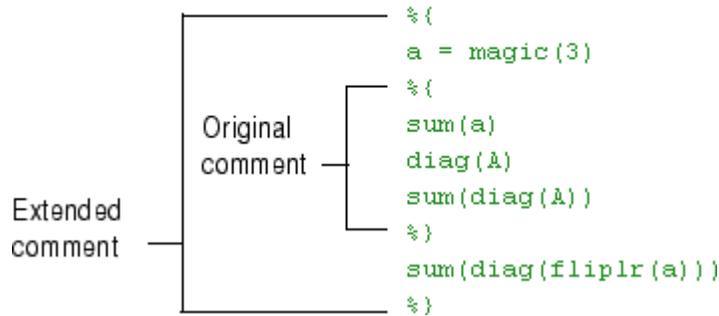
Commenting Out Part of a Statement

To comment out the end of a statement in an M-file, put the comment
character, %, before the comment. When you run the file, MATLAB ignores
any text on the line after the %.

Any text following a % within a line is
considered to be a comment.

```
a = zeros(10)  % Initialize matrix
```

To comment out text within a multiline statement, use the ellipsis (...).
MATLAB ignores any text appearing after the ... on a line and continues
processing on the next line. This effectively makes a comment out of anything
on the current line that follows the .... The following example comments
out the Middle Initial line.

```
header = ['Last Name, '...
'First Name, '...
... 'Middle Initial, '...
'Title']
```

MATLAB ignores the text following the ... on the line

```
... 'Middle Initial, '...
```

Note that `Middle Initial` is green, which is the syntax highlighting color for a comment.

MATLAB continues processing the statement with the next line

```
'Title']
```

MATLAB effectively runs

```
headers = ['Last Name, ' ...
'First Name, ' ...
'Title']
```

Formatting Comments in M-Files

To make comment lines in M-files wrap when they reach a certain column,

1 Specify the maximum column number using preferences for the Editor/Debugger. Select **Language > M**. For **Comment formatting**, set the **Max width**.

2 Select contiguous comment lines that you want to limit to the specified maximum width.

3 Select **Text > Wrap Selected Comments**.

   The selected comment lines are reformatted so that no comment line in the selected area is longer than the maximum. Lines that were shorter than the specified maximum are merged to make longer lines if they are at the same level of indentation.

To automatically limit comment lines to the maximum width while you type, select the **Comment formatting** preference to **Autowrap comments**.

For example, assume you select **Autowrap comments** and set the maximum width to be 75 characters, which is the width that will fit on a printed page using the default font for the Editor/Debugger. When typing a comment line,

as you reach the 75th column, the comment automatically continues on the next line.

Tab Completion in the Editor/Debugger

The Editor/Debugger helps you automatically complete the names of these items as you type them in an M-file:

- Functions or models on the search path or in the current directory

- Variables, including structures, in the current workspace, where the current workspace is shown in the **Stack** on the toolbar.

- Handle Graphics properties for figures in the current workspace

Type the first few characters of the item name and then press the **Tab** key. To use tab completion, you must have the tab completion preference for the Editor/Debugger selected. For details, see "Keyboard Preferences" on page 3-43.

Tab completion is also available in the Command Window. There are a few minor differences in how tab completion works in the Command Window, the most notable being that Command Window tab completion supports the completion of filenames, whereas the Editor/Debugger tab completion does not.

Note Tab completion does not complete the names of variables you define in an M-file, but only those variables in the current workspace. This means that while editing, it only completes the names of variables in the base workspace. While debugging, it only completes the names of variables in the current function workspace.

These examples demonstrate how to use tab completion:

- "Basic Example — Unique Completion" on page 6-22

- "Multiple Possible Completions" on page 6-23

- "Narrowing Completions Shown" on page 6-24

Basic Example — Unique Completion

This example illustrates a basic use for tab completion in the Editor/Debugger.
In an M-file opened in the Editor/Debugger, type the beginning of a function
or model on the MATLAB search path or in the current directory, for example,

```
horz
```

and press **Tab**. The Editor/Debugger automatically completes the name,
which for this example displays the function name

```
horzcat
```

Complete the statement, adding any arguments, operators, or options. If the
Editor/Debugger does not complete the name horzcat but instead moves the
cursor to the right, you do not have the preference set for tab completion. The
Editor/Debugger also moves the cursor to the right when you try to complete a
filename; filename tab completion is not supported in the Editor/Debugger,
but is supported in the Command Window.

You can use tab completion anywhere in the line, not just at the beginning.
For example, if you type

```
a = horz
```

and press **Tab**, the Editor/Debugger completes horzcat.

The Editor/Debugger also completes the names of variables in the current
workspace. For example, if there is a variable costs_march in the currently
selected workspace, type cost and press **Tab**. The Editor/Debugger completes
the variable name costs_march. If MATLAB displays No Completions
Found, costs_march does not exist in the current workspace.

Multiple Possible Completions

If there is more than one name that starts with the characters you typed, when you press the **Tab** key, the Editor/Debugger displays a list of all names that start with those characters. For example, assume you had created the variable costs_march in the base workspace. In an M-file in the Editor/Debugger, type

    cos

and press **Tab**. The Editor/Debugger displays

The resulting list of possible completions includes the variable name you created, costs_march, but also includes functions and models that begin with cos, including cosets from Communications Toolbox, if it is installed and on the MATLAB search path.

Continue typing to make your entry unique. For example, type the next character, such as t in the example. The Editor/Debugger selects the first item in the list that matches what you typed, in this case, costs_march. Press **Enter** (or **Return**) or **Tab** to select that item, which completes the name in the M-file. In the example, MATLAB displays costs_march at the prompt.

You can navigate the list of possible completions using up and down arrow keys, and **Page Up** and **Page Down** keys. You can clear the list without selecting anything by pressing **Esc**. The list of possible completions might include items that are not valid commands, such as private functions.

Narrowing Completions Shown

You can narrow the list of completions shown by typing a character and then pressing **Tab** if the **Keyboard** preference **Tab key narrows completions** is selected. This is particularly useful for large lists. For example, type cam and press **Tab** to see the possible completions. There is a scroll bar with the list because there are too many completions to be seen at once.
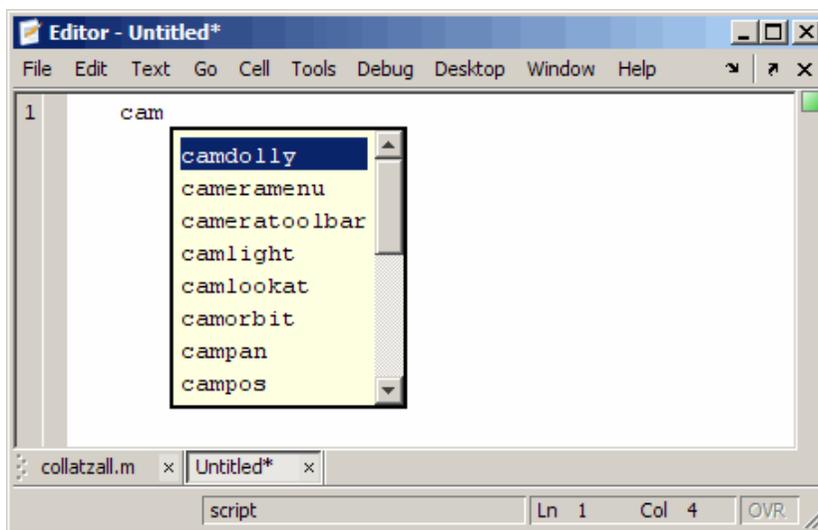
Type p and press **Tab** again. The Editor/Debugger narrows the list, showing only all possible camp completions.
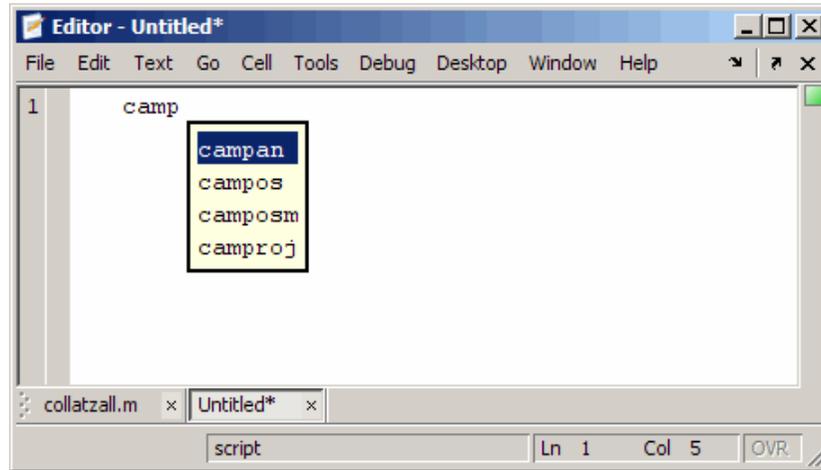
Continue narrowing the list in the same way. For the above example, type o and press **Tab** to further narrow the list. Press **Enter** or **Return** to select an item, which completes the name at the prompt.

Tab Completion for Structures

For structures that are in the current workspace, after the period separator, press **Tab**. For example, type

```
mystruct.
```

and press **Tab** to display all fields of mystruct. If you type a structure and include the start of a unique field after the period, pressing **Tab** completes that structure and field entry.

For example, type

```
mystruct.n
```

and press **Tab**, which completes the entry mystruct.name, where mystruct is in the current workspace and contains no other fields that begin with n.

Tab Completion for Properties

Complete property names for figures in the current workspace using tab completion, as in this graphics example. Here, f is a figure. Type

```
set(f, 'pap
```

and press **Tab**. The Editor/Debugger displays
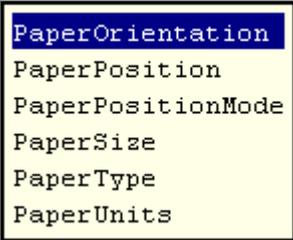
Select a property from the list. For example, type

```
u
```

and press **Enter**. The Editor/Debugger completes the property, including the closing quote.

```
set(f, 'paperunits'
```

Continue adding to the statement, as in this example,

```
set(f, 'paperunits', 'c
```

and press **Tab**. The Editor/Debugger automatically completes the property

```
set(f, 'paperUnits', 'centimeters'
```

because centimeters is the only possible completion.

Using Tab for Spacing

If the preference for tab completion is selected, and you want to also use the **Tab** key to add spacing within your statements, add a space before pressing **Tab**. For example, to create this statement

```
if a=mate      %test input value
```

add a space after `mate` and then press **Tab**. If you do not include the space, the following happens instead:

```
if a=material
```

This is because the tab completion feature automatically causes `mate` to complete as the `material` function.

Alternatively, turn off the tab completion preference to use **Tab** for spacing in the Editor/Debugger.

Appearance of an M-File — Making Files More Readable

In this section...
"Syntax Highlighting" on page 6-28
"Indenting" on page 6-29
"Function Indenting" on page 6-29
"Line and Column Numbers" on page 6-30
"Highlight Current Line" on page 6-30
"Right-Hand Text Limit" on page 6-31
"View Function or Subfunction" on page 6-31
"Code Folding—Expanding and Collapsing M-File Constructs" on page 6-31
"Split Screen Display" on page 6-38

Note You can specify the default behaviors for some of these features—see "Preferences" on page 2-59.

Syntax Highlighting

Some entries appear in different colors to help you better find matching elements, such as if/else statements. Similarly, unterminated strings have a different color than terminated strings. This is called syntax highlighting and is used in the Command Window and History, as well as in the Editor/Debugger. For more information, see the Command Window documentation for "Syntax Highlighting" on page 3-16.

When you paste or drag a selection from the Editor/Debugger to another application, such as Microsoft Word, the pasted text maintains the syntax highlighting colors and font characteristics from the Editor/Debugger. MATLAB pastes the selection to the clipboard in RTF format, which many Windows and Macintosh applications support.

Indenting

Automatic Indenting

You can set an indenting preference so that program control entries are automatically indented to make reading loops, such as `while/end` statements, easier. To do so, select **File > Preferences > Editor/Debugger > Language**, and select a **Language**, for example, `M`. For **Indenting for Enter key**, select **Smart indenting** or **Block indent**, and then click **OK**. Use **No indent** instead if you want to indent manually. For more information about indenting preferences, see the in the online documentation. Specify the indenting size and other options using in the online documentation.

Manual Indenting

You can manually apply smart indenting to selected lines — select the lines and then select **Smart Indent** from the **Text** menu, or right-click and select it from the context menu. This feature indents lines that start with keyword functions or that follow lines containing certain keyword functions. Smart indenting can help you to follow the code sequence.

To move the current or selected lines further to the left, select **Decrease Indent** from the **Text** menu. To move the current or selected lines further to the right, select **Increase Indent** from the **Text** menu.

You can also indent a line by pressing the **Tab** key at the start of a line. Or select a line or group or lines and press the **Tab** key. Press **Shift+Tab** to decrease the indent for the selected lines. This works differently if you select the Editor/Debugger **Tab** preference for **Emacs-style Tab key smart indenting** — when you position the cursor in any line or select a group of lines and press **Tab**, the lines indent according to smart indenting practices.

For more information about manual indenting, see in the online documentation.

Function Indenting

If you select the language preference for smart indent, you can select from three indenting options when you enter a subfunction or a nested function

(a function within a function) in the Editor/Debugger. For details, see in the online documentation.

Line and Column Numbers

Line numbers are displayed along the left side of the Editor/Debugger window. You can elect not to show the line numbers using preferences — for details, see in the online documentation.

The line and column numbers for the current cursor position are shown in the far right side of the status bar in the Editor/Debugger.

Highlight Current Line

You can set a preference to highlight the current line, that is the line with the caret (also called the cursor). This is useful, for example, to help you see where copied text will be inserted when you paste.

To highlight the current line, select
File > Preferences > Editor/Debugger > Display, and
under **General display options**, select the check box for **Highlight current line**. You can also specify the color used to highlight the line.

Current line (where the caret/cursor) is highlighted.
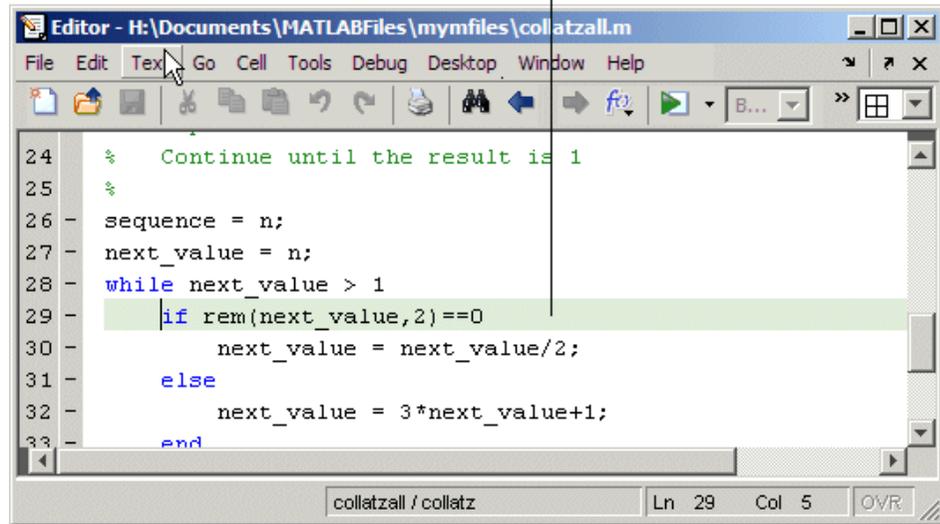
Right-Hand Text Limit

By default, a light red vertical line (rule) appears at column 75 in the Editor/Debugger, providing a cue as to when a line becomes wider than desired, which is useful if you plan to print the file, for example. You can hide the line or change the column number at which it appears — see in the online documentation.

View Function or Subfunction

The function or subfunction the cursor is currently at is shown at the right side of the status bar in the Editor/Debugger.

Code Folding—Expanding and Collapsing M-File Constructs

You can enable or disable the ability to expand and collapse M-file programming constructs; this ability is referred to as code folding. Programming constructs include functions and function help.

Code folding is particularly useful for improving readability when an M-file contains numerous subfunctions and you want to hide code on which you are not currently working.

When you fold a function, all the code associated with that function (including any help code) is collapsed such that the Editor/Debugger displays only the function definition line. The function definition line is appended with an ellipsis icon to indicate there is more function code.

The following image shows the `collatz` function code expanded and the `collatzall` and `collatzplot_new` functions collapsed:
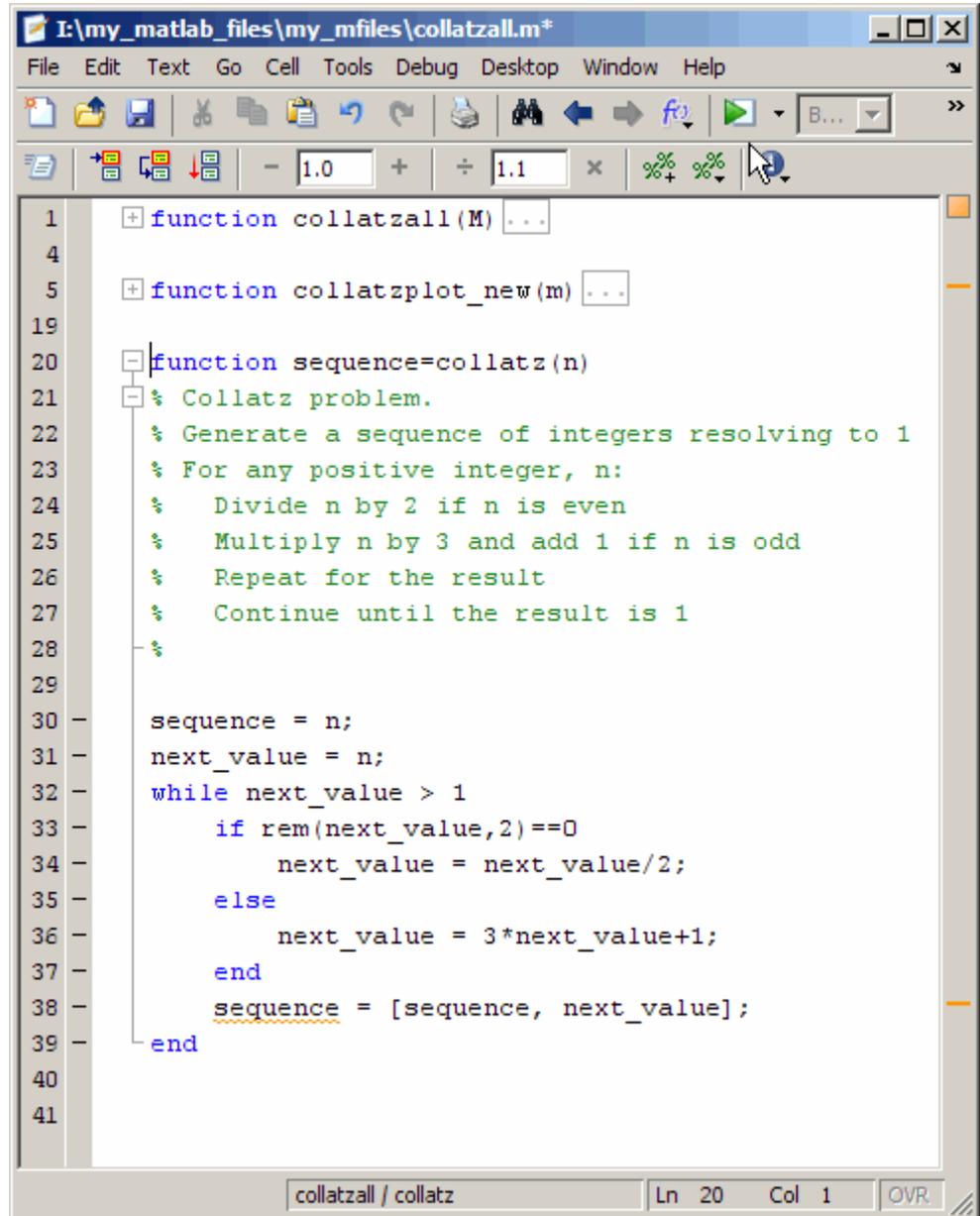
```
 I:\my_matlab_files\my_mfiles\collatzall.m*                           _ □ X
File   Edit   Text   Go   Cell   Tools   Debug   Desktop   Window   Help

 1     ⊞ function collatzall(M) ...
 4
 5     ⊞ function collatzplot_new(m) ...
19
20     ⊟ function sequence=collatz(n)
21     ⊟ % Collatz problem.
22       % Generate a sequence of integers resolving to 1
23       % For any positive integer, n:
24       %    Divide n by 2 if n is even
25       %    Multiply n by 3 and add 1 if n is odd
26       %    Repeat for the result
27       %    Continue until the result is 1
28       %
29
30  -    sequence = n;
31  -    next_value = n;
32  -    while next_value > 1
33  -        if rem(next_value,2)==0
34  -            next_value = next_value/2;
35  -        else
36  -            next_value = 3*next_value+1;
37  -        end
38  -        sequence = [sequence, next_value];
39  -    end
40
41

                    collatzall / collatz          Ln  20    Col  1    OVR
```

When you expand a function, but collapse its associated help code, the Editor/Debugger displays all the function code and just the H1 line of the help code. The H1 line ends with a commented ellipsis icon to indicate there is additional help code, as shown in the following image:
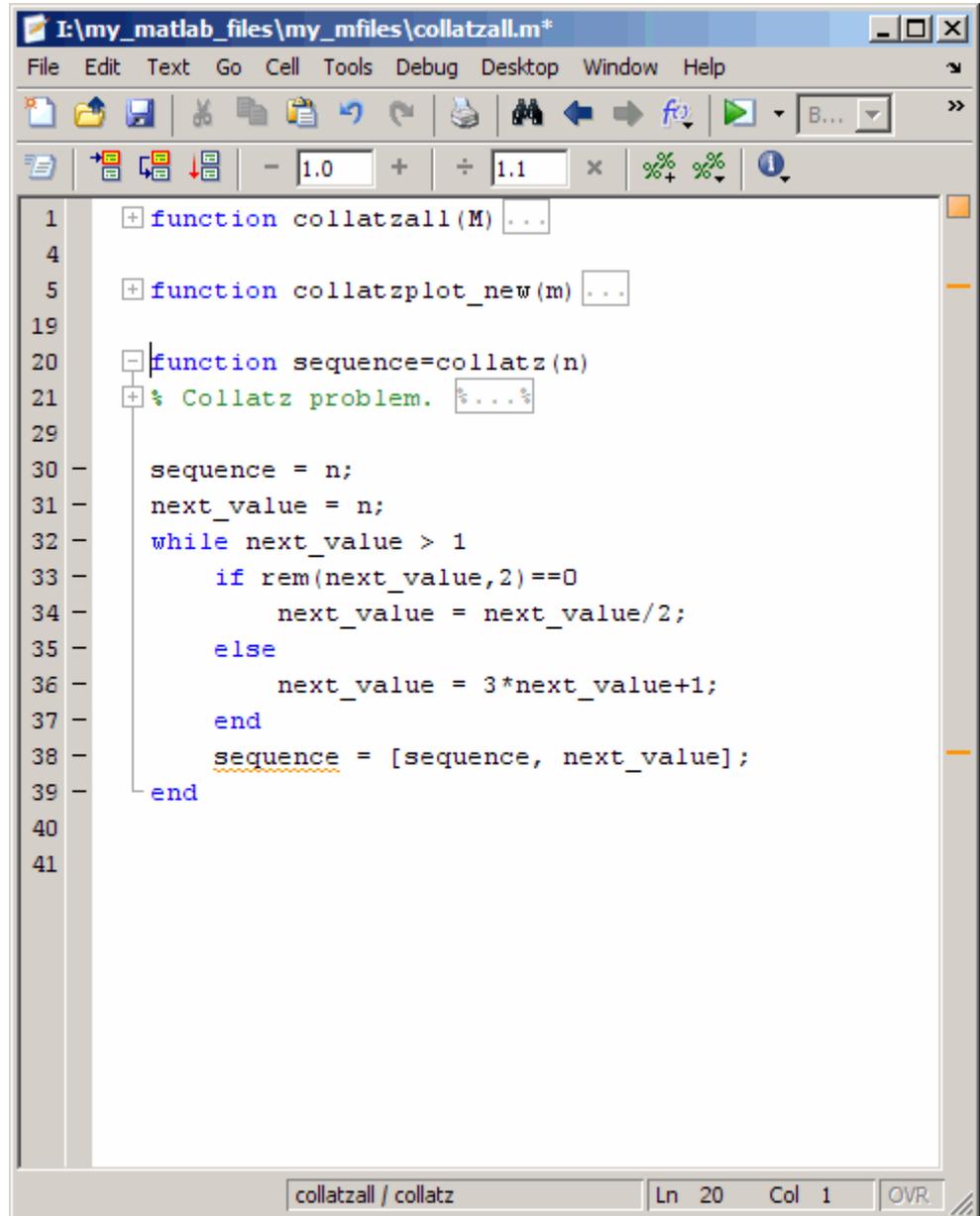
```
1     ⊞ function collatzall(M) [...]
4
5     ⊞ function collatzplot_new(m) [...]
19
20    ⊟ function sequence=collatz(n)
21    ⊞ % Collatz problem. [%...%]
29
30 −      sequence = n;
31 −      next_value = n;
32 −      while next_value > 1
33 −          if rem(next_value,2)==0
34 −              next_value = next_value/2;
35 −          else
36 −              next_value = 3*next_value+1;
37 −          end
38 −          sequence = [sequence, next_value];
39 −      end
40
41
```

To expand code for a construct that is currently collapsed, do one of the following:

- Click the plus sign icon ⊞ to the left of the construct that you want to expand.

- Place your cursor in the code that you want to expand, right-click, and then select **Code Folding > Expand** from the context menu.

To collapse code for a construct that is currently expanded, do one of the following:

- Click the minus sign icon ⊟ to the left of the construct that you want to collapse.

- Place your cursor in the code that you want to collapse, right-click, and then select **Code Folding > Collapse** from the context menu.

To expand or collapse all of the code in an M-file, place your cursor anywhere within the M-file, right-click, and then select **Code Folding > Expand All** or **Code Folding > Collapse All** from the context menu.

For information on the structure of an M-file, including a description of a function definition line and an H1 line, see Basic Parts of an M-File in the MATLAB Programming documentation

Viewing Folded Code in a Tooltip

You can view code that is currently folded by positioning the pointer over its ellipsis icon [⋯]. The code displays in a tooltip. This enables you to get a quick view of the code without requiring you to unfold it.

The following image shows the tooltip that displays when the pointer is placed over the ellipsis icon on line 5 of collatzall.m when the collatzplot_new function is folded.

```
Editor - I:\my_matlab_files\my_mfiles\collatzall.m

File  Edit  Text  Go  Cell  Tools  Debug  Desktop  Window  Help

1      function collatzall(M) ...
4
5      function collatzplot_new(m) ...
19
20     function se
21     % Collatz p
22     % Generate
23     % For any p
24     %    Divide
25     %    Multipl
26     %    Repeat
27     %    Continu
28     %
29
30 -   sequence = n;
31 -   next_value = n;
32 -   while next_value > 1
33 -       if rem(next_value,2)==0
34 -           next_value = next_value/2;
35 -       else
36 -           next_value = 3*next_value+1;
37 -       end
38 -       sequence = [sequence, next_value];
39 -   end
40
```

Tooltip content:

```
function collatzplot_new(m)
% Plot length of sequence for Collatz problem
% Prepare figure
clf
set(gcf,'DoubleBuffer','on')
set(gca,'XScale','linear')
%
% Determine and plot sequence and sequence length
for N = 1:m
        plot_seq = collatz(N);
        seq_length(N) = length(plot_seq);
        line(N,seq_length(N),'Marker','.','MarkerSize',9,'Color','blue')
        drawnow
end
```

```
sine_wave.m  x    collatzplot.m  x   collatzall.m  x

collatzall / collatz          Ln  39    Col  3    OVR
```

Code Folding Behavior and Preferences

Be aware of the following:

- You can change the current code folding settings, by selecting **File > Preferences > Editor/Debugger > Code Folding** and then clicking **Help** for assistance.

- By default, the first time you open an existing M-file in MATLAB release 7.5 (R2007b), code folding is enabled and all constructs are expanded.

- Constructs that are collapsed when you close an M-file remain collapsed when you reopen the file.

- If you copy a collapsed construct from one region of an M-file and paste it in another region, the construct is expanded in the pasted location.

- If you print a file with one or more collapsed constructs, those constructs are expanded in the printed version of the file.

Split Screen Display

You can simultaneously display two different parts of a file in the Editor/Debugger. This makes it easy to compare different lines in a file or to copy and paste from one part of a file to another.

Split the screen horizontally by selecting **Window > Split Screen > Top/Bottom**. Or to split it vertically, select **Left/Right**.

Alternatively, when there is a scroll bar, split the document into top and bottom views by dragging the splitter bar, (as shown in the following illustration), down from above the vertical scroll bar. Similarly, to split into left and right views, drag the splitter bar from the left of the horizontal scroll bar. The pointer assumes a double-headed arrow shape when it is positioned on the splitter bar.

Drag splitter bar
down to create top
and bottom views.

Split documents vertically or horizontally.

Drag splitter bar
right to create left
and right views.
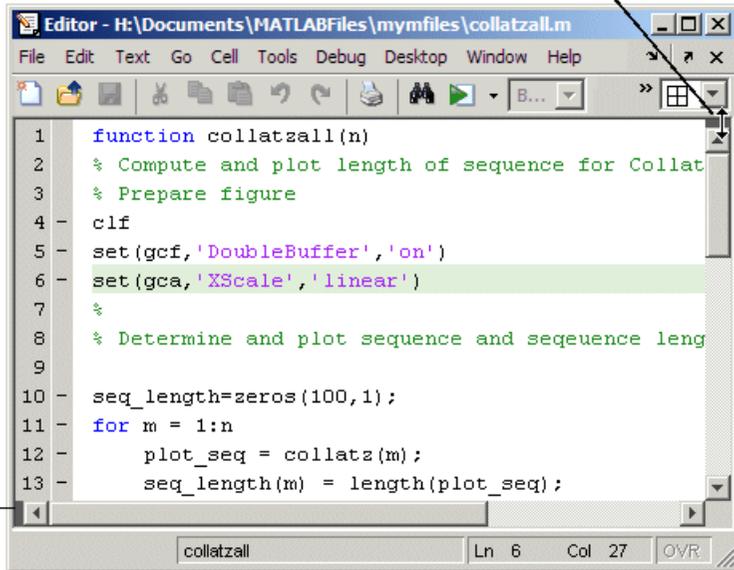
Top/bottom
split.

Double-click the
splitter
to remove the
split. Drag the
splitter to resize
the views.
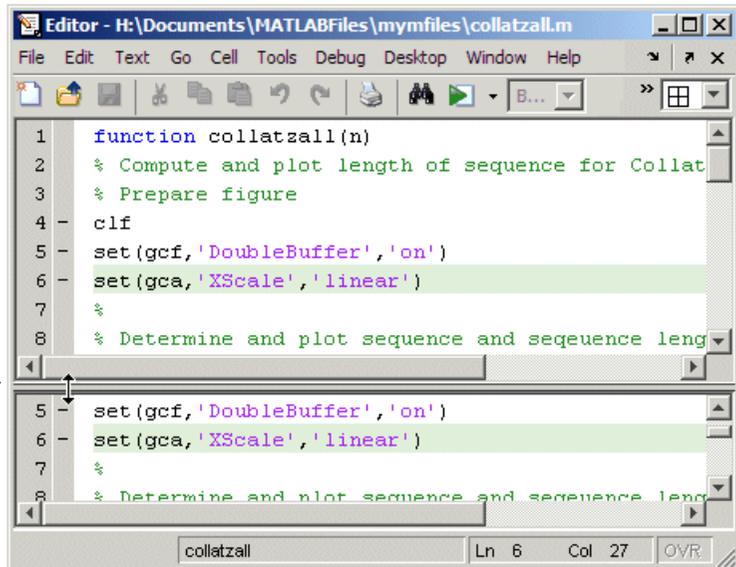
Left/right split.   Double-click the splitter to remove the split.
                                     Drag the splitter to resize the views.
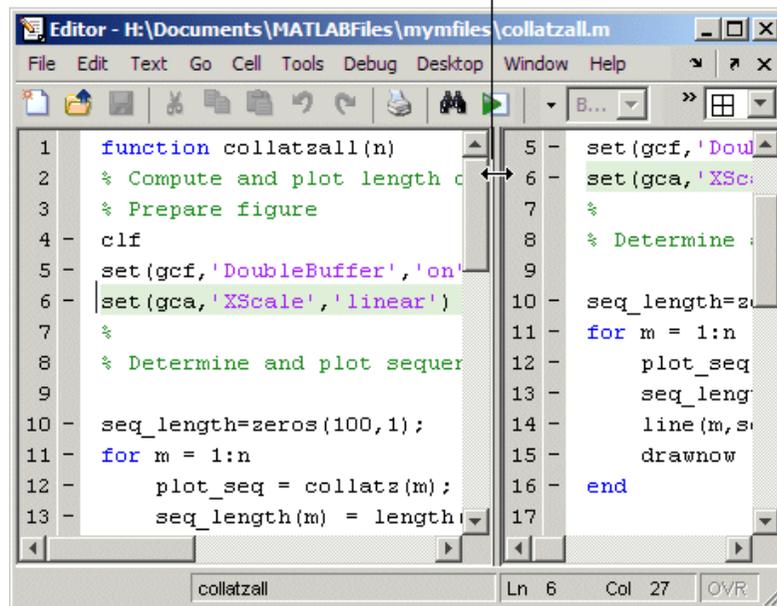
Adjust the size of the views by dragging the splitter. The pointer assumes an arrow shape when it is positioned on the splitter.

Only one view is active at any time, meaning, you will only see the cursor in one of the views. To change the active view use **Window > Split Screen > Switch Focus** or its keyboard equivalent, which is shown with the menu item. The cursor returns to its last position in that view.

Make changes to the document in either view. Both views of the file are always current, so you see the changes in either view.

You split each open document individually, so there can be multiple configurations at once. You can split some documents horizontally, others vertically, and leave others unsplit. When you open a document, it always opens unsplit, regardless of its split status when you last had it open.

You can remove a document split using any of these methods:

- Drag the splitter to an edge of the window.

- Double-click the splitter.

- Select **Window > Split > Screen > Off**.

See also "Summary of Actions for Arranging Documents" on page 2-11 for instructions to display multiple documents simultaneously.

Navigating in an M-File

Going to a Line Number

Select **Go > Go To**. In the resulting **Go To** dialog box, select the **Line number** option, enter a line number, and click **OK**. The cursor moves to that line number in the current M-file.

Going to a Function (Subfunctions and Nested Functions)

To go to a function within an M-file (either a subfunction or a nested function), select **Go > Go To**. In the resulting **Go To** dialog box, select the **Function** option, and then select an entry from the list of subfunctions and nested functions in the file. Click **OK**.
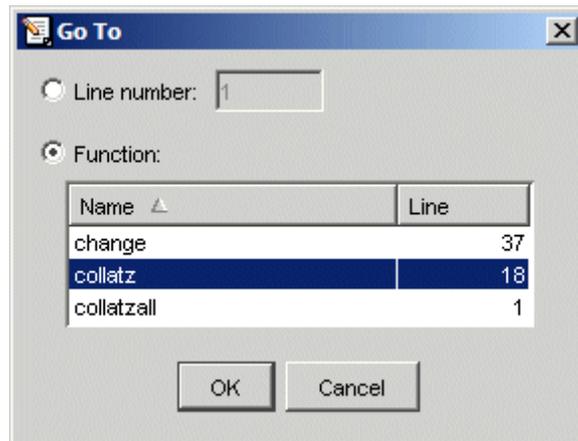
Functions in the list appear alphabetically by name. To order them by their position in the file, click the **Line** column heading. The list does not include functions that are called from the M-file, but only shows lines in the current M-file that begin with a `function` statement.

Alternatively, click the Show Functions button on the toolbar. Then select the subfunction or nested function you want to go to from the list. The functions are listed in order of position in the file.

Note that the status bar shows the function and subfunction the current line is part of.

Going to a Cell

For M-file scripts that contain cells for rapid code iteration or publishing, the **Go To** dialog box lists cell titles.

Going to a Bookmark

You can set a bookmark at a line in a file in the Editor/Debugger so you can quickly go to the bookmarked line. This is particularly useful in long files. For example, while working on a line, if you need to look at another part of the file and then return, set a bookmark at the current line, go to the other part of the file, and then go back to the bookmark.

To set a bookmark, position the cursor anywhere in the line and select **Go > Set/Clear Bookmark**. A bookmark icon appears to the left of the line.

```
   11 - while next_value > 1
```

To go to a bookmark, select **Next Bookmark** or **Previous Bookmark** from the **Go** menu.

To clear a bookmark, position the cursor anywhere in the line and select **Go > Select/Clear Bookmark**.

Bookmarks are not maintained after you close a file.

Navigating Back and Forward in Files

Use **Go > Back** (and **Go > Forward**) to go to lines you previously edited or navigated to in a file. The feature goes to the lines in the sequence you accessed them. As an alternative to the menu items, use the Back and Forward buttons on the toolbar.

Use Back and Forward buttons or menu items to navigate
to lines you previously edited or navigated to.
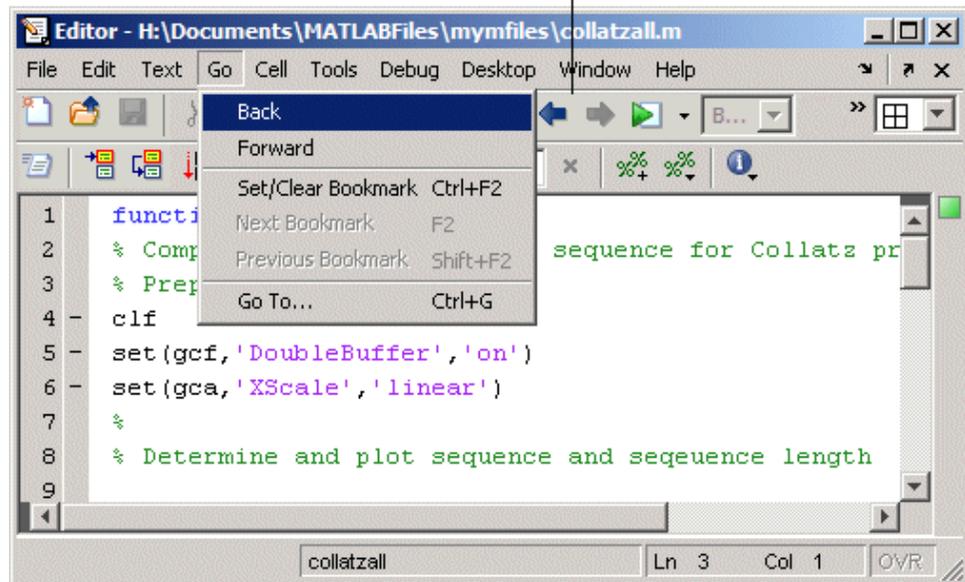
For example, if you open a file and make changes at lines 3, 9, and 6, use
Go > Back to return to line 9, then 3, then 1, and then use **Go > Forward**
to go from 1 to 3 to 9 to 6, and then return to 3. Detailed instructions to
accomplish this are

1 Select **Go > Back** to return from line 6 to line 9.

2 Select **Go > Back** again to return to line 3.

3 Select **Go > Back** again to return to line 1, which is the first line you
originally navigate to in a file by virtue of opening it.

4 Use **Go > Forward** to reverse the direction of the feature—select
Go > Forward to navigate to line 3.

5 Select **Go > Forward** to navigate to line 9.

6 Reverse the direction of the feature again—select **Go > Back** to navigate
to line 3.

Lines Navigated to Using Go Back

Use **Go > Back** and **Forward** to go to lines you previously edited or navigated to via these features:

Feature	Examples	Notes
Opening a file (first line in the file)	**File > Open**	None
Changes made using text-editing tools	**Delete** key, or **Text > Increase Indent**	Edits made to a selection of lines are represented by the first line in the selection. Changes made using **Cell > Insert Cell Divider** and **Cell > Insert Text Markup** are not considered as having been previously navigated to.
Changes made using Find and Replace	**Edit > Find and Replace**	Changes made using **Replace All** are not considered as having been previously navigated to.
Find features	**Edit > Find and Replace**, **Find Next**, **Find Previous**, and **Find Selection**	None
Incremental search	**Ctrl+S** and **Ctrl+R**	None
Show Function button	$f_\downarrow$	None
Opening a selection	**File > Open Selection**	None
Go to	**Go > Go To** line number, function, or cell title	None
Bookmark navigation	**Go > Next Bookmark** and **Previous Bookmark**	A line at which you **Set/Clear Bookmark** is not considered as having been previously navigated to.
Hyperlink access	From warnings or errors in the Command Window, from **Find Files** results, and from reports like the Profiler	None

Feature	Examples	Notes
Debugging navigation	Lines with breakpoints that were stopped at while running, and lines stepped to	A line at which you set a breakpoint is not considered as having been previously navigated to, unless it was actually stopped at during execution.
Cell mode navigation	**Cell > Next Cell** and **Previous Cell**, and **Cell > Evaluate Current Cell and Advance**	Lines accessed using **Cell > Evaluate Current Cell** are not considered as having been previously navigated to.

Interrupting the Sequence of Go Back and Forward

If you use **Go > Back** and **Go > Forward**, and then edit another line or navigate to another line using the list of features described in the above table, the **Go > Back** or **Go > Forward** feature sequence is interrupted. You can still go to the lines preceding the interruption point in the sequence, but you cannot go to any lines after that point. Any lines you edit or navigate to after interrupting the sequence are added to the sequence after the interruption point.

For example,

1 Open a file and edit lines 2, then 4, and then 6.

2 Use **Go > Back** to move back to line 4, and then back to line 2.

3 You could then **Go > Forward** to lines 4 and 6, or **Go > Back** to line 1.

   Instead, make an edit at line 3. Now you cannot **Go > Forward** to lines 4 and 6 and you can only **Go > Back** to line 2 and then line 1.

Closed Files and Behavior of Go Back and Forward

Go > Back and **Forward** do not go to lines in closed files.

Split Screen and Behavior of Go Back and Forward

When you have a split screen display, **Go > Back** and **Forward** go to the view in which the line was originally navigated to or edited in. If you remove

the split, **Go > Back** and **Forward** do not go to any lines that were visited in the lower (or right) view.

Opening a Selection in an M-File

You can open a subfunction, function, file, variable, or Simulink model from within a file in the Editor/Debugger. Position the cursor in the name and then right-click and select **Open Selection** from the context menu. Based on what the selection is, the Editor/Debugger performs a different action.

Selection	Action
Subfunction	Cursor moves to the subfunction within the current M-file. If no subfunction by that name is found in the current M-file, the Editor/Debugger runs the open function on the selection, which opens the selection in the appropriate tool, as shown for the other selection types in this table.
M-file or other text file	Opens in the Editor/Debugger.
Figure file (`.fig`)	Opens in a figure window.
Variable	Opens in the Array Editor.
Model	Opens in Simulink.
Other	If the selection is some other type, **Open selection** looks for a matching file in a private directory in the current directory and performs the appropriate action.

After selecting a name, you can also choose **Help on Selection** from the context menu to see documentation for the item. For example, select a function, right-click and select **Help on Selection**. The reference page for that function opens in a popup window, or if the reference page does not exist, the M-file help appears. For more information, see "Getting Pop-Up Help for Functions" on page 4-49

Finding Text in Files

In this section...

Finding Text in the Current File

Within the current file, select the text you want to find. From the **Edit** menu, select **Find Selection**. The next occurrence of that text is selected. Select **Find Selection** again (or **Find Next**) to continue finding more occurrences of the text.

To find the previous occurrence of selected text (find backwards) in the current file, select **Find Previous** from the **Edit** menu. The previous occurrence of the text is selected. Repeat to continue finding the previous occurrences of the text.

Finding and Replacing Text in the Current File

You can search for specified text within multiple files, and then replace the text within a file.

Finding Text

To search for text in files, click the Find button 🔍 in the Editor/Debugger toolbar, or select **Edit > Find and Replace**. Complete the resulting **Find & Replace** dialog box.
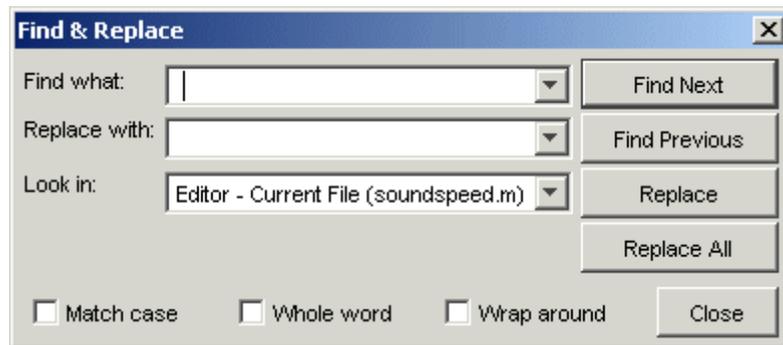
The search begins at the current cursor position. MATLAB finds the text you specified and highlights it. To find another occurrence, click **Find Next** or **Find Previous**, or use the keyboard shortcuts **F3** and **Shift+F3** (or **Command+F3** and **Command+Shift+F3** with Macintosh key bindings).

MATLAB beeps when a search for **Find Next** reaches the end of the file, or when a search for **Find Previous** reaches the top of the file. If you have **Wrap around** selected, it continues searching after beeping.

Use **F3** and **Shift+F3** to continue finding the specified text even after closing the **Find & Replace** dialog box. You can go to another file and find the specified text in it.

Change the selection in the **Look in** field to search for the specified text in other MATLAB desktop tools.

Replacing Text
After finding text using the **Find & Replace** dialog box, you can replace the text in the current file:

1 In the **Replace with** field, type the text that is to replace the found text.

2 Click **Replace** to replace the text currently selected, or click **Replace All** to replace all instances in the current file.

   The text is replaced. For **Replace All**, the number of instances that were replaced appears in the Editor/Debugger status bar.

3 To save the changes to the file, select **Save** from the **File** menu.

You can repeat this for multiple files.

Function Alternative for Finding Text

Use `lookfor` to search for the specified text in the first line of help for all M-files on the search path.

Finding Files or Text in Multiple Files

To find directories and filenames that include specified text, or whose contents contain specified text, use **Edit > Find Files**. For details, see "Finding Files and Content Within Files" on page 5-49.

Incremental Search

With the incremental search feature, the cursor moves to the next or previous occurrence of the specified text in the current file. It is similar to the Emacs search feature. Incremental search is also available in the Command Window — see "Incremental Search" on page 3-35.

To use the incremental search feature in the Editor/Debugger, follow these steps:

1 Position the cursor where you want the search to begin.

2 How you begin the incremental search depends on your setting for theEditor/Debugger key bindings preference and in which direction you want to search:

- Press **Ctrl+S** to search forward or **Ctrl+R** to search backward for Emacs and Macintosh key bindings.

- Press **Ctrl+Shift+S** to search forward or **Ctrl+Shift+R** to search backward for Windows key bindings.

An incremental search field appears in the left side of the status bar of the current file window. **F Inc Search** means search *F*orward from the cursor. The field label is instead **R Inc Search** when you search backwards.

Incremental search field. **F** means search *Forward* from the cursor.

3 In the incremental search field, type the text you want to find. For example, type `plot`.

As you type the first letter, `p`, the first occurrence of that letter after the cursor is highlighted. In the example shown, the cursor is in the middle of line 2, so the first occurrence of `p`, the `p` in `problem` on line 2, is highlighted.

Incremental search is case sensitive for uppercase letters. In the above example, searching for uppercase `P`, would instead find the `P` in `Prepare` on line 3.

When you type the next letter in the term you are searching for, the first occurrence of the term becomes highlighted. In the example, when you add the letter `l` to the `p` so that the incremental search field now has `pl`, the `pl` in `plot` on line 8 is highlighted. When you add `ot` to the term in the incremental search field, the whole word `plot` in line 8 is highlighted.

- If you mistype in the incremental search field, use the backspace key to remove the last letters and make corrections.

- After finding the p, press **Ctrl+W** to highlight the rest of the word found, in this case plot, which also puts the complete word in incremental search field.

4 To find the next occurrence of plot in the file, press **Ctrl+S**. To find the previous occurrence of the text, press **Ctrl+R**.

5 If MATLAB beeps, it either means the search is at the end or beginning of the file, or it means that the text was not found.

- When the text is not found, Failing appears in the incremental search field. Modify the search term in the incremental search field and try again. Use **Ctrl+G** to automatically remove characters back to the last successful search. For example, if plode fails, **Ctrl+G** removes the de from the search term because plo does exist in the file.

- When at the end or beginning of the file, press **Ctrl+S** or **Ctrl+R** again to wrap to the beginning (or end) of the file and continue the search. Use **Ctrl+G** after a finding a string to clear the search and return the cursor to the starting point.

6 To end the incremental search, press **Esc** or **Enter**, or any other noncharacter or number key except **Tab** or backspace.

The incremental search field no longer appears in the status bar. The cursor is now located at the position where the string was last found.

If you press **Ctrl+S** or **Ctrl+R** after displaying the blank incremental search field, the search term from your previous incremental search appears in the field. Then the backspace key deletes the entire previous search term, rather than just the last letter.

Comparing Files — File Comparison Tool

In this section...

What Is the File Comparison Tool?

The File Comparisons tool identifies differences line by line between two files. Some other applications refer to this as a file diff. As an example, you can use this to easily compare an autosaved version of a file to the latest version.

Running the File Comparison Tool

To use the tool, follow these steps:

1 Open one of the files you want to compare in the Editor/Debugger.

To open the example file provided, lengthofline.m, run

```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...
  'examples','lengthofline.m'))
```

2 Select **Tools > Compare Against > Browse**. Navigate to the file you want to compare against, select the file, and click **Open**. To open the example file provided, select lengthofline2.m. Other options available are

- **Tools > Compare Against > Autosave Version** to compare the open file to the Editor/Debugger's automatic copy, *filename*`.asv`. For more information, see "Autosave" on page 6-63.

- **Tools > Compare Against Version on Disk** to compare an open file that has been changed but not saved to the saved version.

3 The File Comparisons tool opens, displaying the files side by side and highlighting lines that do not match. Pink highlighting and an x at the start of a line indicates that the content of the lines differs between the two files. Green highlighting and a > at the start of a line indicates a line that exists in the file presented on the right side of the page but not in the file presented on the left side of the page. Green highlighting and a < at the end of a line indicates a line that exists in the file presented on the left side of the page but not in the file presented on the right side of the page.
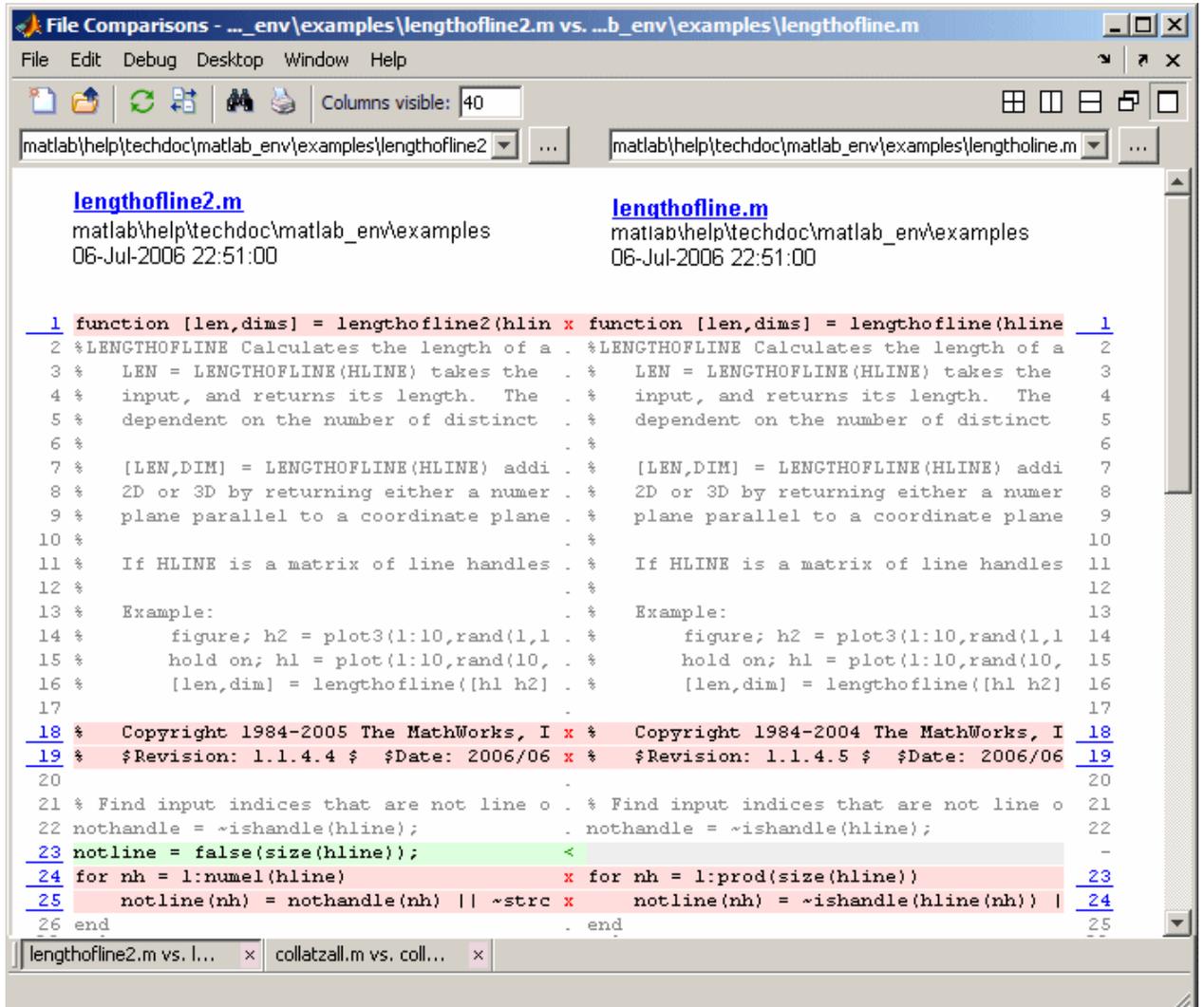
4 Use features of the File Comparisons tool to work with the results.

Increase or Decrease Line Lengths Shown

By default, the display is 60 columns wide for each file. To see the full line length for each file, use a high number for **Columns visible**, and drag the

vertical edges of the window to make it wider. With a narrower window size, if there are more columns shown in the left file but fewer in the right file, reduce the number for **Columns visible** to see enough in both sides.

Exchange Positions

To move the file on the left side to the right side and vice-versa, select **File > Swap Sides**, or click the equivalent toolbar button .

Show Updated Files

After making changes to and saving the files in the Editor/Debugger, update the results in the File Comparisons tool by selecting **File > Refresh** or clicking the equivalent toolbar button .

Find Text in Files

To find a phrase in the currently displayed files, select **Edit > Find**, or click the equivalent toolbar button. The resulting Find dialog box is the same as the one you use in the Command Window—for more information, see "Find Dialog Box" on page 3-34.

Compare to Other Files

You can replace an existing file in the tool. Drag a different filename from the Current Directory browser or Windows Explorer to the left or right side of the File Comparisons window, replacing the file currently shown there. Alternatively, type the path to a file or browse to find a file using the field below the File Comparisons toolbar. Another option is to use **File > Open**.

Perform New and View Previous Comparisons

You can perform another comparison by selecting **File > New File Comparison**. Supply the files to compare using the techniques described in "Compare to Other Files" on page 6-57. You can see the results of either pair of comparisons by selecting its entry in the document bar (as shown at the bottom of the window in the above example illustration).

Alternative Ways to Access the Tool

These are additional ways you can access the tool:

- From the MATLAB desktop, select **Desktop > File Comparisons**.

- From the Current Directory browser, select a file, right-click, and from the context menu, select **Compare Against**.

- For two files in the same directory, from the Current Directory browser, select the files, right-click, and from the context menu, select **Compare Selected Files**.

Supply the files to compare using the techniques described in "Compare to Other Files" on page 6-57.

Keyboard Shortcuts in the Editor/Debugger

Following is the list of keys that serve as shortcuts for using the Editor/Debugger. This list does not include shortcut keys (sometimes called hot keys) for menu items—you can view those on the menus. If you select the **Emacs** "Editor/Debugger Key Bindings" on page 3-45 preference, you can also use the **Ctrl+key** combinations shown. See also general desktop "Keyboard Shortcuts" on page 2-40.

Key or Mouse Action for Windows Preference	Additional Control Key for Emacs Preference	Key or Mouse Action for Macintosh Preference	Operation
↑	**Ctrl+P**	↑	Move to *p*revious line.
↓	**Ctrl+N**	↓	Move to *n*ext line.
Ctrl+Home	None	**Cmd+Home**	Move to top of file.
Ctrl+End	None	**Cmd+End**	Move to end of file.
Ctrl+↑	None	**Home**	Scroll up without moving cursor position (with cell mode disabled). Move to top of current cell or top of previous cell (with cell mode enabled).
Ctrl+↓	None	**End**	Scroll down without moving cursor position (with cell mode disabled). Move to top of next cell (with cell mode enabled).
Page Down	**Ctrl+V**	**Page Down**	Move down one screen.
Page Up	**Alt+V**	**Page Up**	Move up one screen.
←	**Ctrl+B**	←	Move *b*ack one character.
→	**Ctrl+F**	→	Move *f*orward one character.
Ctrl+←	None	**Option+**←	Move *l*eft one word.
Ctrl+→	None	**Option+**→	Move *r*ight one word.

Key or Mouse Action for Windows Preference	Additional Control Key for Emacs Preference	Key or Mouse Action for Macintosh Preference	Operation
Home	**Ctrl+A**	**Cmd+←**	Move to beginning of line.
End	**Ctrl+E**	**Cmd+→**	Move to *e*nd of line.
Delete	**Ctrl+D**	**Forward Delete**	Delete character after cursor.
Backspace	none	**Delete**	Delete character before cursor.
None	**Ctrl+K**	None	Cut contents (*k*ill) to end of line.
Double-click	None	Double-click	Select current word. To select additional words, hold mouse after second click and continue dragging left or right.
Triple-click	None	Triple-click	Select current line. To select additional lines, hold mouse after second click and continue dragging up or down.
Ctrl+Shift+←	None	**Option+Shift+←**	Select word to the left
Ctrl+Shift+→	None	**Option+Shift+→**	Select word to the right.
Shift+Home	None	**Cmd+Shift+←**	Select to beginning of line.
Shift+End	None	**Cmd+Shift+→**	Select to end of line.
Shift+Page Up	**Ctrl+Shift+V**	**Shift+Page Up**	Select one screen up.
Shift+Page Down	**Alt+Shift+V**	**Shift+Page Down**	Select one screen down.
Ctrl+Shift+Home	None	**Cmd+Shift+Home**	Select to top of file.
Ctrl+Shift+End	None	**Cmd+Shift+End**	Select to end of file.
Shift+Enter	None	**Shift+Enter**	Add a new line that is not indented.

Key or Mouse Action for Windows Preference	Additional Control Key for Emacs Preference	Key or Mouse Action for Macintosh Preference	Operation
Insert	None	None	Change to overwrite mode from insert mode, or change to insert mode from overwrite mode. View current mode in the status bar: OVR is gray for insert mode. In overwrite mode, what you type replaces existing text, and the cursor is a wide block. (Not supported on Macintosh platforms.)
Shift+F5	None	**Shift+F5**	Exit debug mode. Equivalent to typing dbquit. The Command Window displays the standard prompt >>.

Saving, Printing, and Closing Files in the Editor/Debugger

In this section...
"Saving Files" on page 6-62
"Printing M-Files" on page 6-64
"Closing M-Files" on page 6-64

Saving Files

After making changes to an file, you see an asterisk (*) next to the filename in the title bar of the Editor/Debugger. This indicates there are unsaved changes to the file.

To save the changes, use one of the **Save** commands in the **File** menu:

- **Save** — Saves the file using its existing name. If the file is newly created, the **Save file as** dialog box opens, where you assign a name to the file before saving it. Another way to save is by using the Save button ⊟ on the toolbar. If the file has not been changed, **Save** is grayed out, but you can instead use **Save As** from the **File** menu to save to a different filename.

- **Save As** — The **Save file as** dialog box opens, where you assign a name to the file and save it. By default, if you do not type an extension, MATLAB automatically assigns the `.m` extension to the filename. If you do not want an extension, type a `.` (period) after the filename.

- **Save All** — Saves all named files to their existing filenames. For all newly created files, the **Save file as** dialog box opens, where you assign a name to each untitled file and save it.

You cannot save an M-file while in debug mode. If you try to, MATLAB displays a dialog box asking if you want to exit debug mode and then save the file. While debugging, you can execute sections of an M-file even though there are unsaved changes — see "Running Sections in M-Files That Have Unsaved Changes" on page 6-125.

Note Save any M-files you create and any M-files from The MathWorks that you edit in a directory that is not in the *matlabroot*/toolbox directory tree. If you keep your files in *matlabroot*/toolbox directories, they can be overwritten when you install a new version of MATLAB. Also note that locations of files in the *matlabroot*/toolbox directory tree are loaded and cached in memory at the beginning of each MATLAB session to improve performance. If you save files to *matlabroot*/toolbox directories using an external editor or add or remove files from these directories using file system operations, run rehash toolbox before you use the files in the current session. If you make changes to existing files in *matlabroot*/toolbox directories using an external editor, run clear functionname before you use the files in the current session. For more information, see rehash or "Toolbox Path Caching in MATLAB" on page 1-17.

Autosave

As you make changes to a file in the Editor/Debugger, every five minutes the Editor/Debugger automatically saves a copy of the file to a file of the same name but with an .asv extension. The autosave copy is useful if you have system problems and lose changes made to your file. In that event, you can open the autosave version, filename.asv, and then save it as filename.m to use the last good version of filename. For example, if you edit filename.m and do not save it for five minutes, MATLAB saves the file including the unsaved changes, to filename.asv.

Use autosave preferences to turn the autosave feature off or on, to specify the number of minutes between automatic saves, and to specify the file extension and location for autosave files. For details, see in the online documentation.

If the file you are editing is in a read-only directory and the autosave preference for location is the source file directory, an autosave copy of the file is *not* made.

Deleting Autosave Files. By default, autosave files are not automatically deleted when you delete the source file. To keep autosave to M-file relationships clear and current, it is a good practice when you rename or remove an M-file to delete or rename its corresponding autosave file.

There is a preference to **Automatically delete autosave files**. With this preference selected, when you close an M-file in the Editor/Debugger, MATLAB automatically deletes the corresponding autosave file.

Accessing Your Source Control System

If you use a source control system for M-files, you can access it from within the Editor/Debugger using **File > Source Control**. For more information, see Chapter 9, "Source Control Interface".

Printing M-Files

To print an entire M-file, select **File > Print**, or click the Print button 🖨 on the toolbar. To print the current selection, select **File > Print Selection**. Complete the standard print dialog box that appears.

Specify printing options for the Editor/Debugger by selecting **File > Page Setup**. For example, you can specify printing with a header. For more information, see "Printing and Page Setup Options for Desktop Tools" on page 2-52.

Closing M-Files

To close the current M-file, select **Close filename** from the **File** menu, or click the Close box in the Editor/Debugger menu bar. This is different from the Close box in the titlebar of the Editor/Debugger, which closes all open files in that Editor/Debugger window.

Close box for Editor/Debugger. Closes all open files in this Editor/Debugger window.

Close box for current file.

To close all files within the Editor/Debugger, select **Window > Close Editor Documents**. This does not close any files undocked from the Editor/Debugger. The Editor/Debugger remains open with no files in it.

If each file is open in a separate window, close all the files at once using the **Close All Documents** item in the **Window** menu. Note that this also closes desktop documents of all types, including Array Editor documents.

When you close a file that has unsaved changes, you are prompted to save the file. If you do not want to be prompted, hold **Ctrl** and click the Close box. The prompt will not appear and the document will close without saving any unsaved changes.

Running M-Files in the Editor/Debugger

Running M-Files with No Input Arguments in the Editor/Debugger

In the Editor/Debugger, to run a script M-file, or a function M-file that requires no input arguments, click the Run button on the toolbar. The button's tooltip includes the name of the file to be run, which is useful when you have more than one file open. Alternatively, select **Debug > Run filename**.

If the file is not in a directory on the search path or in the current directory, a dialog box appears, presenting you with options that allow you to run the file. You can either change the current directory to the directory containing the file, or you can add the directory containing the file to the search path.

If the file has unsaved changes, running it from the Editor/Debugger automatically saves the changes before running. In that event, the menu item is **Save File and Run filename**.

If the M-file is a script, you can view the value of a variable in the file, which is called a *datatip* (like a tooltip for data). You need to set the preference to show datatips in edit mode — select **File > Preferences > Display**, and for

General Display Options, select the check box for **Enable datatips in edit mode**.

Using Configurations — Running M-Files with Input Arguments in the Editor/Debugger

In the Editor/Debugger, you can provide values for a function's input arguments using a configuration, and then run that configuration to use the assigned values. When you are editing a function M-file, use a configuration as an alternative to running the function in the Command Window. You can associate multiple configurations with an M-file, for different input values. MATLAB saves the configurations between sessions.

For example, the function `collatzplot_new.m`, which computes and plots the Collatz sequence for any given positive integer, requires you to specify the integer as an input value. You cannot simply run `collatplot_new.m` in the Editor/Debugger because the input value is not defined. One way to specify the input value is to run the M-file in the Command Window. Configurations allow you to run `collatzplot_new(`*specific value*`)` in the Editor/Debugger.

You can also use configurations to provide preparatory or setup information prior to running an M-file, whether it takes input arguments or not.

Create and Run a Configuration for an M-file

Follow these steps to create and run a configuration for an M-file in the Editor/Debugger. These steps specify Editor/Debugger toolbar buttons, but you can also use equivalent items in the **Debug** menu.

1 Open the file you want to run in the Editor/Debugger. For example, open `collatzplot_new.m` by running

```
cd ([matlabroot '/help/techdoc/matlab_env/examples'])
edit collatzplot_new.m
```

To work with `collatzplot_new.m` on your system, save the file to a directory for which you have write permission. In the example, the file is saved to `I:\my_matlab_files\my_mfiles\collatzplot_new.m`

2 Click the down arrow on the Run button in the Editor/Debugger toolbar

Select and run configuration and select **Edit Configurations for** ***filename***, where filename in this example is collatzplot_new.m.
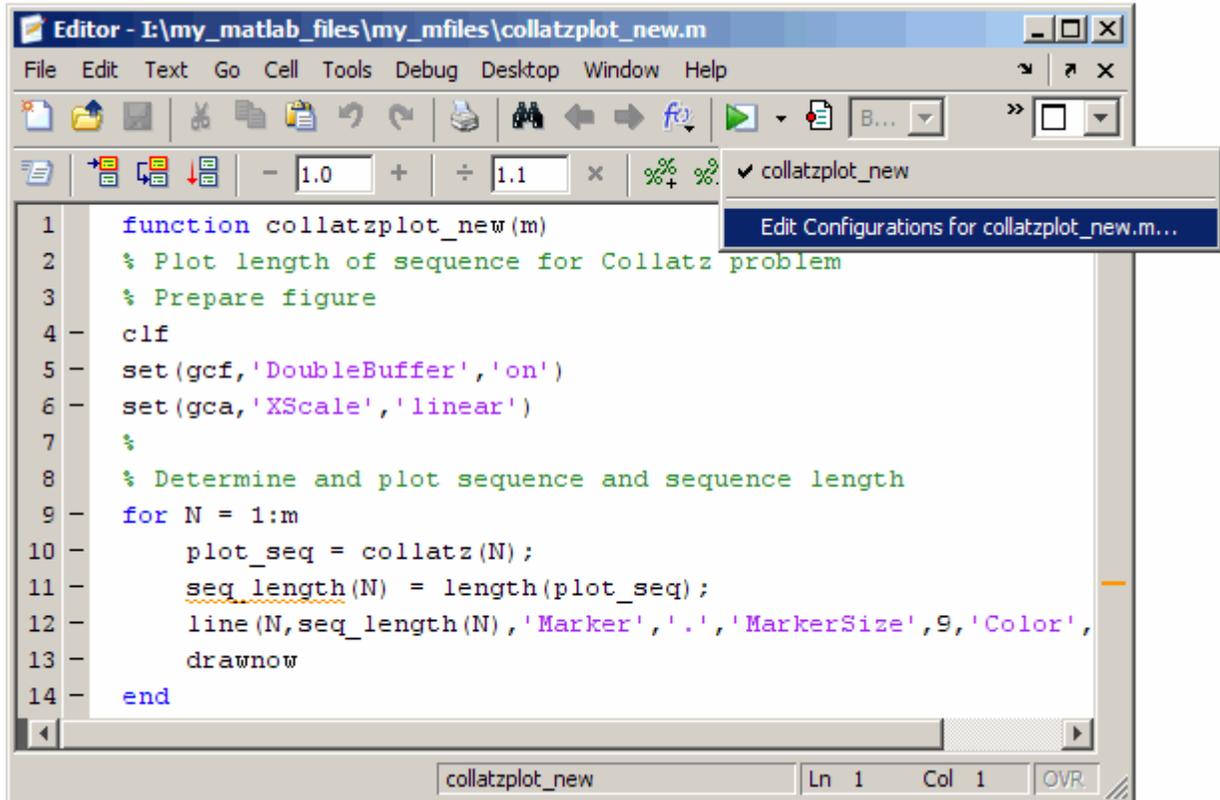
The Edit M-File Configurations dialog box opens, with a default configuration template for collatzplot_new.m.
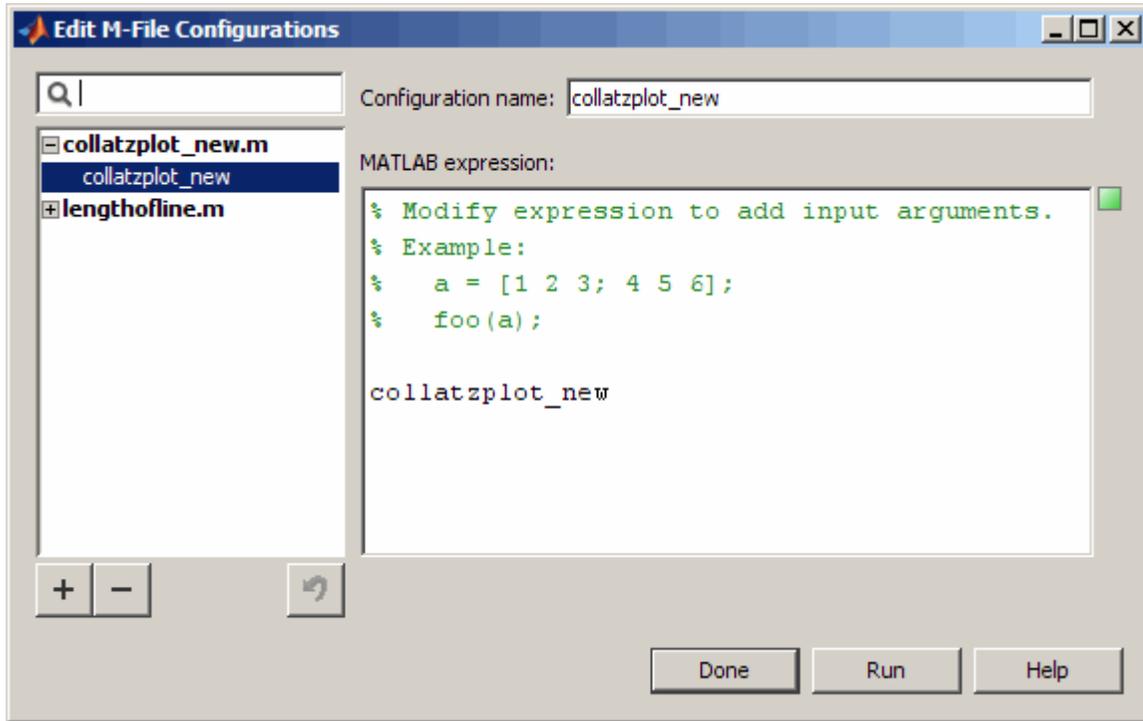
3 In the dialog box, enter MATLAB statements in the **MATLAB expression** area of the dialog box, specifying what you want to run. Delete the existing comments or replace them with comments relevant to your configuration. To undo and redo, use the keyboard shortcuts for your platform, such as **Ctrl+Z** and **Ctrl+Y** for Windows.

In this example, set m equal to 3, a small value useful for debugging purposes. Complete the statement to run collatzplot_new(m).

```
MATLAB expression:
% For debugging purposes
m=3;
collatzplot_new(m)
```

The MATLAB expression area provides syntax highlighting and shows M-Lint messages, similar to the Editor/Debugger.

4 To ensure your configuration runs as expected, click **Run** to execute the statements in the **MATLAB expression** area of the dialog box. In this example, collatzplot_new(3) runs, and a figure window displays the plot.
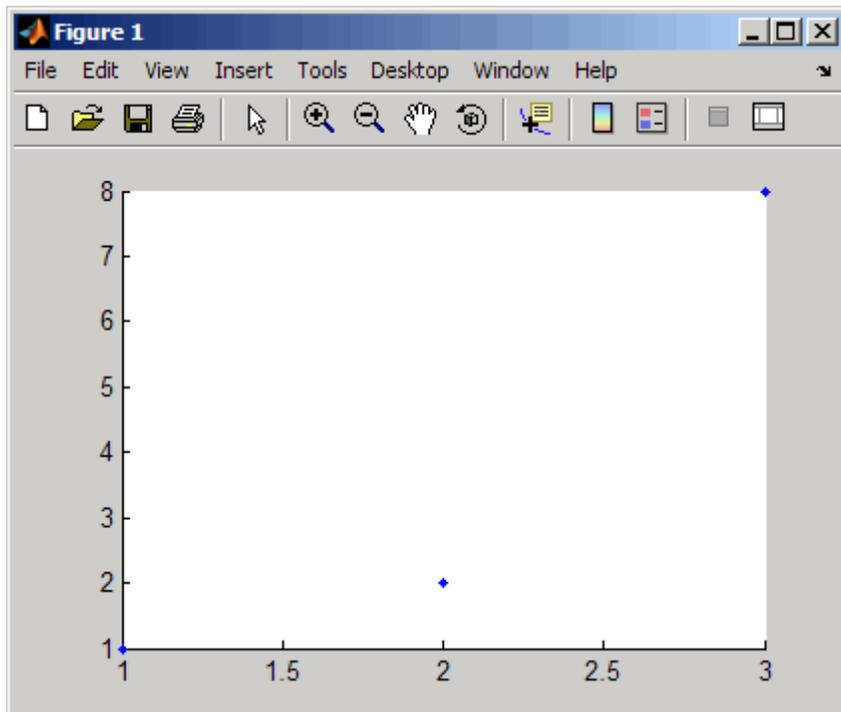
5 You can modify the statements in the **MATLAB expression** area of the dialog box and click **Run** to see the results of the changes. You can also modify the M-file and save the changes while the Edit M-File Configurations dialog box is open, and then click **Run** to see the results of the M-file changes.

6 You can assign a name using **Configuration name** in the Edit M-File Configurations dialog box. By default, the configuration name is the M-file name. If you expect to create multiple configurations for an M-file, assign each a name that helps you identify the configuration. In this example, name the configuration collatzplot_new_test.

7 To close the Edit M-File Configurations dialog box, click **Done**. MATLAB saves the configuration and its association with the M-file.

8 After creating a configuration, you can view and run the configuration without opening the Edit M-File Configurations dialog box.

In the Editor/Debugger toolbar, click the down arrow on the Run button and position the pointer on a configuration name. MATLAB displays a tooltip showing the configuration's expression so you can see what will run.
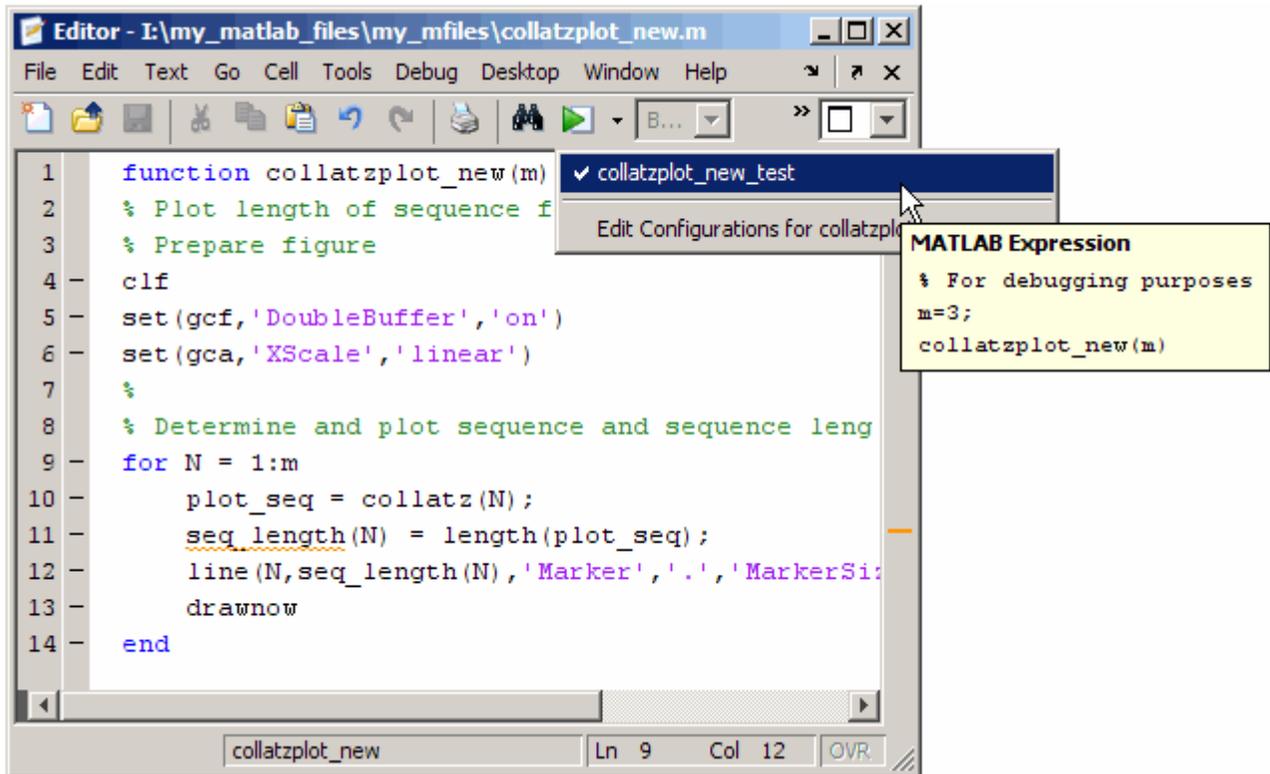
To run the configuration, select the configuration name. MATLAB runs the expression you specified in the configuration. For example, select `collatzplot_new_test`, and MATLAB runs `collatzplot_new(3)`, as specified in step 3. You can modify the M-file, save it, and run the configuration from the toolbar to see the effects of the M-file changes.

Create and Run Multiple Configurations for an M-File

You can create multiple configurations for a given M-file, allowing you to run with different values for input arguments, each for a different purpose. Create a named configuration for each purpose, all associated with the M-file. Then any time you open the M-file, chose and run the configuration you want. For example, for `collatzplot_new(m)` you might use three values for `m` and have three configurations:

- Small value, e.g., 3, for debugging and testing

- Realistic value, e.g., 200 or more, for a specific project

- Random value to observe changes

1 Open the Edit M-File Configurations dialog box. Select the M-file to which you want to add a configuration, or select a configuration associated with that M-file. Click the Add button ➕ (under the list of M-files and configurations). MATLAB creates a new default configuration template, in this example, `collatzplot_new`.

The example shows `collatzplot_new` and its default expression, as well as one previously created configuration associated with `collatzplot_new.m`, `collatzplot_new_test`.
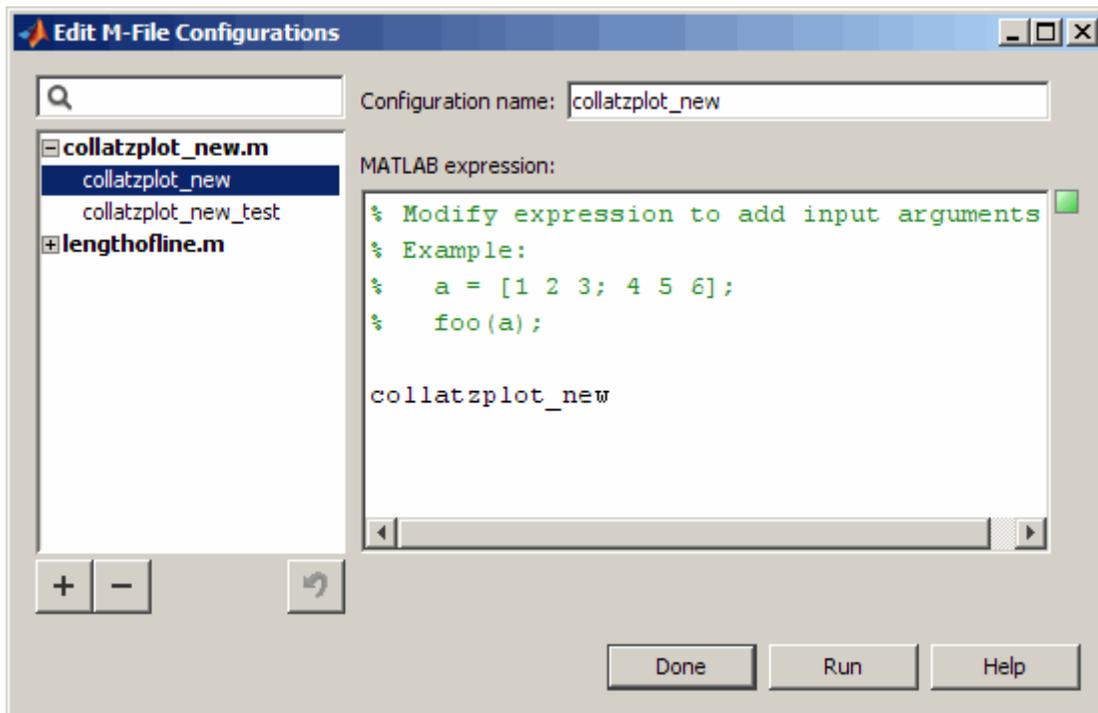
2 In the Edit M-File Configurations dialog box, modify, run, and name the new configurations as you did for the initial configuration,

collatplot_new_test, described in "Create and Run a Configuration for an M-file" on page 6-67.

For example, rename collatzplot_new to collatzplot_new_largevalue, and replace the default template expression with

```
m=200;
collatzplot_new(m)
```

To create another configuration, click the Add button ⊞. Rename collatzplot_new to collatzplot_new_random and replace the default template expression with

```
% Random value
m=int16(rand*50);
collatzplot_new(m)
clear all
```

3 Select a configuration in the listing to see and modify its expression, or to rename the configuration. Click the expander next to an M-file name (**+** and **-** icon on Windows) to see or hide all the configurations associated with that M-file.

4 To get a quick view of the expression in a configuration, position the pointer on the name of a configuration without selecting it. In this example, collatzplot_new_largevalue is selected and you can edit its expression or name. The pointer is positioned on collatzplot_new_test and you can see the statements in it.
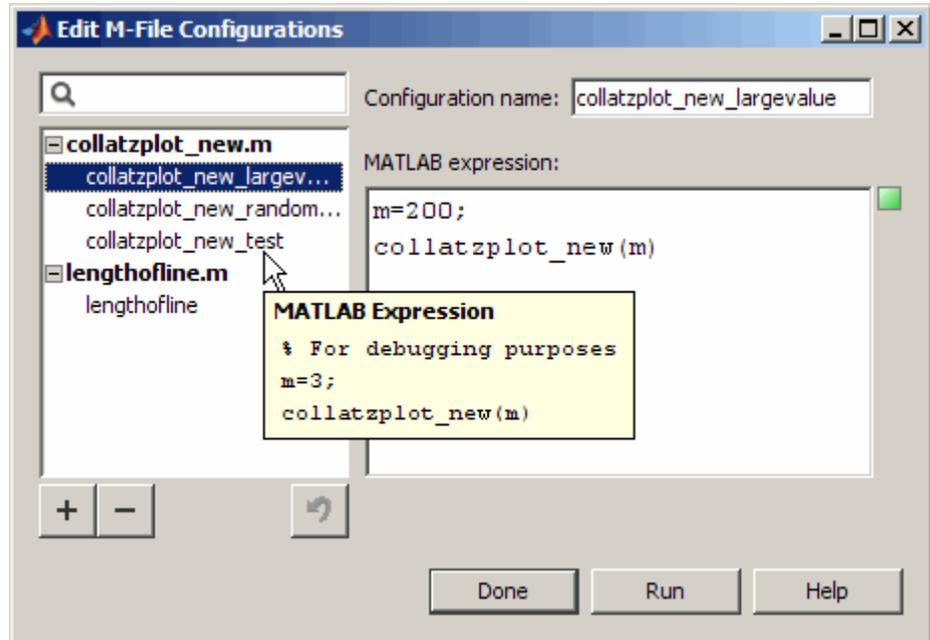
5 Click **Done** to close the Edit M-File Configurations dialog box.

Find Configurations

1 Open any M-file in Editor/Debugger. For example, open the MATLAB function `sin`.

2 Open the Edit M-File Configurations dialog box. MATLAB automatically creates a default configuration for `sin.m`, if none exists.

In the left panel, MATLAB displays a list of all M-files containing configurations.
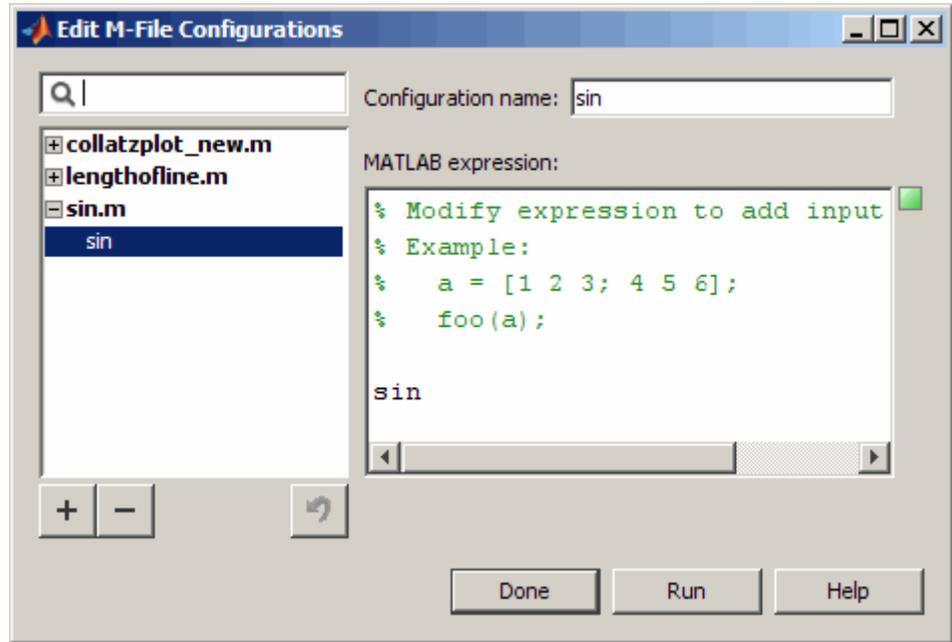
3 Type a term in the filter field [Q] to find an M-file or configuration by name.

MATLAB displays only those M-files whose names contain the term, or whose associated configurations contain the term in their name. As you type, MATLAB filters out files and configurations that do not contain the term.

For example, type `rand`. In this example, only one M-file, `collatzplot_new.m`, has a configuration that contains the term `rand`.
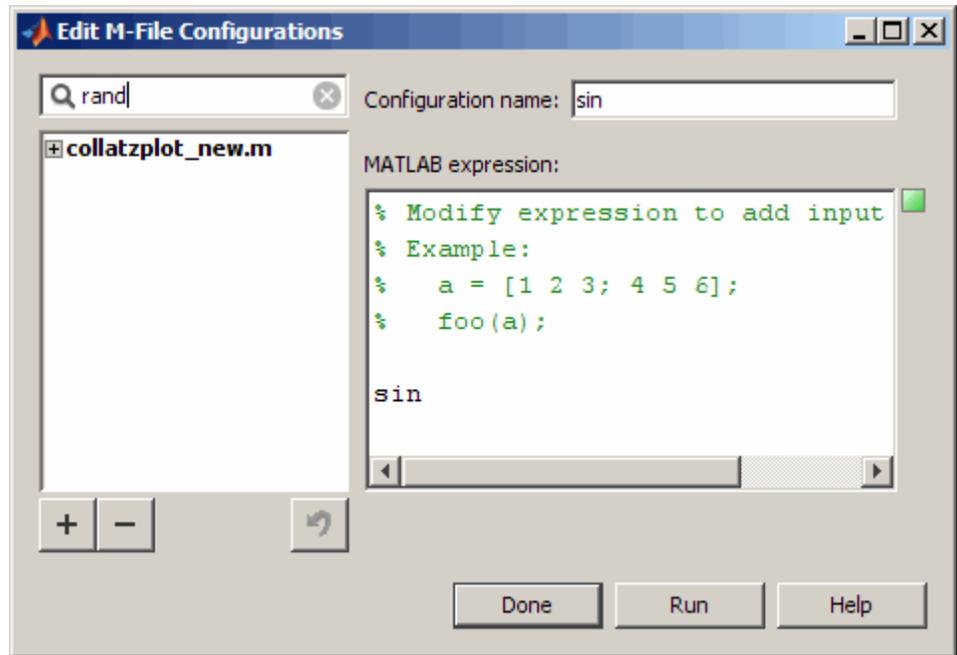
4 Expand the M-file name to see associated configurations whose names contain the term you entered in the filter field. In this example, click the expander (**+** on Windows) for collatzplot_new.m to see configurations whose names contain rand. If you cannot view the entire name of a configuration, drag the separator bar to the right of the list, making the left panel wider. For the example, there is only one configuration, collatzplot_new_randomvalue.
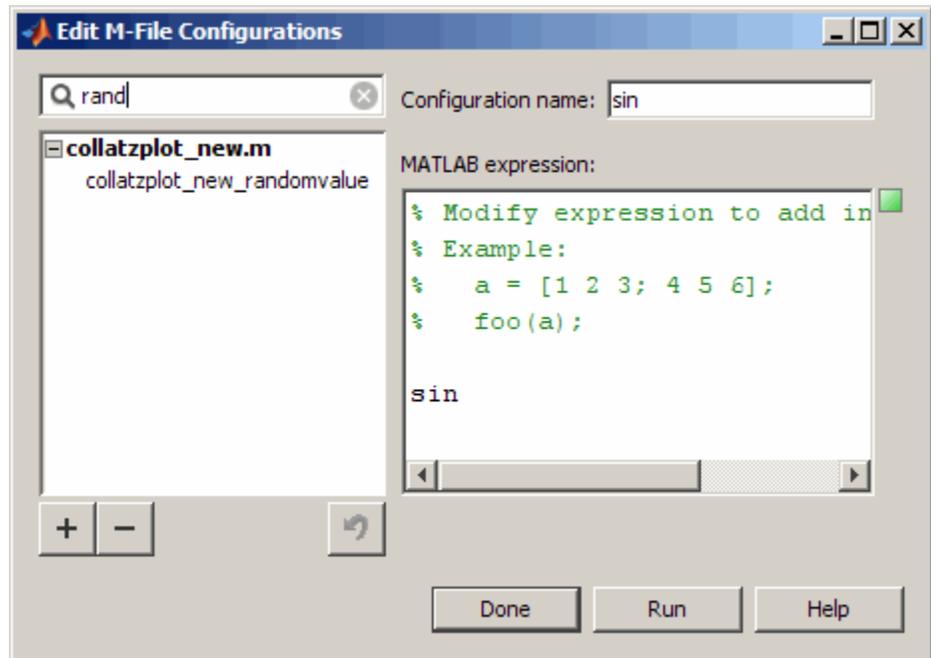
5 To see the expression in that configuration, select the configuration, or position the pointer over the name.

6 As you type additional letters in the filter field, fewer M-files remain in the list of results. Use the backspace key to modify the term. If there are no M-files or configurations containing the term, the list will be empty.

7 To clear the filter and show all M-files with configurations, click the clear button ⊗ in the filter field.

Remove Configurations

If you no longer need a configuration because you do not use it or because you deleted the M-file with which it is associated, it is a good practice to delete the configuration.

1 Open any M-file in the Editor/Debugger.

2 Open the Edit M-File Configurations dialog box.

3 Select the configuration you want to delete.

4 Click the Remove button ⊟.

5 To undo the last deletion, click the Undo button ↻ You cannot undo the last deletion after having closed the Edit M-File Configuration dialog box.

Reassociate and Rename Configurations

Each configuration is associated with a specific M-file. If you move or rename the M-file, you need to redefine the association. If you delete an M-file, you might want to delete the associated configurations, or associate them with a different M-file. You might also need to modify the statements in the configurations so they will run.

When MATLAB cannot associate a configuration with an M-file, the Edit M-File Configurations dialog box displays the M-file name in red, displays a **File Not Found** message, and allows you to find the M-file to which you want to associate the configuration. In this example, MATLAB cannot find the file collatzplot_new.m, which has three configurations associated with it. For this example, collatzplot_new.m had been renamed to collatzplot.m, so the configurations associated with collatzplot_new.m need to be reassociated.
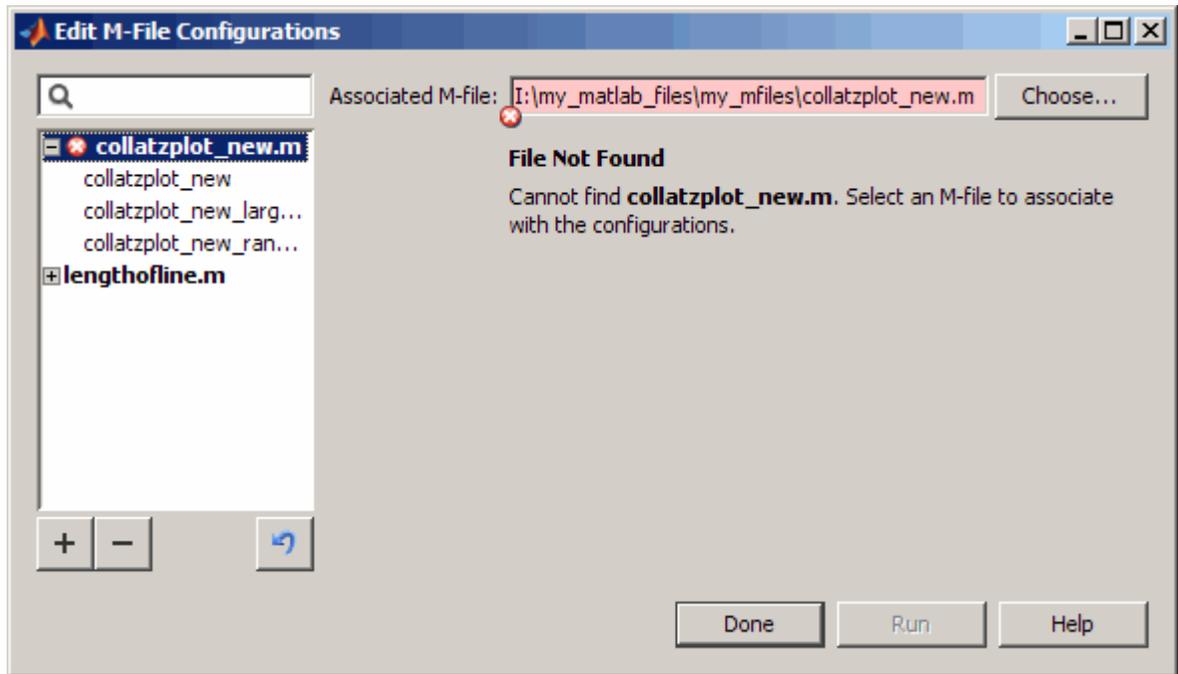
To reassociate configurations

1 In the list of configurations (left pane), select the M-file. The **Associated M-file** displays the full path to the M-file that was associated with the configurations. Click **Choose**.

2 In the resulting Open dialog box, navigate to and select the M-file with which you now want to associate the configurations. Click **Open**

In this example, you want to associate the configurations with `collatzplot.m`; select `collatzplot.m`.

3 In the Edit M-File Configurations dialog box, the **Associated M-file** value reflects the change you made and the **File Not Found** message no longer appears.
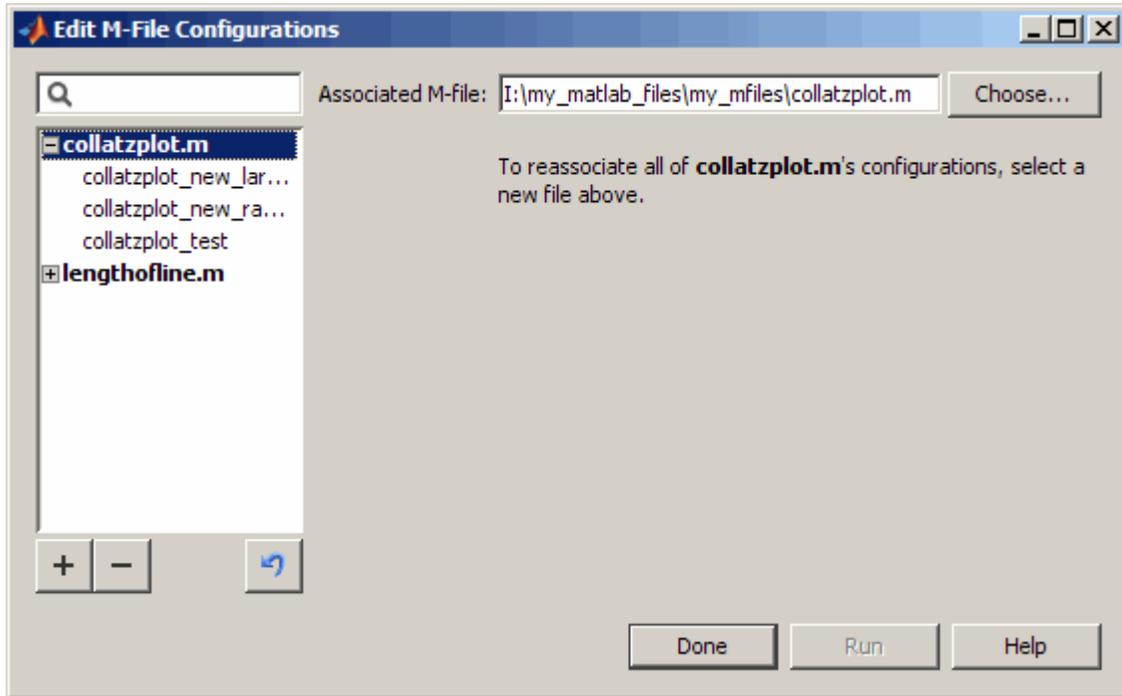
4 You might want to rename the configurations to be consistent with the new M-file name, or at least to not reflect the former M-file name. This is not required, but it is a good practice. To do so, select a configuration from the list in the left pane. In the right pane, edit the value for **Configuration name**. Repeat for all configurations associated with the M-file.

In this example, remove `collatzplot_new` from the start of each configuration name.
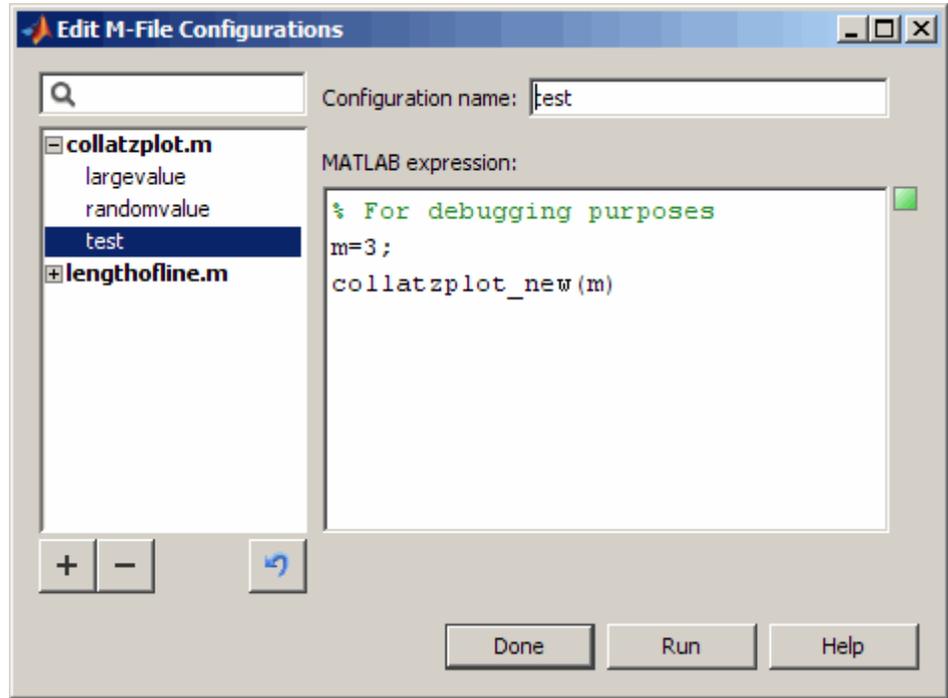
5 For an M-file name change, you might need to modify the configuration statements to run correctly. For this example, modify the `collatzplot_new(m)` statement in each configuration to use `collatzplot(m)`.
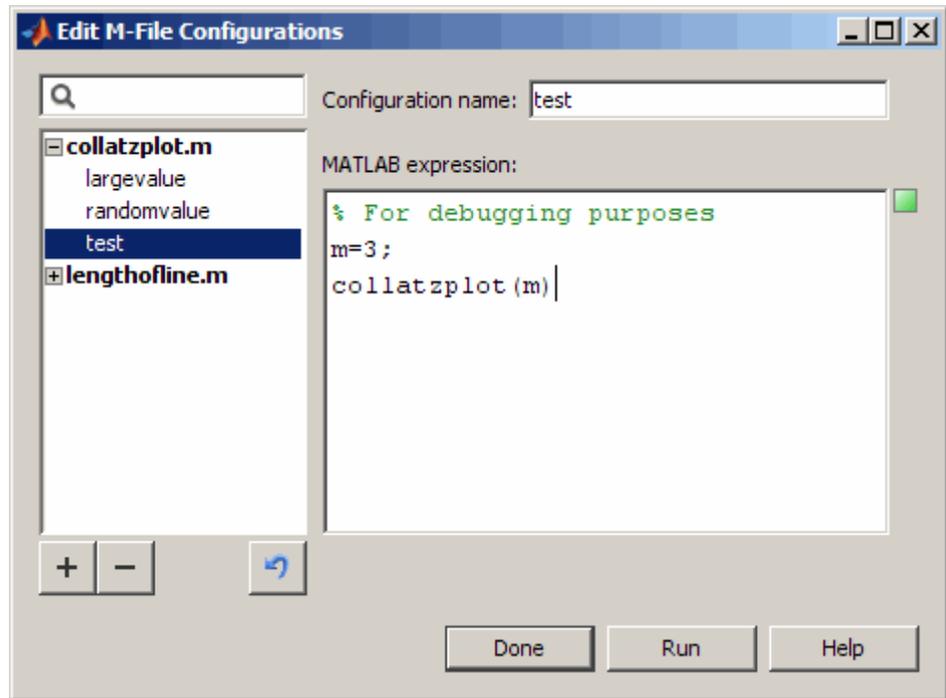
See Also — Other Ways to Run M-Files from the Editor/Debugger

- See "Running an M-File with Breakpoints" on page 6-111 for additional information about running M-files while debugging.

- While debugging, you can execute sections of an M-file even though there are unsaved changes—see "Running Sections in M-Files That Have Unsaved Changes" on page 6-125.

- You can execute M-files one section at a time and quickly modify values incrementally using the toolbar—for more information, see "Using Cells for Rapid Code Iteration and Publishing Results" on page 6-133.

Finding Errors, Debugging, and Correcting M-Files

This section introduces general techniques for finding errors and using M-Lint automatic code analyzer to detect possible improvements in M-files. It then illustrates MATLAB debugger features found in the Editor/Debugger, as well equivalent Command Window debugging functions, using a simple example.

There are two kinds of errors:

• Syntax errors — For example, misspelling a function name or omitting a parenthesis.

• Run-time errors — These errors are usually algorithmic in nature. For example, you might modify the wrong variable or code a calculation incorrectly. Run-time errors are usually apparent when an M-file produces unexpected results. Run-time errors are difficult to track down because the function's local workspace is lost when the error forces a return to the MATLAB base workspace. The process of isolating and fixing these run-time problems is referred to as *debugging*.

In addition to finding and fixing problems with your M-files, you might want to improve the performance and make other enhancements using MATLAB tools.

Use the following techniques to isolate the causes of errors and improve your M-files.

Technique or Tool	Description	For More Information
Syntax highlighting and Delimiter matching	Syntax highlighting helps you identify unterminated strings in an M-file before you run the file. Delimiter matching helps you correctly match pairs of parentheses, brackets, braces, and keywords.	"Syntax Highlighting" on page 6-28 "Matching Delimiters (Parentheses)" on page 3-17

Technique or Tool	Description	For More Information
Error Messages	When you run an M-file with a syntax error, MATLAB will most likely detect it and display an error message in the Command Window describing the error and showing its line number in the M-file. Click the underlined portion of the error message, or position the cursor within the message and press **Ctrl+Enter**. The offending M-file opens in the Editor/Debugger, scrolled to the line containing the error. To check for syntax errors in an M-file without running the M-file, use the `pcode` function.	None
M-Lint	Use the M-Lint code analyzer to help you verify the integrity of your code and learn about potential improvements. Access M-Lint messages automatically while you work in a file in the Editor/Debugger, or run an M-Lint report for an existing file. To evaluate the McCabe complexity (also known as the cyclomatic complexity) of an M-File, use the `mlint` function with the `-cyc` option.	"M-Lint Code Analyzer" on page 6-87 and the reference page for the `mlint` function
Editor/ Debugger Graphical Debugger and MATLAB Debugging Functions	The MATLAB Editor/Debugger graphical debugger and MATLAB debugging functions are useful for correcting run-time problems because you can access function workspaces and examine or change the values they contain. You can set and clear *breakpoints*, indicators that temporarily halt execution in an M-file. While stopped at a breakpoint, you can change workspace contexts, view the function call stack, and execute the lines in an M-file one by one.	"Debugging Process and Features" on page 6-103

Technique or Tool	Description	For More Information
Other Debugging Techniques	• Add keyboard statements to the M-file — keyboard statements stop M-file execution at the point where they appear and allow you to examine and change the function's local workspace. This mode is indicated by a special K>>prompt. Resume function execution by typing return and pressing the **Enter** key. For more information, see the keyboard reference page. • Remove selected semicolons from the statements in your M-file—semicolons disable the display of output in the M-file. By removing the semicolons, you instruct MATLAB to display these results on your screen as the M-file executes. • List dependent functions—use the depfun function to see the dependent functions.	Reference pages for keyboard and depfun function
Cells	In the Editor/Debugger, isolate sections of an M-file, called cells, so you can easily make changes to and run a single section.	"Using Cells for Rapid Code Iteration and Publishing Results" on page 6-133
Profiler	Use the Profiler to help you improve performance and detect problems in your M-files. Access the Profiler from the Editor/Debugger by selecting **Tools > Open Profiler**.	"Profiling for Improving Performance" on page 7-27
Directory Reports	The M-file Directory Reports help you polish and package M-files before providing them to others to use. Access all of these tools from the Current Directory browser. You can access some of these directly from the Editor/Debugger **Tools** menu.	"Directory Reports in Current Directory Browser" on page 7-2

M-Lint Code Analyzer

In this section...
"What Is M-Lint?" on page 6-87
"Ways to Use M-Lint" on page 6-87
"M-Lint Automatic Code Analyzer in the Editor/Debugger" on page 6-88
"Suppressing M-Lint Indicators and Messages" on page 6-98

What Is M-Lint?

The M-Lint code analyzer checks your code for problems and recommends modifications to maximize performance and maintainability.

Ways to Use M-Lint

You can use M-Lint in two different ways, both of which report the same information:

- Run a report for an existing M-file or group of M-files. To do so, from an M-file in the Editor/Debugger, select **Tools > M-Lint > Show M-Lint Report**. Make any changes to your file based on the M-Lint messages in the report. After making changes, you must save the file and rerun the report to see if your changes addressed the issues noted in M-Lint messages. To run M-Lint for all files in a directory, access M-Lint from the Current Directory browser — select **View > Directory Reports > M-Lint Code Check Report**. For details, see "M-Lint Code Check Report" on page 7-16.

- Continuously check code in the Editor/Debugger while you work. View M-Lint messages and make changes to your file based on the messages. The code analyzer updates automatically and continuously so you can see if your changes addressed the issues noted in the M-Lint messages. For some messages, M-Lint offers automatic code correction. For details about specific M-Lint messages, see "M-Lint Code Check Report" on page 7-16. Information about using the continuous checking and correction interface in the Editor/Debugger is explained here.

M-Lint Automatic Code Analyzer in the Editor/Debugger

To use the M-Lint continuous code checking in an M-file in the Editor/Debugger, perform the following steps:

1 Ensure the M-Lint messaging preference is enabled: Select **File > Preferences > M-Lint** and select the **Enable integrated M-Lint warning and error messages** check box. To follow these instructions, be sure the **Underlining** option is set to Underline warnings and errors. Click **OK**.
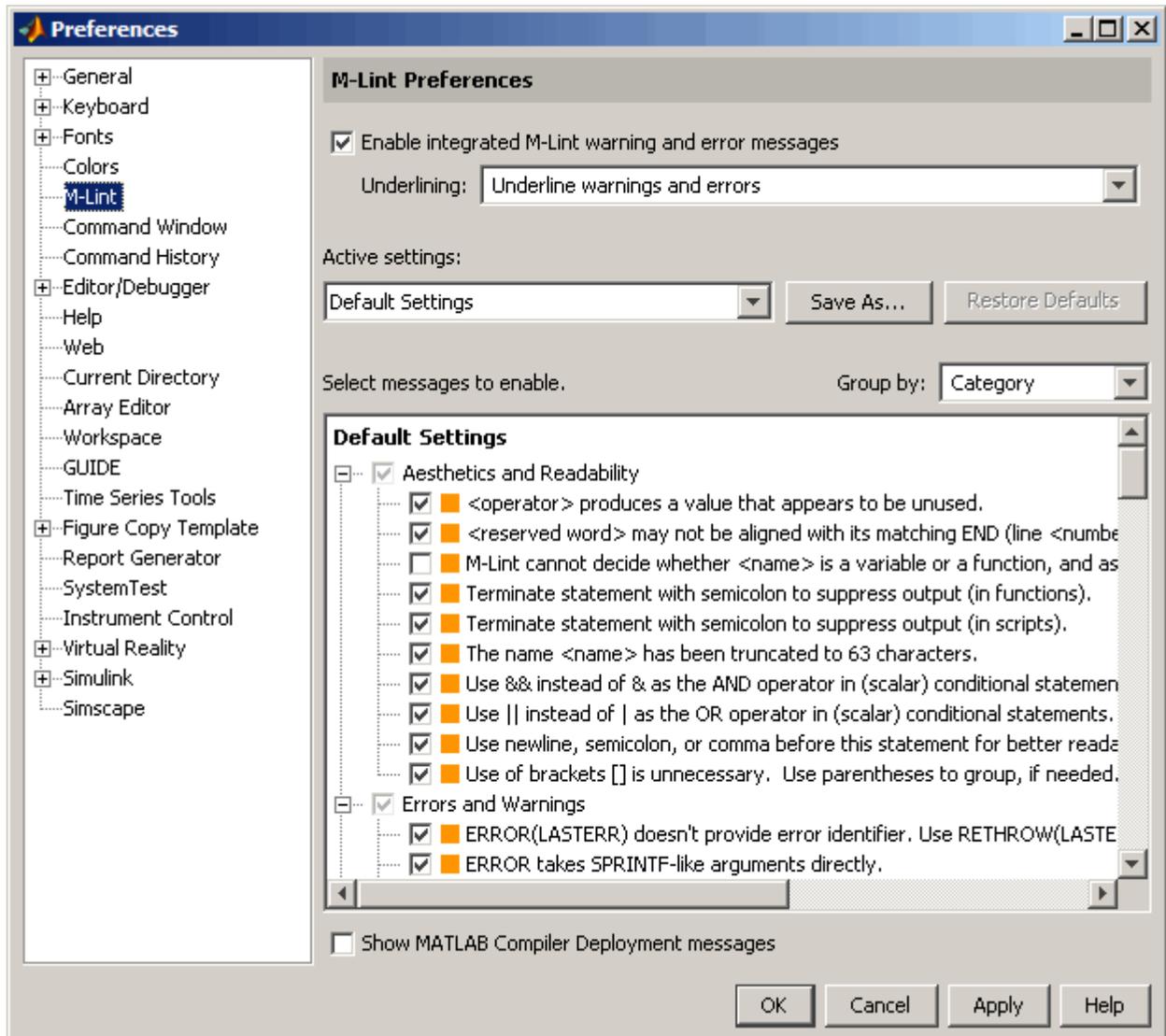
2 Open an M-file in the Editor/Debugger. This example uses the sample file `lengthofline.m`:

  a Open the example file:

```
open(fullfile(matlabroot,'help','techdoc','matlab_env','examples','lengthofline.m'))
```

b Save the example file to a directory to which you have write access. For the example, lengthofline.m is saved to I:\MATLABFiles\mymfiles.

3 The M-Lint message indicator at the top right edge of the window conveys the M-Lint messages reported for the file:

- Red means syntax errors were detected. Another way to detect some of these errors is using syntax highlighting to identify unterminated strings, and delimiter matching to identify unmatched keywords, parentheses, braces, and brackets.

- Orange means warnings or opportunities for improvement were detected, but no errors were detected.

- Green means no errors, warnings, or opportunities for improvement were detected.

For the example, the indicator is red, meaning there is at least one error in the file.

M-Lint message indicator for all messages in entire file:
- Red means errors detected
- Orange means warnings or improvement opportunities detected
- Green means none detected

Click indicator to go to next line that has an associated M-Lint message.

Current cursor position

4 Click the M-Lint message indicator to go to the next code fragment containing an M-Lint message. The next code fragment is relative to the current cursor position, viewable in the status bar.

In the lengthofline example, the first message is at line 22. The cursor moves to the beginning of line 22.

5 The code fragment for which there is an M-Lint message is underlined in either red for errors or orange for warnings and improvement opportunities.

To view the M-Lint message, move the pointer within the underlined fragment. The message appears with a yellow highlighted background, similar to datatips (see "Viewing Values as Datatips in the Editor/Debugger" on page 6-115).

Position cursor within orange underlined code fragment and
Editor/Debugger displays the related M-Lint message.
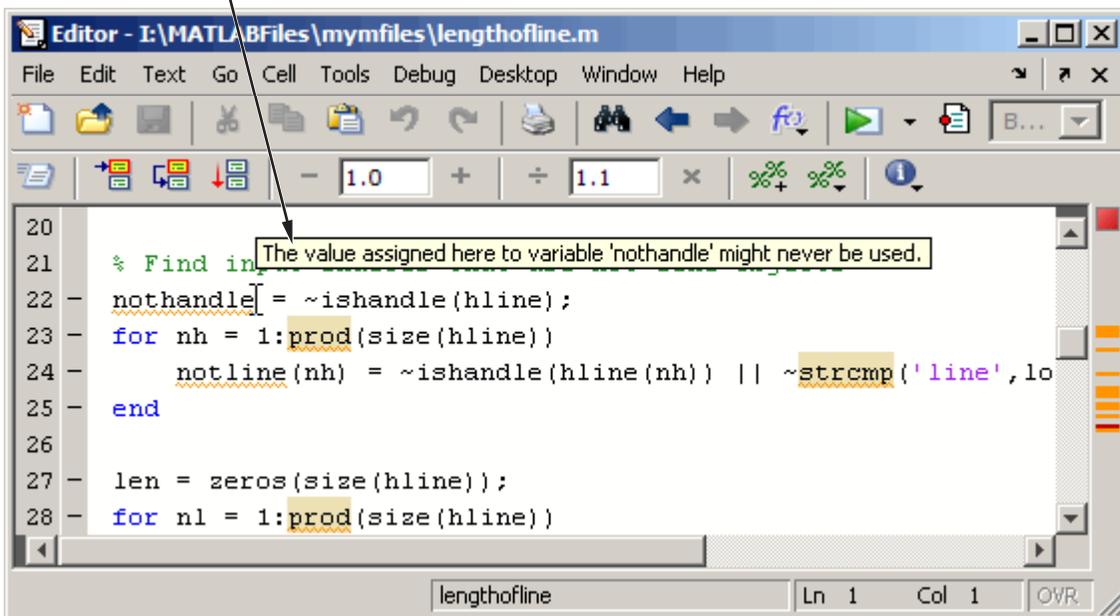
This message means that in line 22, nothandle is assigned a value, but is probably not used anywhere after that in the file. The line might be

extraneous and you could delete it. But it might be that you actually intended to use the variable, as shown in step 6 of this example.

6 Make changes to your code as needed. The M-Lint indicator and underlining automatically update to reflect the changes you make, even if you do not save the file.

In this example, the intention was to use `nothandle` as a performance improvement by determining the value prior to the loop. Changing `~ishandle(hline(nh))` in line 24 to `nothandle(nh)` means there is no longer an M-Lint message associated with line 22. For more information about what the warning and improvement messages in this example mean and actions you can take to address them, see "Messages and Resulting Changes for the lengthofline Example" on page 7-21.

7 Some errors and warnings are highlighted, indicating M-Lint can automatically fix the code. For example, in `lengthofline`, line 23, `prod` is underlined because there is an M-Lint warning, and it is highlighted because an automatic fix is available. When you view the M-Lint message, it also indicates the auto-fix that is available.
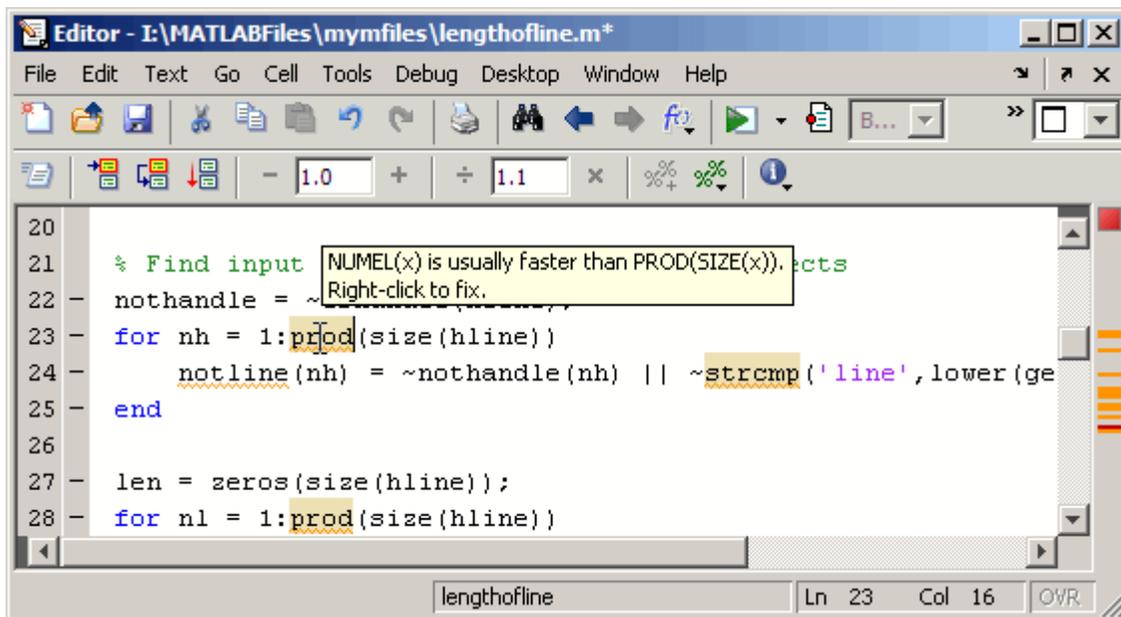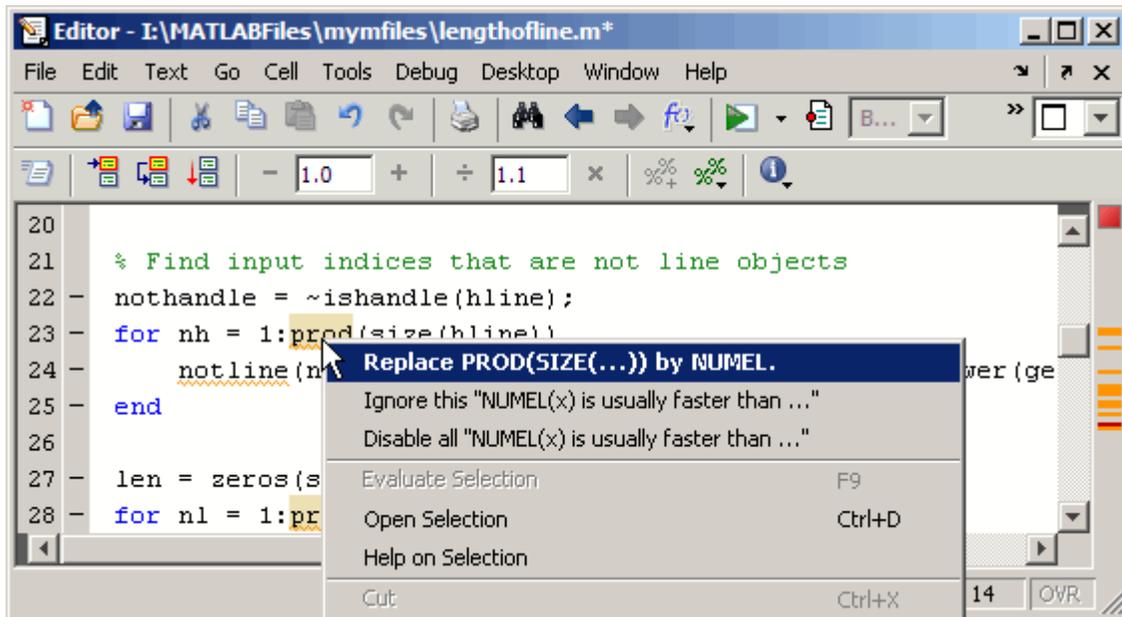
Right-click the highlighted code (for a single-button mouse, use **Ctrl**+click). The first item in the context menu indicates the automatic fix that M-Lint can perform. Select it and M-Lint automatically corrects the code. In this example, M-Lint replaces `prod(size(hline))` with `numel(hline)`.

There is a preference you can set for the color—for more information, see "Other Colors" on page 2-75.

8 You might want to ignore certain M-Lint messages and do not want the messages to display; for more information, see "Suppressing M-Lint Indicators and Messages" on page 6-98.

9 You can click the M-Lint message indicator to go to the next message, or use the other way to view messages, which is the M-Lint message bar. Each marker in the bar represents a line that has associated M-Lint messages. A red marker means there is an error at that line, while an orange marker means there are warnings or suggested improvements, but no errors at that line.

a Position the pointer at a marker in the message bar to view the message. For example, to see an error in `lengthofline`, position the pointer at a red marker in the message bar. There is only one error in the file and with the pointer positioned over it, the associated M-Lint messages appears. Click the marker to go to the first code fragment in the line that resulted in an M-Lint message. For the example, click the red marker, which takes you to the first suspect code fragment in line 48.

6-95

```
temp = diff([data{1}(:) data{2}(:) data{3}(;)]);
```

Multiple messages can represent a single problem or multiple problems. Addressing one might address all of them, or after addressing one, the other messages might change or what you need to do might become clearer.

M-Lint message bar. Each marker represents an M-Lint message.

Position the pointer at a marker to view the message, in this example the only error marker in the file. There can be multiple messages for one line. Click a marker to go to the code fragment that resulted in the message. In this example, the multiple messages actually result from a single error.
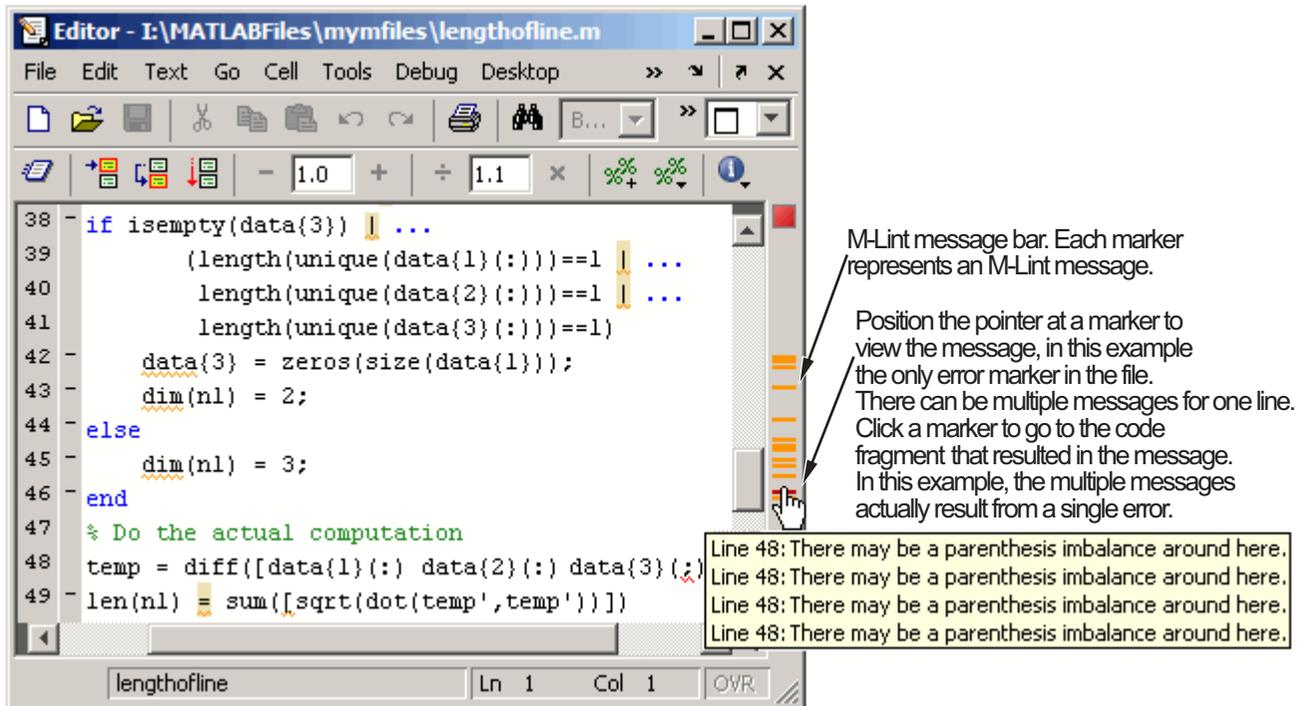
b Make changes to address the problem noted in the M-Lint message—the M-Lint indicators update automatically.

In the example, the M-Lint message suggest a delimiter imbalance. You can check that by moving the arrow key over each of the delimiters to see if MATLAB indicates a mismatch. This requires that **File > Preferences > Keyboard > Delimiter Matching** has the **Match on arrow key** option selected. There are no mismatched delimiters. The actual problem is the semicolon in parentheses, data{3}(;), is incorrect and should be a colon. In line 48, change data{3}(;) to data{3}(:). When you make the change, the underline no longer appears in line 48.

That single change addressed the issues in all of the M-Lint messages for line 48.

Because the change you made removed the only error in the file, the M-Lint message indicator at the top of the bar changes from red to orange, indicating that only warnings and potential improvements remain.

c If there are multiple messages associated with a line, there might be multiple underlined code fragments that are adjacent, as in the above example, making it difficult to display the message of interest. In those cases, it might be easier to view the messages via the marker on the message bar.

10 After making changes to address all M-Lint messages, or disabling designated messages, the M-Lint message indicator becomes green The example file with all M-Lint messages addressed has been saved as `lengthofline2.m`, which you can open by running

```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...
'examples','lengthofline2.m'))
```
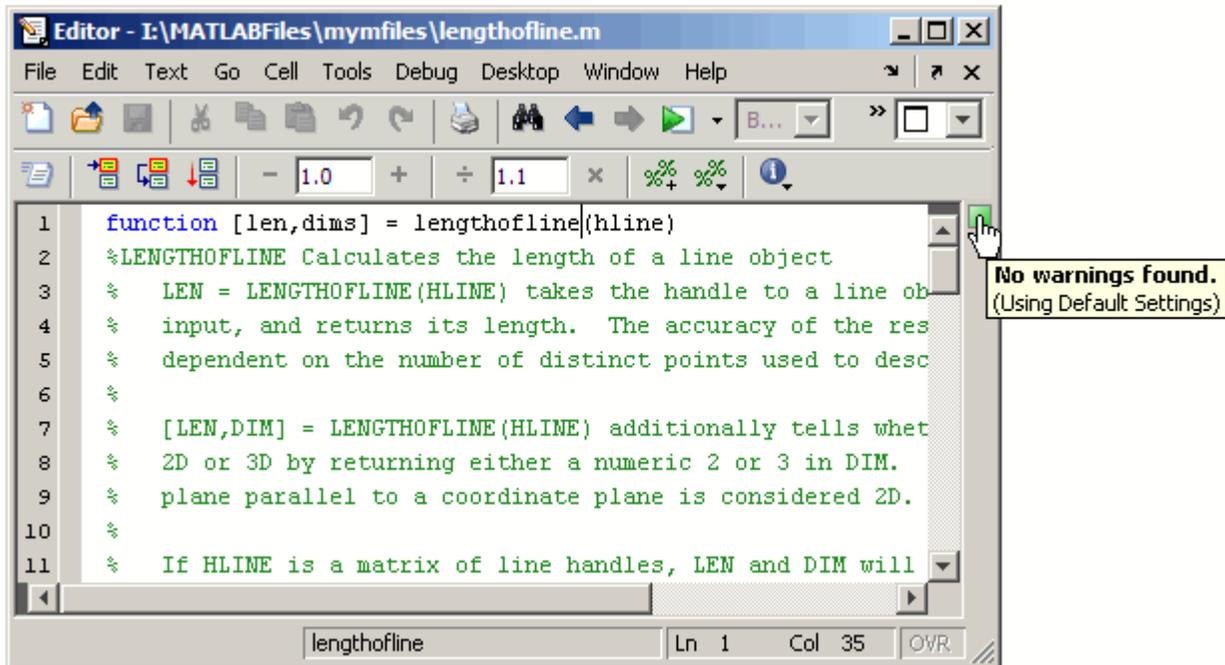
Suppressing M-Lint Indicators and Messages

Depending on what stage you are at in completing the M-file, you might want to restrict the underlining, which you can do via the M-Lint preference referred to in step 1, above. For example, when first coding, you might prefer no underlines because they would be distracting. For details, click the **Help** button in the Preferences dialog box.

M-Lint does not provide perfect information about every situation and in some cases, you might not want to make any changes based on an M-Lint message. In the event you do not want to change the code but you also do not want to see the M-Lint indicator and message for that line, instruct M-Lint to suppress them. For the `lengthofline` example, in line 49, the first M-Lint message is `Terminate statement with semicolon to suppress output (in functions)`. Adding a semicolon to the end of a statement suppresses output and is a common practice. M-Lint alerts you to lines that produce output but lack the terminating semicolon. If you want to view output from line 49, do not add the semicolon as M-Lint suggests.

There are a few different ways to suppress the M-Lint indicators and messages:

- "Ignore Only a Specific Instance" on page 6-99
- "Disable All Instances in All Files" on page 6-101
- "Disable Specified Messages or in Selected Files as Needed" on page 6-101

Note that you cannot suppress M-Lint error messages such as syntax errors, and therefore, the following options do not apply.

Ignore Only a Specific Instance

Right-click at the M-Lint underline (for a single-button mouse, use **Ctrl**+click). From the context menu, select Ignore this "Terminate statement with semicolon...". M-Lint adds a %#ok<NOPRT> to the end of the line, which instructs MATLAB not to check for a terminating semicolon at that line. M-Lint removes the underline and mark in the M-Lint indicator bar for that message.

If there are two messages on a line that you do not want M-Lint to display, right-click separately at each underline and select the appropriate entry from the context menu. M-Lint expands the %#ok syntax. For the example, ignoring both messages for line 49 would add %#ok<NBRAK,NOPRT>.

For more information about %#ok, see the mlint function reference page.

This method of suppressing the messages changes the M-file. If M-Lint preferences are to set to enable this message, the specific instance of the message suppressed in this way will not appear because the %#ok takes precedence over the preference setting. If you later decide you want M-Lint to check for a terminating semicolon at that line, delete the %#ok<NOPRT> from the line.
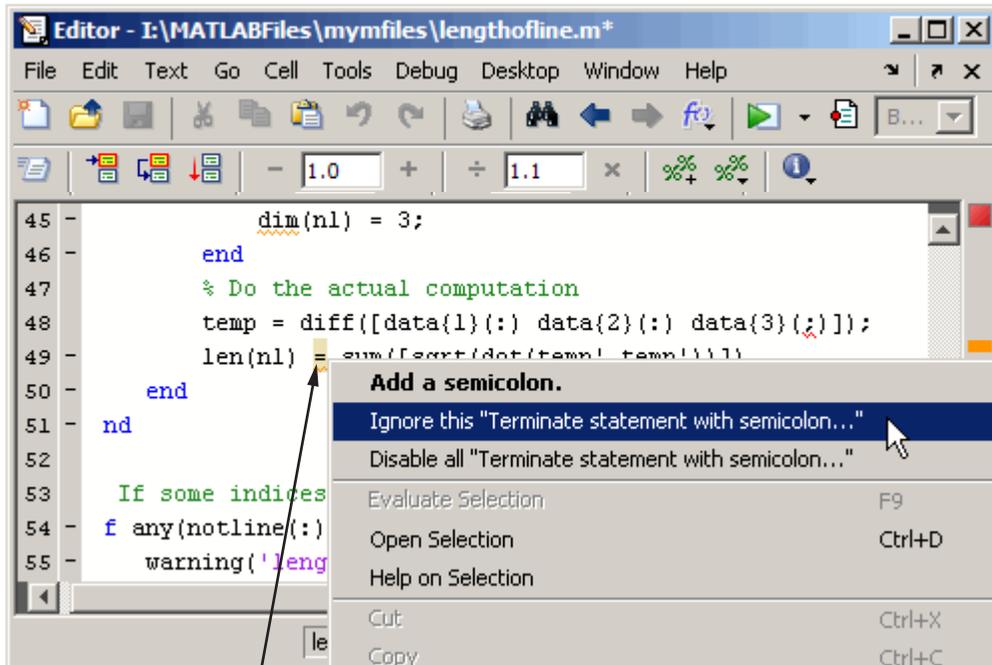
Right-click at an M-Lint underline and select the option instructing M-Lint to ignore only this instance of the message, as shown in this example.

M-Lint adds %#ok for a specific message to the
end of a line for which you specified the M-Lint message
should be suppressed.

Disable All Instances in All Files

Right-click at the M-Lint underline (for a single-button mouse, use **Ctrl**+click).
From the context menu, select `Disable all "Terminate statement with
semicolon..."`. Doing so modifies the M-Lint preference setting, which
applies to all occurrences in all files, unless a line includes a %#ok for that
message. For more information about the M-Lint preference, including how
to restore MATLAB default settings, select **File > Preferences > M-Lint**,
and click **Help**.

Disable Specified Messages or in Selected Files as Needed

Use M-Lint preferences by selecting **File > Preferences > M-Lint**. Then
enable specific messages or categories of messages and save the settings to a

txt file. You can reuse the settings for any M-file, or provide the settings file to another user.

To use the saved settings, either select the settings file in M-Lint preferences, or in the Editor/Debugger. In the Editor/Debugger, right-click the M-Lint message bar (for a single-button mouse, use **Ctrl**+click), or select **Tools > M-Lint**. The currently-selected setting choice is shown, preceded by a bullet point. You can choose from any of the settings files, such as the MLintNoSemis example, as shown here.

M-Lint default settings are currently selected (as indicated by the bullet point preceding that menu item). Select any txt file to use the M-Lint settings specified in that file.
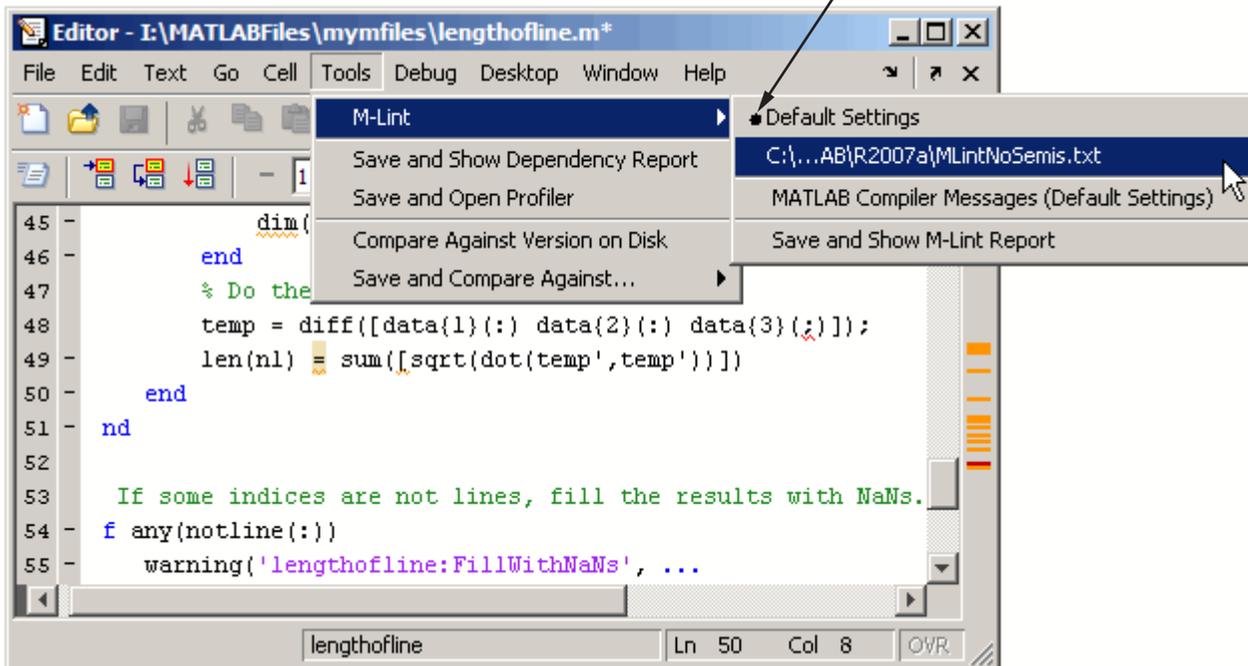
For more about M-Lint settings and preferences, click **Help** in the M-Lint preferences panel.

Debugging Process and Features

Ways to Debug M-Files

You can debug the M-files using the Editor/Debugger, which is a graphical user interface, as well as by using debugging functions from the Command Window. You can use both methods interchangeably. These topics and the example describe both methods.

Preparing for Debugging

Do the following to prepare for debugging:

- Open the file — To use the Editor/Debugger for debugging, open it with the file to run.

- Save changes — If you are editing the file, save the changes before you begin debugging. If you try to run a file with unsaved changes from within the Editor/Debugger, the file is automatically saved before it runs. If you run a file with unsaved changes from the Command Window, MATLAB runs the saved version of the file, so you will not see the results of your changes.

- Add the files to a directory on the search path or put them in the current directory — Be sure the file you run and any files it calls are in directories

that are on the search path. If all files to be used are in the same directory, you can instead make that directory be the current directory.

Debugging Example — The Collatz Problem

The debugging process and features are best described via an example. To prepare to use the example, create two M-files, `collatz.m` and `collatzplot.m`, that produce data for the Collatz problem.

For any given positive integer, `n`, the `Collatz` function produces a sequence of numbers that always resolves to 1. If `n` is even, divide it by 2 to get the next integer in the sequence. If `n` is odd, multiply it by 3 and add 1 to get the next integer in the sequence. Repeat the steps until the next integer is 1. The number of integers in the sequence varies, depending on the starting value, `n`.

The Collatz problem is to prove that the Collatz function will resolve to 1 for all positive integers. The M-files for this example are useful for studying the Collatz problem. The file `collatz.m` generates the sequence of integers for any given n. The file `collatzplot.m` calculates the number of integers in the sequence for all integers from 1 through m, and plots the results. The plot shows patterns that can be further studied.

Following are the results when `n` is 1, 2, or 3.

n	Sequence	Number of Integers in the Sequence
1	1	1
2	2 1	2
3	3 10 5 16 8 4 2 1	8

M-Files for the Collatz Problem. Following are the two M-files you use for the debugging example. To create these files on your system, open two new M-files. Select and copy the following code from the Help browser and paste it into the M-files. Save and name the files `collatz.m` and `collatzplot.m`. Save them to your current directory or add the directory where you save them to the search path. One of the files has an embedded error to illustrate the debugging features.

Code for `collatz.m`.

```
function sequence=collatz(n)
% Collatz problem. Generate a sequence of integers resolving to 1
% For any positive integer, n:
%  Divide n by 2 if n is even
%  Multiply n by 3 and add 1 if n is odd
%  Repeat for the result
%  Continue until the result is 1%

sequence = n;
next_value = n;
while next_value > 1
 if rem(next_value,2)==0
  next_value = next_value/2;
 else
  next_value = 3*next_value+1;
 end
 sequence = [sequence, next_value];
end
```

Code for `collatzplot.m`.

```
function collatzplot(m)
% Plot length of sequence for Collatz problem
% Prepare figure
clf
set(gcf,'DoubleBuffer','on')
set(gca,'XScale','linear')
%
% Determine and plot sequence and sequence length
for N = 1:m
 plot_seq = collatz(N);
 seq_length(N) = length(plot_seq);
 line(N,plot_seq,'Marker','.','MarkerSize',9,'Color','blue')
 drawnow
end
```

Trial Run for Example. Open the file `collatzplot.m`. Make sure the current directory is the directory in which you saved `collatzplot`.

Try out `collatzplot` to see if it works correctly. Use a simple input value, for example, 3, and compare the results to those shown in the preceding table. Typing

```
collatzplot(3)
```

produces the plot shown in the following figure.
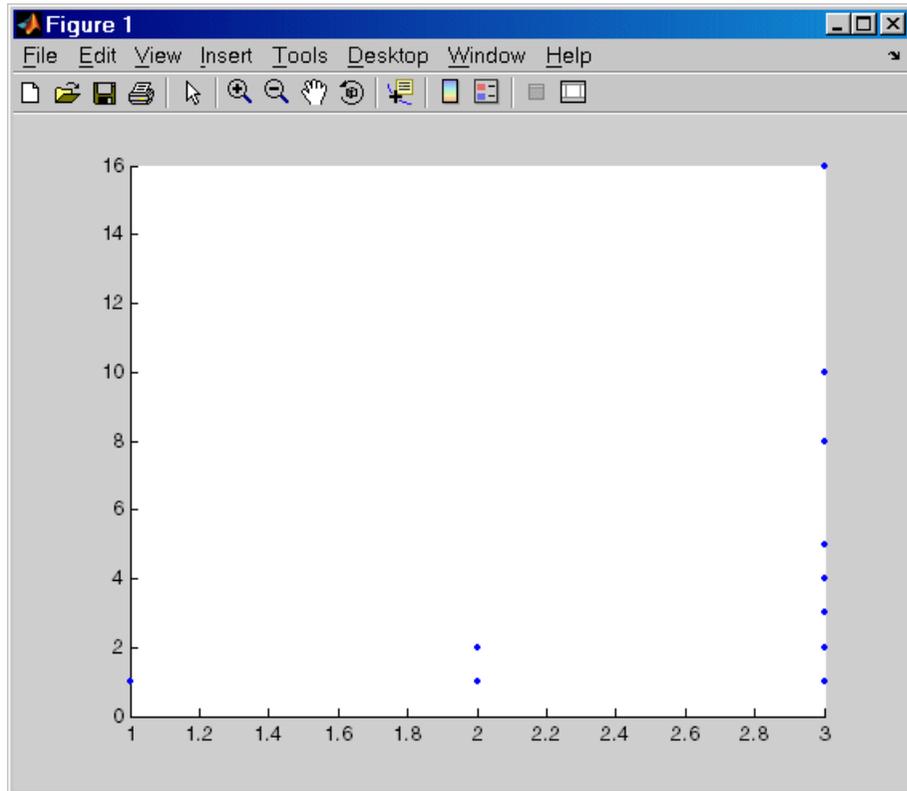
The plot for n = 1 appears to be correct—for 1, the Collatz series is 1, and contains one integer. But for n = 2 and n = 3, it is wrong because there should be only one value plotted for each integer, the number of integers in the sequence, which the preceding table shows to be 2 (for n = 2) and 8 (for n = 3). Instead, multiple values are plotted. Use MATLAB debugging features to isolate the problem.

Setting Breakpoints

Set breakpoints to pause execution of the M-file so you can examine values where you think the problem might be. You can set breakpoints in the Editor/Debugger, using functions in the Command Window, or both.

There are three basic types of breakpoints you can set in M-files:

- A standard breakpoint, which stops at a specified line in an M-file. For details, see "Setting Standard Breakpoints" on page 6-108.

- A conditional breakpoint, which stops at a specified line in an M-file only under specified conditions. For details, see "Conditional Breakpoints" on page 6-126.

- An error breakpoint that stops in any M-file when it produces the specified type of warning, error, or NaN or infinite value. For details, see "Error Breakpoints" on page 6-129.

You can disable standard and conditional breakpoints so that MATLAB temporarily ignores them, or you can remove them. For details, see "Disabling and Clearing Breakpoints" on page 6-119. Breakpoints are not maintained after you exit the MATLAB session.

You can only set valid standard and conditional breakpoints at executable lines in saved files that are in the current directory or in directories on the search path. When you add or remove a breakpoint in a file that is not in a directory on the search path or in the current directory, a dialog box appears, presenting you with options that allow you to add or remove the breakpoint. You can either change the current directory to the directory containing the file, or you can add the directory containing the file to the search path.

Do not set a breakpoint at a `for` statement if you want to examine values at increments in the loop. For example, in

```
for n = 1:10
    m = n+1;
end
```

MATLAB executes the `for` statement only once, which is efficient. Therefore, when you set a breakpoint at the `for` statement and step through the file, you

only stop at the `for` statement once. Instead place the breakpoint at the next line, `m=n+1` to stop at each pass through the loop.

You cannot set breakpoints while MATLAB is busy, for example, running an M-file, unless that M-file is paused at a breakpoint.

Setting Standard Breakpoints

To set a standard breakpoint using the Editor/Debugger, click in the breakpoint alley at the line where you want to set the breakpoint. The breakpoint alley is the narrow column on the left side of the Editor/Debugger, just right of the line number. Set breakpoints at lines that are preceded by a - (dash). Lines not preceded by a dash, such as comments or blank lines, are not executable — if you try to set a breakpoint there, it is actually set at the next executable line. Other ways to set a breakpoint are to position the cursor in the line and then click the Set/Clear Breakpoint button 🔲 on the toolbar, or select **Set/Clear Breakpoint** from the **Debug** menu or the context menu. A breakpoint icon appears.

Set Breakpoints for the Example. It is unclear whether the problem in the example is in `collatzplot` or `collatz`. To start, set breakpoints in `collatzplot.m` at lines 10, 11, and 12. The breakpoint at line 10 allows you to step into `collatz` to see if the problem might be there. The breakpoints at lines 11 and 12 stop the program where you can examine the interim results.

Click where there is a - (dash) to set a breakpoint at that line. The red icon indicates a valid breakpoint is set at that line.

```
Editor - D:\mymfiles\collatzplot.m
File  Edit  Text  Go  Cell  Tools  Debug  Desktop  Window  Help

1      function collatzplot(m)
2      % Plot length of sequence for Collatz problem
3      % Prepare figure
4   -  clf
5   -  set(gcf,'DoubleBuffer','on')
6   -  set(gca,'XScale','linear')
7      %
8      % Determine and plot sequence and sequeence length
9   -  for N = 1:m
10  ◊      plot_seq = collatz(N);
11  ◊          seq_length(N) = length(plot_seq);
12  ◊      line(N,plot_seq,'Marker','.','MarkerSize',9,'Color','blue')
13  -      drawnow
14  -  end

collatzplot        Ln  1      Col  1      OVR
```

Valid (Red) and Invalid (Gray) Breakpoints. Red breakpoints are valid standard breakpoints. If breakpoints are instead gray, they are not valid.

When breakpoints are gray, they are not valid. In this example, it is because the file has not been saved since changes were made to it. Save the file to make the breakpoints valid (red).

```
 Editor - D:\mymfiles\collatzplot.m*                            _ □ ×
File  Edit  Text  Go  Cell  Tools  Debug  Desktop  Window  Help        ↘ | ↗ ×

 1     function collatzplot(m)
 2     % Plot length of sequence for Collatz problem
 3     % Prepare figure
 4 -   clf
 5 -   set(gcf,'DoubleBuffer','on')
 6 -   set(gca,'XScale','linear')
 7     %
 8     % Determine and plot sequence and sequeence length
 9 -   for N = 1:m
10 ◯       plot_seq = collatz(N);
11 ◯           seq_length(N) = length(plot_seq);
12 ◯       line(N,plot_seq,'Marker','.','MarkerSize',9,'Color','blue')
13 -       drawnow
14 -   end

                          collatzplot            Ln  8      Col  22    OVR
```

Breakpoints are gray for either of these reasons:

- The file has not been saved since changes were made to it. Save the file to make breakpoints valid. The gray breakpoints become red, indicating they are now valid. Any gray breakpoints that were entered at invalid breakpoint lines automatically move to the next valid breakpoint line with the successful file save.

- There is a syntax error in the file. When you set a breakpoint, an error message appears indicating where the syntax error is. Fix the syntax error and save the file to make breakpoints valid.

Function Alternative for Setting Breakpoints

To set a breakpoint using the debugging functions, use dbstop. For the example, type

```
dbstop in collatzplot at 10
dbstop in collatzplot at 11
dbstop in collatzplot at 12
```

Some useful related functions are

- dbtype — Lists the M-file with line numbers in the Command Window.

- dbstatus — Lists breakpoints.

Running an M-File with Breakpoints

After setting breakpoints, run the M-file from the Command Window or the Editor/Debugger.

Running the Example

For the example, run collatzplot for the simple input value, 3, by typing in the Command Window

```
collatzplot(3)
```

The example, collatzplot, requires an input argument and therefore runs only from the Command Window and not from the Editor/Debugger.

Results of Running an M-File Containing Breakpoints

Running the M-file results in the following:

- The prompt in the Command Window changes to

    K>>

    indicating that MATLAB is in debug mode.

- The program pauses at the first breakpoint. This means that line will be executed when you continue. The pause is indicated in the Editor/Debugger by the green arrow just to the right of the breakpoint, which in the example, is line 10 of collatzplot as shown here.

    `10 ◆➡        plot_seq = collatz(N);`

    If you use debugging functions from the Command Window, the line at which you are paused is displayed in the Command Window. For the example, it would show

    ```
    10   plot_seq = collatz(N);
    ```

- The function displayed in the **Stack** field on the toolbar changes to reflect the current function (sometimes referred to as the caller or calling workspace). The call stack includes subfunctions as well as called functions. If you use debugging functions from the Command Window, use dbstack to view the current call stack.

- If the file you are running is not in the current directory or a directory on the search path, you are prompted to either add the directory to the path or change the current directory.

In debug mode, you can set breakpoints, step through programs, examine variables, and run other functions.

Note that MATLAB could become nonresponsive if it stops at a breakpoint while displaying a modal dialog box or figure that your M-file creates. In that event, use **Ctrl+C** to go the MATLAB prompt.

Stepping Through an M-File

While debugging, you can step through an M-file, pausing at points where you want to examine values.

Use the step buttons or the step items in the **Debug** menu of the Editor/Debugger or desktop, or use the equivalent functions.

Toolbar Button	Debug Menu Item	Description	Function Alternative
▶	**Continue** or **Run** or **Save and Run**	Continue execution of M-file until completion or until another breakpoint is encountered. The menu item says **Run** or **Save and Run** if a file is not already running.	dbcont
None	**Go Until Cursor**	Continue execution of M-file until the line where the cursor is positioned. Also available on the context menu.	None

Toolbar Button	Debug Menu Item	Description	Function Alternative
	Step	Execute the current line of the M-file.	`dbstep`
	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`
	Step Out	After stepping in, run the rest of the called function or subfunction, leave the called function, and pause.	`dbstep out`

Continue Running in the Example

In the example, `collatzplot` is paused at line 10. Because the problem results are correct for `N/n = 1`, continue running until `N/n = 2`. Press the Continue button three times to move through the breakpoints at lines 10, 11, and 12. Now the program is again paused at the breakpoint at line 10.

Stepping In to Called Function in the Example

Now that `collatzplot` is paused at line 10 during the second iteration, use the Step In button or type `dbstep in` in the Command Window to step into `collatz` and walk through that M-file. Stepping into line 10 of `collatzplot` goes to line 9 of `collatz`. If `collatz` is not open in the Editor/Debugger, it automatically opens if you have selected **Debug > Open M-Files When Debugging**.

The pause indicator at line 10 of `collatzplot` changes to a hollow arrow ⇨, indicating that MATLAB control is now in a subfunction called from the main program. The call stack shows that the current function is now `collatz`.

In the called function, `collatz` in the example, you can do the same things you can do in the main (calling) function—set breakpoints, run, step through, and examine values.

Examining Values

While the program is paused, you can view the value of any variable currently in the workspace. Examine values when you want to see whether a line of code has produced the expected result or not. If the result is as expected, continue running or step to the next line. If the result is not as expected, then that line, or a previous line, contains an error. Use the following methods to examine values:

- "Selecting the Workspace" on page 6-114
- "Viewing Values as Datatips in the Editor/Debugger" on page 6-115
- "Viewing Values in the Command Window" on page 6-115
- "Viewing Values in the Workspace Browser and Array Editor" on page 6-116
- "Evaluating a Selection" on page 6-117
- "Examining Values in the Example" on page 6-117

Many of these methods are used in "Examining Values in the Example" on page 6-117.

Selecting the Workspace

Variables assigned through the Command Window and created using scripts are considered to be in the base workspace. Variables created in a function belong to their own function workspace. To examine a variable, you must first select its workspace. When you run a program, the current workspace is shown in the **Stack** field. To examine values that are part of another workspace for a currently running function or for the base workspace, first select that workspace from the list in the **Stack** field.

If you use debugging functions from the Command Window, use `dbstack` to display the call stack. Use `dbup` and `dbdown` to change to a different workspace. Use `who` or `whos` to list the variables in the current workspace.

Workspace in the Example. At line 10 of `collatzplot`, you stepped in, so the current line is 9 in `collatz`. The **Stack** shows that `collatz` is the current workspace.

Viewing Values as Datatips in the Editor/Debugger

In the Editor/Debugger, position the pointer to the left of a variable on that line. Its current value appears — this is called a datatip, which is like a tooltip for data. The datatip stays in view until you move the pointer. If you have trouble getting the datatip to appear, click in the line and then move the pointer next to the variable.

A related function is datatipinfo.

Datatips in the Example. Position the pointer over n in line 9 of collatz. The datatip shows that n = 2, as expected.

Hold the pointer over a variable. Its current value temporarily displays as a datatip.

Viewing Values in the Command Window

You can examine values while in debug mode at the K>> prompt. To see the variables currently in the workspace, use who. Type a variable name in the

Command Window and MATLAB displays its current value. For the example, to see the value of n, type

```
n
```

MATLAB returns the expected result

```
n =
  2
```

and displays the debug prompt, K>>.

Viewing Values in the Workspace Browser and Array Editor

You can view the value of variables in the **Value** column of the Workspace browser. The Workspace browser displays all variables in the current workspace. Use the **Stack** in the Workspace browser to change to another workspace and view its variables.
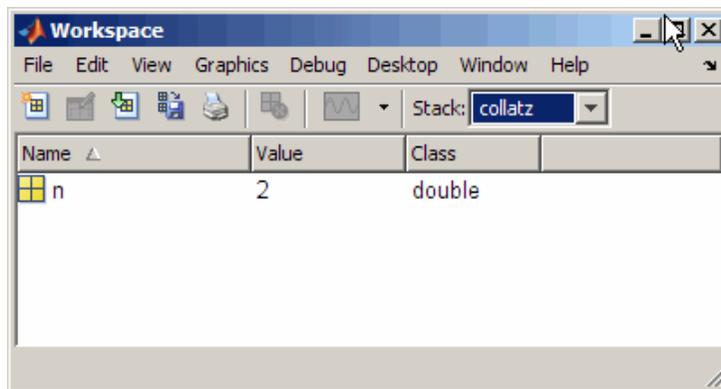
The **Value** column does not show all details for all variables. To see details, double-click a variable in the Workspace browser. The Array Editor opens, displaying the content for that variable. You can open the Array Editor directly for a variable using openvar.

To see the value of n in the Array Editor for the example, type

```
openvar n
```

and the Array Editor opens, showing that n = 2 as expected.
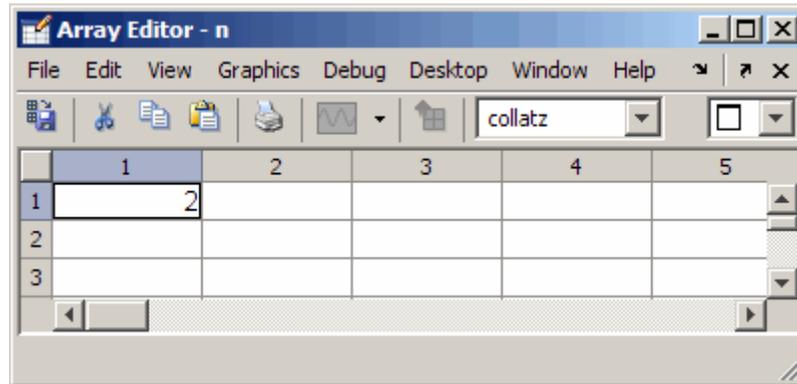
Evaluating a Selection

Select a variable or equation in an M-file in the Editor/Debugger. Right-click and select **Evaluate Selection** from the context menu (for a single-button mouse, use **Ctrl**+click). MATLAB displays the value of the variable or equation in the Command Window. You cannot evaluate a selection while MATLAB is busy, for example, running an M-file.

Examining Values in the Example

Step from line 9 through line 13 in collatz. Step again, and the pause indicator jumps to line 17, just after the if loop, as expected. Step again, to line 18, check the value of sequence in line 17 and see that the array is

    2   1

as expected for n = 2. Step again, which moves the pause indicator from line 18 to line 11. At line 11, step again. Because next_value is now 1, the while loop ends. The pause indicator is at line 11 and appears as a green down arrow ⬇. This indicates that processing in the called function is complete and program control will return to the calling program. Step again from line 11 in collatz and execution is now paused at line 10 in collatzplot.

Note that instead of stepping through collatz, the called function, as was just done in this example, you can step out from a called function back to the

calling function, which automatically runs the rest of the called function and returns to the next line in the calling function. To step out, use the Step Out button or type `dbstep out` in the Command Window.

In `collatzplot`, step again to advance to line 11, then line 12. The variable `seq_length` in line 11 is a vector with the elements

```
1   2
```

which is correct.

Finally, step again to advance to line 13. Examining the values in line 12, $N = 2$ as expected, but the second variable, `plot_seq`, has two values, where only one value is expected. While the value for `plot_seq` is as expected

```
2   1
```

it is the incorrect variable for plotting. Instead, `seq_length(N)` should be plotted.

Correcting Problems and Ending Debugging

These are some of the ways to correct problems and end the debugging session:

- "Changing Values and Checking Results" on page 6-118
- "Ending Debugging" on page 6-119
- "Disabling and Clearing Breakpoints" on page 6-119
- "Saving Breakpoints" on page 6-121
- "Correcting an M-File" on page 6-121
- "Completing the Example" on page 6-121
- "Running Sections in M-Files That Have Unsaved Changes" on page 6-125

Many of these features are used in "Completing the Example" on page 6-121.

Changing Values and Checking Results

While debugging, you can change the value of a variable in the current workspace to see if the new value produces expected results. While the

program is paused, assign a new value to the variable in the Command Window, Workspace browser, or Array Editor. Then continue running or stepping through the program. If the new value does not produce the expected results, the program has a different problem.

Ending Debugging

After identifying a problem, end the debugging session. You must end a debugging session if you want to change and save an M-file to correct a problem, or if you want to run other functions in MATLAB.

Note It is recommended that you quit debug mode before editing an M-file. If you edit an M-file while in debug mode, you can get unexpected results when you run the file. If you do edit an M-file while in debug mode, breakpoints turn gray, indicating that results might not be reliable. See "Valid (Red) and Invalid (Gray) Breakpoints" on page 6-109 for details.

If you attempt to save an edited M-file while in debug mode, a dialog box appears allowing you to exit debug mode and save the file.

To end debugging, click the Exit Debug Mode button 🗐, or select **Exit Debug Mode** from the **Debug** menu.

You can instead use the function dbquit or the **Shift+F5** keyboard shortcut to end debugging.

After quitting debugging, pause indicators in the Editor/Debugger display no longer appear, and the normal prompt >> appears in the Command Window instead of the debugging prompt, K>>. You can no longer access the call stack.

Disabling and Clearing Breakpoints

Disable a breakpoint to temporarily ignore it. Clear a breakpoint to remove it.

Disabling and Enabling Breakpoints. You can disable selected breakpoints so the program temporarily ignores them and runs uninterrupted, for example, after you think you identified and corrected a problem. This is especially useful for conditional breakpoints — see "Conditional Breakpoints" on page 6-126.

To disable a breakpoint, right-click the breakpoint icon and select **Disable Breakpoint** from the context menu, or click anywhere in a line and select **Enable/Disable Breakpoint** from the **Debug** or context menu. You can also disable a conditional breakpoint by clicking the breakpoint icon. This puts an X through the breakpoint icon as shown here.

Disabled breakpoint   10 ⊗   plot_seq = collatz(N);

After disabling a breakpoint, you can enable it to make it active again, or clear it. To enable it, right-click the breakpoint icon and select **Enable Breakpoint** from the context menu, or click anywhere in a line and select **Enable/Disable Breakpoint** from the **Breakpoints** or context menu. The X no longer appears on the breakpoint icon and program execution will pause at that line.

When you run dbstatus, the resulting message for a disabled breakpoint is

```
Breakpoint on line 10 has conditional expression 'false'.
```

Clearing (Removing) Breakpoints. All breakpoints remain in a file until you clear (remove) them or until they are automatically cleared. Clear a breakpoint after determining that a line of code is not causing a problem.

To clear a breakpoint in the Editor/Debugger, click anywhere in a line and select **Set/Clear Breakpoint** from the **Debug** or context menu. The breakpoint for that line is cleared. Another way to clear a breakpoint is to click a standard breakpoint icon, or a disabled conditional breakpoint icon.

To clear all breakpoints in all files, select **Debug > Clear Breakpoints in All Files**, or click its equivalent button ⊠ on the toolbar.

The function that clears breakpoints is dbclear. To clear all breakpoints, use dbclear all. For the example, clear all of the breakpoints in collatzplot by typing

```
dbclear all in collatzplot
```

Breakpoints are automatically cleared when you

- End the MATLAB session
- Clear the M-file using `clear name` or `clear all`

Note When `clear name` or `clear all` is in a statement in an M-file that you are debugging, it clears the breakpoints.

Saving Breakpoints

You can use the `s=dbstatus` syntax and then save `s` to save the current breakpoints to a MAT-file. At a later time, you can load `s` and restore the breakpoints using `dbstop(s)`. For more information, including an example, see the `dbstatus` reference page.

Correcting an M-File

To correct a problem in an M-file,

1 Quit debugging.

Do not make changes to an M-file while MATLAB is in debug mode. If you do edit an M-file while in debug mode, breakpoints turn gray, indicating that results might not be reliable. See"Valid (Red) and Invalid (Gray) Breakpoints" on page 6-109 for details.

2 Make changes to the M-file.

3 Save the M-file.

4 Set, disable, or clear breakpoints, as appropriate.

5 Run the M-file again to be sure it produces the expected results.

Completing the Example

To correct the problem in the example, do the following:

1 End the debugging session. One way to do this is to select **Exit Debug Mode** from the **Debug** menu.

2 In `collatzplot.m` line 12, change the string `plot_seq` to `seq_length(N)` and save the file.

3 Clear the breakpoints in `collatzplot.m`. One way to do this is by typing

    dbclear all in collatzplot

in the Command Window.

4 Run `collatzplot` for `m = 3` by typing

    collatzplot(3)

in the Command Window.

5 Verify the result. The figure shows that the length of the Collatz series is 1 when `n = 1`, 2 when `n = 2`, and 8 when `n = 3`, as expected.
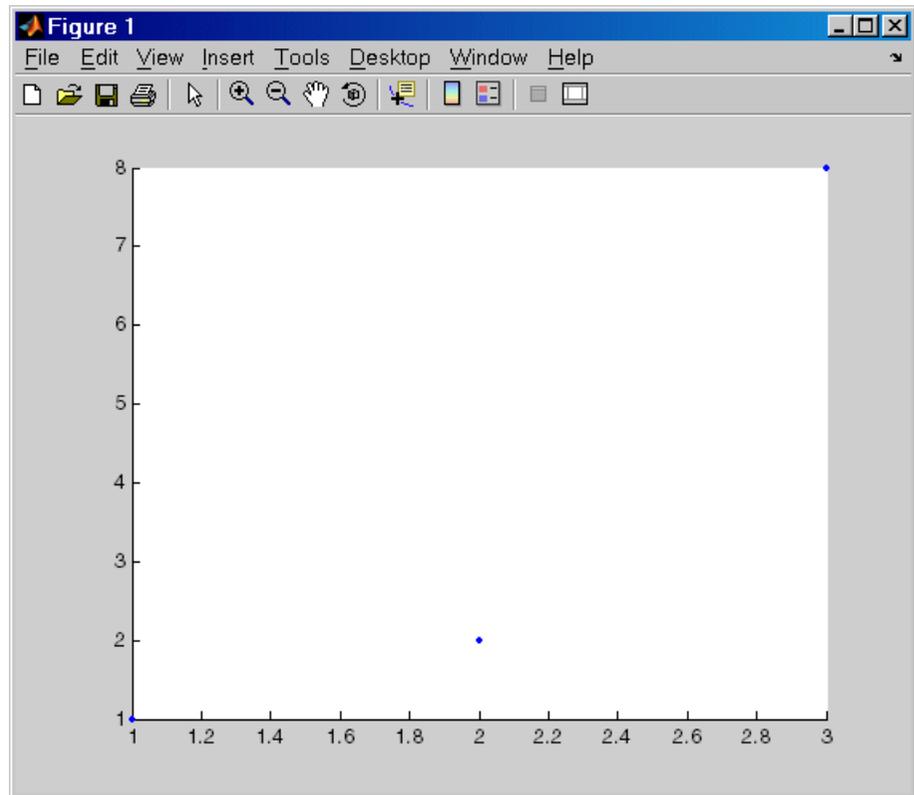
6 Test the function for a slightly larger value of m, such as 6, to be sure the
   results are still accurate. To make it easier to verify collatzplot for m = 6
   as well as the results for collatz, add this line at the end of collatz.m

   ```
   sequence
   ```

   which displays the series in the Command Window. The results for when
   n = 6 are

   ```
   sequence =

        6     3    10     5    16     8     4     2     1
   ```

   Then run collatzplot for m = 6 by typing

```
collatzplot(6)
```

7 To make debugging easier, you ran `collatzplot` for a small value of `m`. Now that you know it works correctly, run `collatzplot` for a larger value to produce more interesting results. Before doing so, you might want to disable output for the line you just added in step 6, line 19 of `collatz.m`, by adding a semicolon to the end of the line so it appears as

```
sequence;
```

Then run

```
collatzplot(500)
```

The following figure shows the lengths of the Collatz series for n = 1 through n = 500.
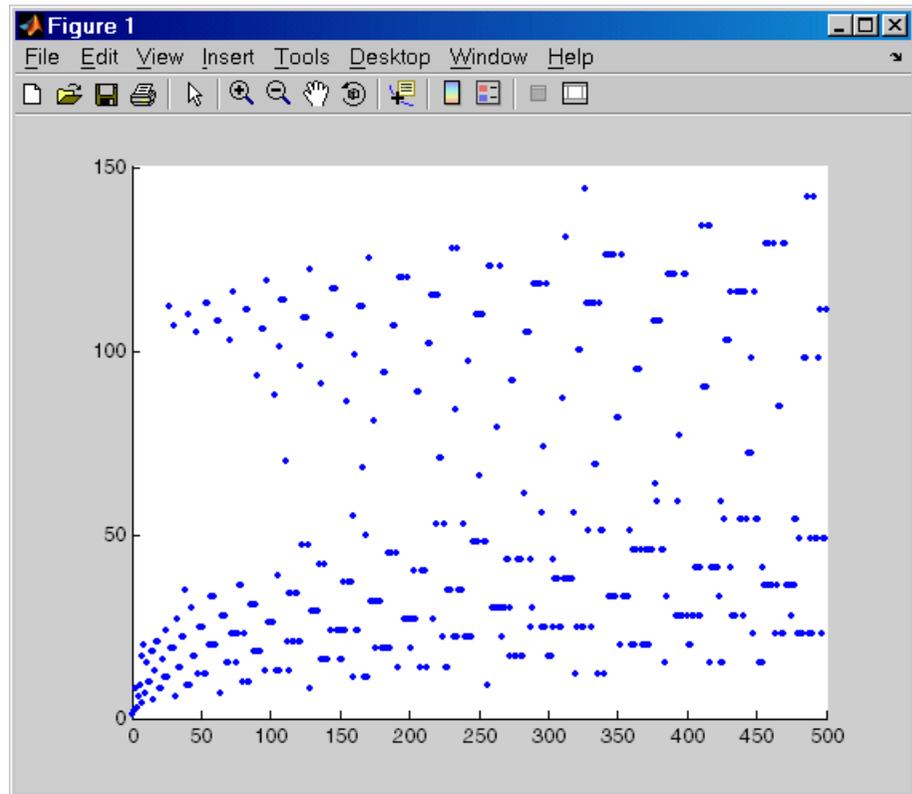
Running Sections in M-Files That Have Unsaved Changes

It is a good practice to make changes to an M-file after you quit debugging, and to save the changes and then run the file. Otherwise, you might get unexpected results. But there are situations where you might want to experiment during debugging, to make a change to a part of the file that has not yet run, and then run the remainder of the file without saving the change.

To do this, while stopped at a breakpoint, make a change to a part of the file that has not yet run. Breakpoints will turn gray, indicating they are invalid. Then select all of the code after the breakpoint, right-click, and

choose **Evaluate Selection** from the context menu. You can also use cell mode to do this.

Conditional Breakpoints

Set conditional breakpoints to cause MATLAB to stop at a specified line in a file only when the specified condition is met. One particularly good use for conditional breakpoints is when you want to examine results after a certain number of iterations in a loop. For example, set a breakpoint at line 10 in collatzplot, specifying that MATLAB should stop only if N is greater than or equal to 2. This section covers the following topics:

- "Setting Conditional Breakpoints" on page 6-126

- "Copying, Modifying, Disabling, and Clearing Conditional Breakpoints" on page 6-128

- "Function Alternative for Conditional Breakpoints" on page 6-128

Setting Conditional Breakpoints

To set a conditional breakpoint, follow these steps:

1 Click in the line where you want to set the conditional breakpoint. Then select **Set/Modify Conditional Breakpoint** from the **Debug** or context menu. If a standard breakpoint already exists at that line, use this same method to make it conditional.

The **MATLAB Editor** conditional breakpoint dialog box opens as shown in this example.
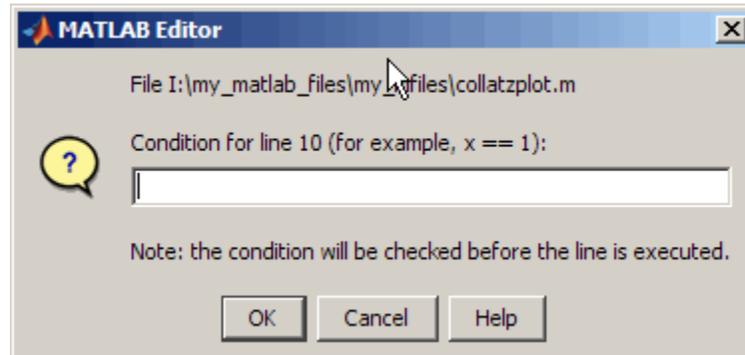
2 Type a condition in the dialog box, where a condition is any legal MATLAB expression that returns a logical scalar value. Click **OK**. As noted in the dialog box, the condition is evaluated before running the line. For the example, at line 10 in `collatzplot`, enter

```
N>=2
```

as the condition. A yellow breakpoint icon (indicating the breakpoint is conditional) appears in the breakpoint alley at that line.

Conditional breakpoint (yellow).

```
 9 +    for N = 1:m
10 ◊        plot_seq = collatz(N);
11 -        seq_length(N) = length(plot_seq);
12 -        line(N,plot_seq,'Marker','.','MarkerSize',9,'Color','blue')
13 -        drawnow
14 -    end
```

When you run the file, MATLAB enters debug mode and pauses at the line only when the condition is met. In the `collatzplot` example, MATLAB runs through the `for` loop once and pauses on the second iteration at line 10 when N is 2. If you continue executing, MATLAB pauses again at line 10 on the third iteration when N is 3.

Copying, Modifying, Disabling, and Clearing Conditional Breakpoints

To copy a conditional breakpoint, right-click the icon in the breakpoint alley and select **Copy** from the context menu. Then right-click in the breakpoint alley at the line where you want to paste the conditional breakpoint and select **Paste** from the context menu.

Modify the condition for the breakpoint in the current line by selecting **Set/Modify Conditional Breakpoint** from the **Debug** or context menu.

Click a conditional breakpoint icon to disable it. Click a disabled conditional breakpoint to clear it.

Function Alternative for Conditional Breakpoints

Use the dbstop function with appropriate arguments to set conditional breakpoints from the Command Window, and use dbclear to clear them. Use dbstatus to view the breakpoints currently set, including any conditions, which are listed in the expression field. If no condition exists, the value in the expression field is [   ] (empty). For details, see the function reference pages: dbstop, dbclear, and dbstatus.

Breakpoints in Anonymous Functions

There can be multiple breakpoints in an M-file line that contains anonymous functions. There can be a breakpoint for the line itself (MATLAB stops at the start of the line), as well as a breakpoint for each anonymous function in that line. When you add a breakpoint to a line containing an anonymous function, the Editor/Debugger asks exactly where in the line you want to add the breakpoint. If there is more than one breakpoint in a line, the breakpoint icon is blue ●.

When there are multiple breakpoints set on a line, the icon is always blue, regardless of the status of any of the breakpoints on the line. Position the mouse on the icon and a tooltip displays information about all breakpoints in that line.

To perform a breakpoint action for a line that can contain multiple breakpoints, such as **Clear Breakpoint**, right-click the breakpoint alley at

that line and select the action. MATLAB prompts you to specify the exact breakpoint on which to act in that line.

When you set a breakpoint in an anonymous function, MATLAB stops when the anonymous function is called. The following illustration shows the Editor/Debugger when you set a breakpoint in the anonymous function sqr in line 2, and then run the file. MATLAB stops when it runs sqr in line 4. After you continue execution, MATLAB stops again when it runs sqr the second time in line 4. Note that the **Stack** display shows the anonymous function.

Stack shows anonymous function.

Breakpoint set in anonymous function sqr.

MATLAB stops when it runs sqr.


```
function c = pythag(a,b)
sqr = @(x) x.^2
disp a,b
c = sqrt(sqr(a)+sqr(b))
disp c
```
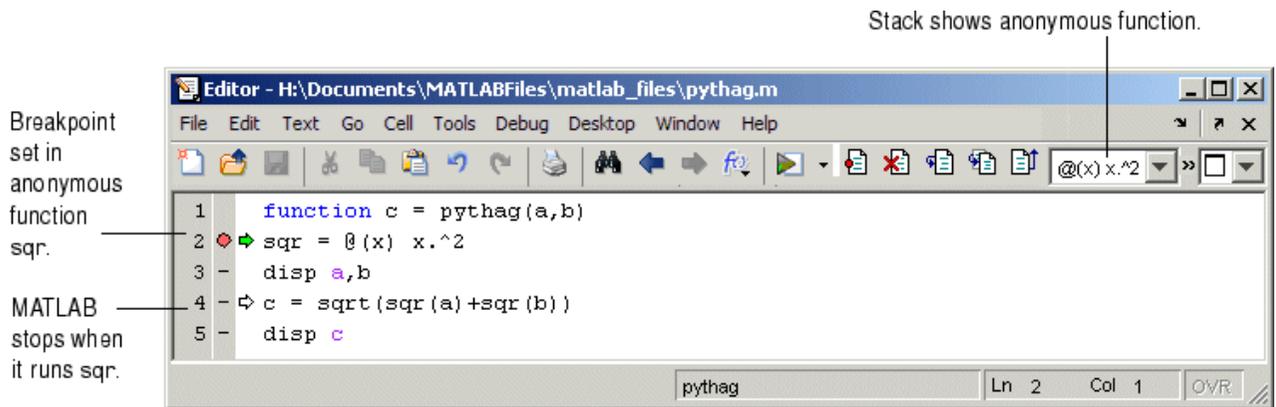
Error Breakpoints

Set error breakpoints to stop program execution and enter debug mode when MATLAB encounters a problem. Unlike standard and conditional breakpoints, you do not set these breakpoints at a specific line in a specific file. Rather, once set, MATLAB stops at any line in any file when the error condition specified via the error breakpoint occurs. MATLAB then enters debug mode and opens the file containing the error, with the pause indicator at the line containing the error. Files open only when the you select **Debug > Open M-Files** . Error breakpoints remain in effect until you clear them or until you end the MATLAB session. You can set error breakpoints from the **Debug** menu in any desktop tool. This section covers the following topics:

- "Setting Error Breakpoints" on page 6-130
- "Error Breakpoint Types and Options" on page 6-130

• "Function Alternative for Error Breakpoints" on page 6-132

Setting Error Breakpoints

To set error breakpoints, select **Debug > Stop if Errors/Warnings**. In the resulting **Stop if Errors/Warnings for All Files** dialog box, specify error breakpoints on all appropriate tabs and click **OK**. To clear error breakpoints, select the **Never stop if ...** option for all appropriate tabs and click **OK**.
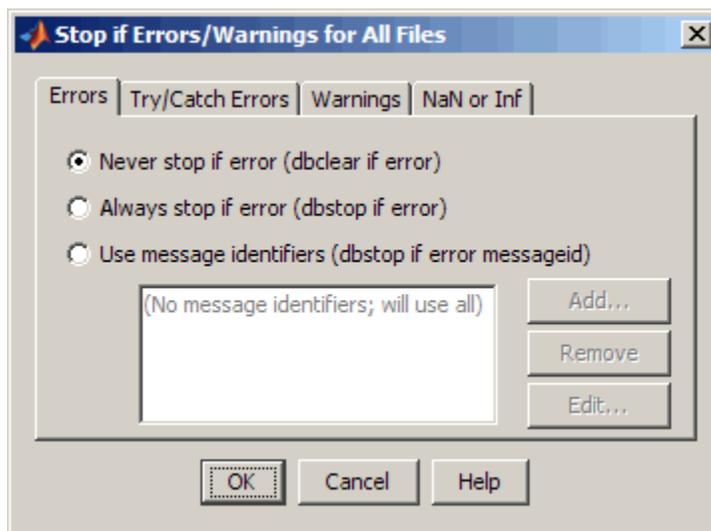
For example, to pause execution when a warning occurs, select the **Warnings** tab, and from it select **Always stop if warning**, then click **OK**. When you run an M-file and MATLAB produces a warning, execution pauses, MATLAB enters debug mode, and the file opens in the Editor/Debugger at the line that produced the warning. To remove the warning breakpoint, select **Never stop if warning** in the **Warnings** tab and click **OK**.

Error Breakpoint Types and Options

The four basic types of error breakpoints you can set are **Errors**, **Try/Catch Errors**, **Warnings**, and **NaN or Inf**. Select the **Always stop if ...** option for each tab to set that type of breakpoint. Select the **Use message identifiers ...** option to limit each type of error breakpoint (except NaN or Inf) so that execution stops only for specific errors.

Errors. When an error occurs, execution stops, unless the error is in a `try...catch` block. MATLAB enters debug mode and opens the M-file to the line in the `try` portion of the block that produced the error. You cannot resume execution.

Try/Catch Errors. When an error occurs in a `try...catch` block, execution pauses. MATLAB enters debug mode and opens the M-file to the line that produced the error. You can resume execution or use debugging features.

Warnings. When a warning is produced, MATLAB pauses, enters debug mode, and opens the M-file, paused at the line that produced the warning. You can resume execution or use debugging features.

NaN or Inf. When an operator, function call, or scalar assignment produces a `NaN` (not-a-number) or `Inf` (infinite) value, MATLAB pauses, enters debug mode, and opens the M-file, paused immediately after the line that encountered the value. You can resume execution or use debugging features.

Use Message Identifiers. Execution stops only when MATLAB encounters one of the specified errors. This option is not available for the **NaN or Inf** type of error breakpoint. To use this feature:

1 Select the **Errors**, **Try/Catch Errors,** or **Warnings** tab.

2 Select the **Use Message Identifiers** option.

3 Click **Add**.

4 In the resulting **Add Message Identifier** dialog box, supply the message identifier of the specific error you want to stop for, where the identifier is of the form `component:message`, and click **OK**.

5 The message identifier you added appears in the **Stop If Errors/Warnings for All Files** dialog box, where you click **OK**.

You can add multiple message identifiers, and edit or remove them.

One way to obtain an error message identifier generated by a MATLAB function for example, is to produce the error, and then run the `lasterror` function. MATLAB returns the error message and identifier. Copy the identifier from the Command Window output and paste it into the **Add**

Message Identifier dialog box. An example of an error message identifier is `MATLAB:UndefinedFunction`. Similarly, to obtain a warning message identifier, produce the warning and then run `[m,id] = lastwarn;` MATLAB returns the last warning identifier to `id`. An example of a warning message identifier is `MATLAB:divideByZero`.

Function Alternative for Error Breakpoints

The function equivalent for each option appears in the **Stop if Errors/Warnings for All Files** dialog box. For example, the function equivalent for **Always stop if error** is `dbstop if error`. Use the `dbstop` function with appropriate arguments to set error breakpoints from the Command Window, and use `dbclear` to clear them. Use `dbstatus` to view the error breakpoints currently set. Error breakpoints are listed in the `cond` field and message identifiers for breakpoints are listed in the `identifier` field of the `dbstatus` output.

Using Cells for Rapid Code Iteration and Publishing Results

What Are Cells?

M-files often have a natural structure consisting of multiple sections. Especially for larger files, you typically focus efforts on a single section at a time, working with the code in just that section. Similarly, when conveying information about your M-files to others, often you describe the sections of the code. To facilitate these processes, use M-file *cells*, where *cell* means a section of code. Specifically, MATLAB uses cells for

- Rapid code iteration in the Editor/Debugger — This makes the experimental phase of your work with M-file scripts easier. The next section, "Rapid Code Iteration Overview" on page 6-133, outlines the process, and is followed by details for defining, evaluating, and modifying values in cells.

- Publishing M-files — This allows you to include code and results in a presentation format such as HTML. Publishing using cells requires you to define cells using the same instructions as for rapid code iteration. You can also make use of the cell navigation and evaluation features used for rapid code iteration. See "Publishing to HTML, XML, LaTeX, Word, and PowerPoint Using Cells" on page 8-2 for complete details.

Rapid Code Iteration Overview

When working with MATLAB, you often experiment with your code — modifying it, testing it, and updating it — until you have an M-file that does what you want. Use the MATLAB Editor/Debugger cells features with M-file scripts to facilitate this process. You can also use cell features with function M-files, but there are some restrictions — see "Using Cells in Function M-Files" on page 6-144.

If you have an active Internet connection, you can watch the Rapid Code Iteration Using Cells video demo for an overview of the major functionality.

This is the overall process of using cells for rapid code iteration:

1 In the MATLAB Editor/Debugger, enable cell mode by selecting **Cell > Enable Cell Mode**. Items in the **Cell** menu become selectable. The cell toolbar appears, unless you had previously hidden it. With cell mode enabled, hide or show the toolbar by right-clicking in the Editor/Debugger menu bar or toolbars and selecting **Cell Toolbar** from the context menu.

2 Define the boundaries of the cells in an M-file script using cell features. Cells are denoted by a specialized comment syntax, %%. For details, see "Defining Cells" on page 6-135.

3 Once you define the cells, use cell features to navigate quickly from cell to cell in your file, evaluate the code in a cell in the base workspace, and view the results. To facilitate experimentation, use cell features to modify values in cells and then reevaluate them, to see how different values impact the result. For details, see "Navigating and Evaluating with Cells" on page 6-139.

4 Cells also facilitate sharing your code and results via cell publishing to a presentation format. For details, see "Publishing to HTML, XML, LaTeX, Word, and PowerPoint Using Cells" on page 8-2.

Cell features.

Cell toolbar.

Current cell is highlighted.

Use display preference to show horizontal lines between cells.
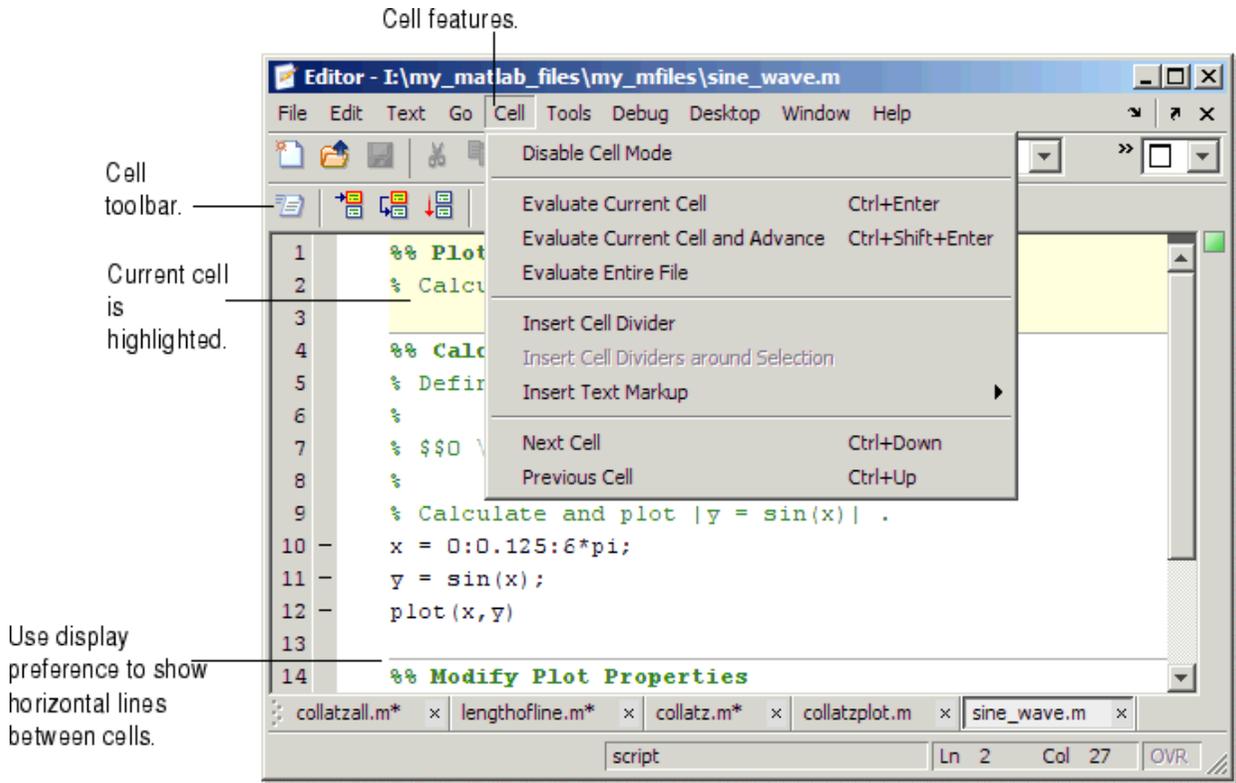
Defining Cells

Cell features operate on cells, where a cell is contiguous lines of code you want to evaluate as a whole in an M-file script. To define a cell, first be sure that cell mode is enabled (see step). Position the cursor just before the line at which you want to start the cell and select **Cell > Insert Cell Divider** or click the Insert Cell Divider button . MATLAB inserts a line after the cursor that consists of two percent signs (%%), which is the "start new cell" indicator to MATLAB. A cell consists of the line starting with %% and the lines that follow, up to the start of the next cell, which is identified by %% at the start of a new line.

You can also define a cell by entering two percent signs (%%) at the start of the line where you want to begin the new cell. Alternatively, select the

lines of code you want in a cell and then select **Cell > Insert Cell Dividers Around Selection**.

You can define a cell at the start of a new empty file, enter code for the cell, define the start of the next cell, enter its code, and so on. Redefine cells by defining new cells, removing existing cells, and moving lines of code.

You can set an Editor/Debugger preference to show a faint gray horizontal line (rule) above each cell that helps distinguish the cells. Select **File > Preferences > Editor/Debugger > Display** and in **Cell display options**, use **Show lines between cells**. The horizontal lines do not appear in the M-file when you print it.

MATLAB does not execute the code in lines beginning with %%, so be sure to put any executable code for the cell on the following line. For program control statements, such as if ... end, a cell must contain both the opening and closing statements, that is, it must contain both the if and the end statements.

Note that the first cell in a file does not have to begin with %%. MATLAB automatically understands any lines above the first %% line to be a cell. If there are no cell dividers in an M-file, MATLAB understands the entire file to be a single cell.

Cell Titles and Highlighting

After the %%, type a space, followed by a description of the cell. The Editor/Debugger emphasizes the special meaning of the start of a cell by making any text following the percent signs appear bold. The text on the %% line is called the *cell title* (like a section title). Including text in cell titles is optional, however, they improve readability of the file and are used for cell publishing features.

When the cursor is positioned in any line within a cell, the Editor/Debugger highlights the entire cell with a yellow background. This identifies it as the *current cell*. For example, it is used when you select the **Evaluate Current Cell** option on the **Cell** menu. If you do not want yellow highlighting for the current cell, change it using preferences. Select **File > Preferences > Editor/Debugger > Display** and change the appropriate **Cell display options**.

Example — Define Cells

This example defines two cells for a simple M-file called sine_wave, shown in the following code and figure. The first cell creates the basic results, while the second label the plot. The two cells in this example allow you to experiment with the plot of the data first, and then when that is final, change the plot properties to affect the style of presentation.

```
% Define the range for x.
% Calculate and plot y = sin(x).
x = 0:1:6*pi;
y = sin(x);
plot(x,y)
title('Sine Wave','FontWeight','bold')
xlabel('x')
ylabel('sin(x)')
set(gca,'Color','w')
set(gcf, 'MenuBar', 'none')
```

M-file before defining cells.

1 Select **Cell > Enable Cell Mode**, if it is not already enabled.

2 Position the cursor at the start of the first line. Select **Cell > Insert Cell Divider**.

The Editor/Debugger inserts `%%` as the first line and moves the rest of the file down one line. All lines are highlighted in yellow, indicating that the entire file is a single cell, unless you do not have that display preference for cells selected.

3 Enter a cell title following the `%%`. Type a space first, followed by the description.

```
Calculate and Plot Sine Wave
```

4 Position the cursor at the start of line 7, `title....` Select **Cell > Insert Cell Divider**.

The Editor/Debugger inserts a line containing only `%%` at line 7 and moves the remaining lines down by one line. A horizontal line that helps you distinguish the two cells appears above the `%%` line, unless you do not have that display preference for cells selected. Lines 7 through 12 are highlighted in yellow, indicating they comprise the current cell.

5 Enter a cell title for the new cell. On line 7, type a space after the `%%`, followed by the description

```
Modify Plot Properties
```

Save the file. The file appears as shown in this figure.

M-file after defining cells.

Use preferences to show horizontal lines between cells and highlighting of current cell


```
1    %% Calculate and Plot Sine Wave
2    % Define the range for x.
3    % Calculate and plot y = sin(x).
4  - x = 0:1:6*pi;
5  - y = sin(x);
6  - plot(x,y)
7    %% Modify Plot Properties
8  - title('Sine Wave','FontWeight','bold')
9  - xlabel('x')
10 - ylabel('sin(x)')
11 - set(gca,'Color','w')
12 - set(gcf, 'MenuBar', 'none')
```

Removing Cells

To remove a cell, delete one of the percent signs (%) from the line that starts the cell. This changes the line from a cell to a standard comment and merges the cell with the preceding cell. You can also just delete the entire line that contains the %%.

Navigating and Evaluating with Cells

While you develop an M-file, you can use these Editor/Debugger cell features:

- "Navigating Among Cells in an M-File" on page 6-140

- "Evaluating Cells in an M-File" on page 6-140

- "Modifying Values in a Cell" on page 6-141

- "Example — Evaluate Cells" on page 6-141

Navigating Among Cells in an M-File

To move to the next cell, select **Cell > Next Cell**. To move to the previous cell, select **Cell > Previous Cell**. To move to a specific cell, click the Show Cell Titles button ![icon] and from it, select the cell title to which you want to move. You can also go to cells by selecting **Edit > Go To**.

Evaluating Cells in an M-File

To evaluate the code in a cell, use the **Cell** menu evaluation items or equivalent buttons in the cell toolbar. When you evaluate a cell, the results display in the Command Window, figure window, or otherwise, depending on the code evaluated.

The cell evaluation features run the code currently shown in the Editor/Debugger, even if the file contains unsaved changes. The file does not have to be on the search path. To evaluate a cell, it must contain all the values it requires, or the values must already exist in the MATLAB workspace.

Note While you can set breakpoints and debug a file containing cells, when you evaluate a file from the **Cell** menu or cell toolbar, breakpoints are ignored. To run the file and stop at breakpoints, use **Run/Continue** in the **Debug** menu. This means you cannot debug while running a single cell.

Evaluate Current Cell. Select **Cell > Evaluate Current Cell** or click the Evaluate Cell button ![icon] to run the code in the current cell.

Evaluate and Advance. Select **Cell > Evaluate Current Cell and Advance** or click the Evaluate Cell and Advance button ![icon] to run the code in the current cell and move to the next cell.

Evaluate File. Select **Cell > Evaluate Entire File** or click the Evaluate Entire File button ![icon] to run all of the code in the file.

Note A beep means there is an error. See the Command Window for the error message.

Modifying Values in a Cell

You can use cell features to modify numbers in a cell, which also automatically reevaluates the cell. This helps you experiment with and fine-tune your code.

To modify a number in a cell, select the number (or place the cursor near it) and use the value modification tool in the cell toolbar. Using this tool, you can specify a number and press the appropriate math operator to add (increment), subtract (decrement), multiply, or divide the number. The cell then automatically reevaluates.
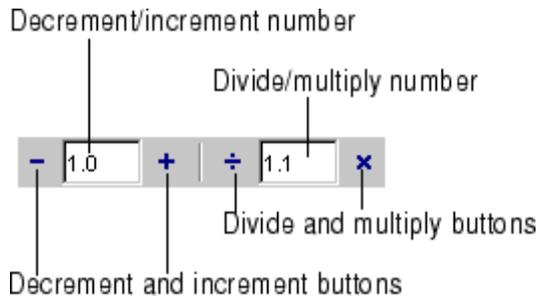
You can use the numeric keypad operator keys (**-**, **+**, **/**, and *) instead of the operator buttons on the toolbar.

Note MATLAB does not automatically save changes you make to values using the cell toolbar. To save changes, select **File > Save**.

Example — Evaluate Cells

In this example, modify the values for x in sine_wave.m:

1 Run the first cell in sine_wav.m. Click somewhere in the first cell, that is, between lines 1 and 6. Select **Cell > Evaluate Current Cell**. The following figure appears.

Plot generated by
running
sine_wave.m.
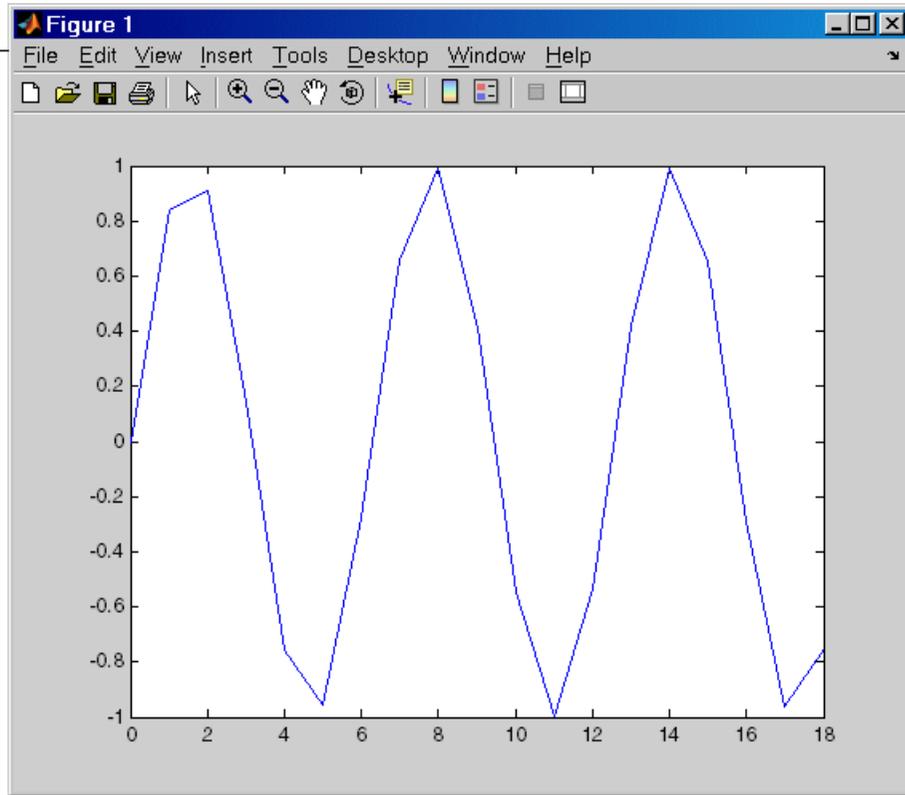
2 Assume you want to produce a smoother curve. Use more values for x in 0:1:6*pi. Position the cursor in line 4, next to the 1. In the cell toolbar, change the 1.1 default multiply/divide by value to 2. Click the Divide button ÷.

Line 4 becomes

```
4 -    x = 0:0.5:6*pi;
```

and the length of x doubles. The plot automatically updates. The curve still has some rough edges.

3 To add more values for x, click the Divide button three more times. Line 4 becomes

```
4 -    x = 0:0.0625:6*pi;
```

The curve is smooth, but because there are more values, processing time is slower. It would be better to find a smaller x that still produces a smooth curve.

4 In the cell toolbar, click the Multiply button once. The increment for x as shown in line 4 changes from 0.0625 to 0.125.

The resulting curve is still smooth.

5 Save these changes. Select **File > Save**.

6 Now you can apply the plot properties, defined in the second cell, that is, lines 7 through 12. You do not need to evaluate the entire file to apply the plot properties. Instead, position the cursor in the second cell and use the shortcut **Ctrl+Enter** to evaluate the current cell. (The shortcut appears with the menu item, **Cell > Evaluate Current Cell**.)

MATLAB updates the figure.
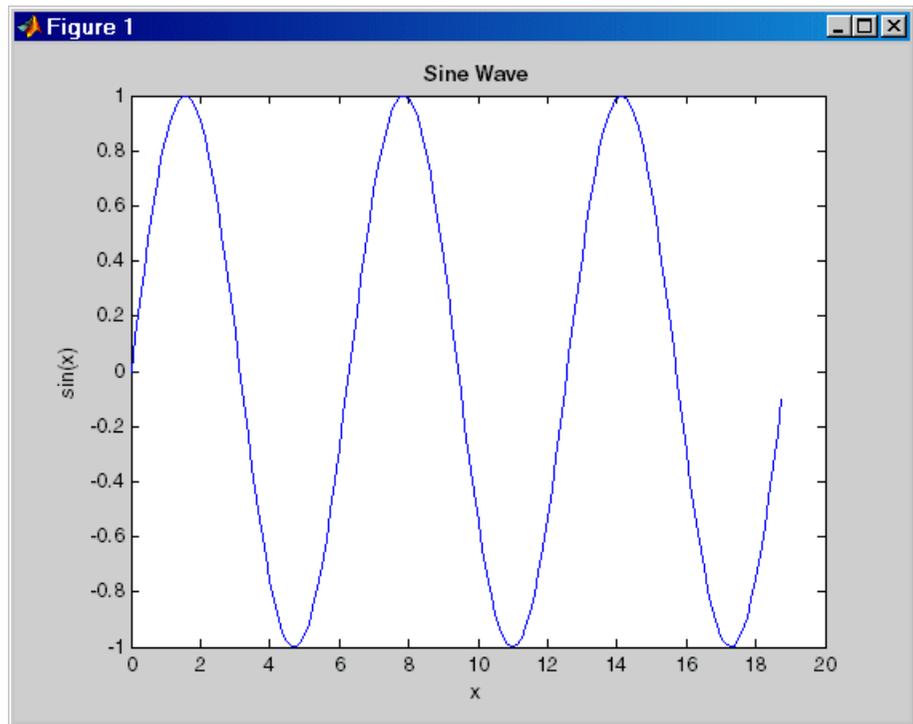
Using Cells in Function M-Files

You can define and evaluate cells in function M-files as long as the variables referred to in the cell are in your workspace. For example, this can be useful during debugging. If execution is stopped at a breakpoint, you can define cells and execute them without saving the file. If you are not debugging, add the necessary variables to the base workspace and then execute the cells.

Tuning and Managing M-Files

This set of tools provides useful information about the M-files in a directory that can help you refine the files and improve performance. The tools can help you polish M-files before providing them to others to use. If you have an active Internet connection, you can watch the Directory Reports video demo for an overview of the major functionality.

Directory Reports in Current Directory Browser

In this section...
"Accessing and Using Directory Reports" on page 7-2
"TODO/FIXME Report" on page 7-4
"Help Report" on page 7-6
"Contents Report" on page 7-9
"Dependency Report" on page 7-13
"Coverage Report" on page 7-15

See also another Directory Report, "M-Lint Code Check Report" on page 7-16, and the File Comparisons tool.

Accessing and Using Directory Reports

Directory reports help you refine the M-files in a directory and improve their performance. They are also useful for when you prepare files for use by others, such as for a finished project, to share on MATLAB Central, or for a toolbox to be distributed.

Access directory reports from the MATLAB Current Directory browser. To display the Current Directory browser, select **Desktop > Current Directory**. For more information, see "Current Directory Browser" on page 5-36.

Navigate to the directory whose M-files you want to produce reports about. Then, in the Current Directory browser toolbar, click the down arrow button and select the type of report you want to run for all the M-files in the current directory.
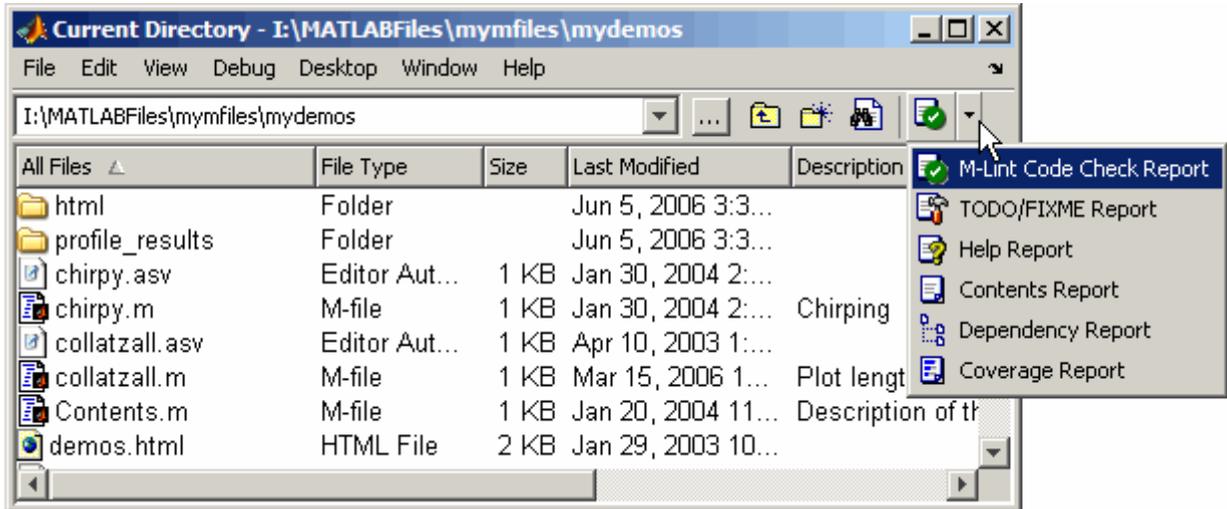
The report you selected appears as an HTML document in the MATLAB Web Browser:

- In a report, click a filename to open that file in the Editor/Debugger, where you can view it or make changes to it. Click a line number to open the file at that line.

- To update a report after making changes to the report options, or after changing any files in the directory, click **Rerun This Report**. Note that this reruns the report for the directory shown in the report, not for the MATLAB current directory.

- While a report is displayed, you can change the MATLAB current directory and then click **Run Report on Current Directory** to generate the same type of report for the new current directory.

- When you run a report, it replaces the report currently displayed. Use the Back ← and Forward → buttons in the toolbar to see a previously run report and then return to the most recent.

You cannot run directory reports when the path is a UNC (Universal Naming Convention) pathname, that is, starts with \\. Instead, use an actual hard drive on your system, or a mapped network drive.

TODO/FIXME Report

The TODO/FIXME Report shows M-files that contain text strings you included as notes to yourself, such as TODO. Use this report to easily identify M-files that still require work or some other actions.

To access this report, follow the instructions in "Accessing and Using Directory Reports" on page 7-2.

In the report, select one or more check boxes to display lines containing the specified strings (TODO and FIXME), and click **Rerun This Report**. You can also select the check box for the text field and enter any text string in the field, such as NOTE or TBD to identify lines containing that string.

The TODO/FIXME Report identifies lines containing specified text strings, such as TODO and NOTE shown here, so you can easily see reminders you included.

Enter any text string in this field.

Click the line number to open the file at that line in the Editor/Debugger, where you can make changes.

TODO/FIXME Report

File   Edit   View   Go   Debug   Desktop   Window   Help

TODO/FIXME Report

Rerun This Report   Run Report on Current Directory

☑ TODO   ☑ FIXME   ☑ NOTE

Report for directory D:\mymfiles\mydemos

M-File List

Contents	
chirpy	19 TODO: add features from image toolbox
collatzall	
fractal	
go	
logo5	
logo6	
logoimage	
moebius	
sift	12 NOTE: I can probably remove this loop
splash	

Help Report

The Help Report presents a summary view of the help component of your M-files. In MATLAB, the M-file help component is all contiguous nonexecutable lines (comment lines and blank lines), starting with the second line of a function M-file or the first line of a script M-file. For more information about creating help for your own M-files, see the reference page for the `help` function.

To access this report, follow the instructions in "Accessing and Using Directory Reports" on page 7-2.

Select one or more check boxes to display the specified help information and click **Rerun This Report**.

Use this information to help you identify files of interest or files that lack help information. It is a good practice to provide help for your files not only to help you recall their purpose, but to help others who might use the files.

Select **Description** to show the first help line.

Select **Show all help** to see the rest of the help.

Select **Show subfunctions** to see help information for subfunctions.

Show Subfunctions

With **Show subfunctions** selected, the Help Report displays help information for all subfunctions called by each function. Help information for subfunctions is highlighted in gray.

Description

With **Description** selected, the Help Report displays the first line of help in the M-file. If the first comment line is empty, or if there is not a comment before the executable code, **No description line**, highlighted in pink, appears instead.

Examples

With **Examples** selected, the Help Report displays the line number where the examples section of the M-file help begins. The Help Report looks for a line in the M-file help that begins with the string `example` or `Example` and displays any subsequent nonblank comment lines. Select this option to easily locate and go to examples in your M-files.

It is a good practice to include examples in the help for your M-files. If you do not have examples in the help for all your M-files, use this option to identify those without examples. If the report does not find examples in the M-file help, **No example**, highlighted in pink, appears.

Show All Help

With **Show all help** selected, the Help Report displays complete M-file help, which is all contiguous nonexecutable lines (comment lines and blank lines), starting with the second line of a function M-file, or the first line of a script M-file. The M-file help shown also includes overloaded functions and methods, which are not actually part of the M-file help comments, but are automatically generated when `help` runs.

If the comment lines before the executable code are empty, or if there are no comments before the executable code, **No help**, highlighted in pink, appears instead.

See Also

With **See Also** selected, the Help Report displays the line number for the see also line in the M-file help. The see also line in M-file help lists related functions. When MATLAB displays the help for an M-file, any function name listed on the see also line appears as a link you can click to display its help. It is a good practice to include a see also line in the help for your M-files.

The report looks for a line in the M-file help that begins with the string See also. If the report does not find a see also line in the M-file help, **No see-also line**, highlighted in pink, appears. This helps you identify those M-files without a see also line, should you want to include one in each M-file.

The report also indicates when an M-file noted in the see also line is not in a directory on the search path. You might want to move that file to a directory that is on the search path. If not, you will not be able to click the link to get help for the file, unless you then add its directory to the path or make its directory become the current directory.

Copyright

With **Copyright** selected, the Help Report displays the line number for the copyright line in the M-file. The report looks for a comment line in the M-file that begins with the string Copyright and is followed by year1-year2 (with no spaces between the years and the hyphen that separates them). It also notes if the end of the date range is not the current year.

It is a good practice to include a copyright line in the help for your M-files, that notes the year you created the file and the current year. For example, for an M-file you created in 2001, include this line

```
% Copyright 2001-2006
```

If the report does not find a copyright line in the M-file help, **No copyright line**, highlighted in pink, appears. This helps you identify those files without a copyright line, should you want to include one in each M-file.

Contents Report

The Contents Report displays information about the integrity of the Contents.m file for the directory. A Contents.m file includes the filename and a brief description of each M-file in the directory. When you type help followed by the directory name, such as help mydemos, MATLAB displays the information in the mydemos/Contents.m file. For more information, see "Providing Help for Your Program" in the MATLAB Programming documentation.

To access this report, follow the instructions in "Accessing and Using Directory Reports" on page 7-2.

If there is no `Contents.m` file for the directory and you run the Contents Report, the report tells you the `Contents.m` file does not exist and asks if you want to create one. Click **yes** to automatically create the `Contents.m` file. Edit the `Contents.m` file in the Editor/Debugger to include the names of files you plan to create, or to remove files that you do not want to expose when displaying help for the directory, such as files for internal use.

You need to update the `Contents.m` file to reflect changes you make to files in the directory. For example, when you remove a file from a directory, remove its entry from the `Contents.m` file. The Contents Report helps you to maintain the `Contents.m` file. It displays discrepancies between the `Contents.m` file and the M-files in the directory.
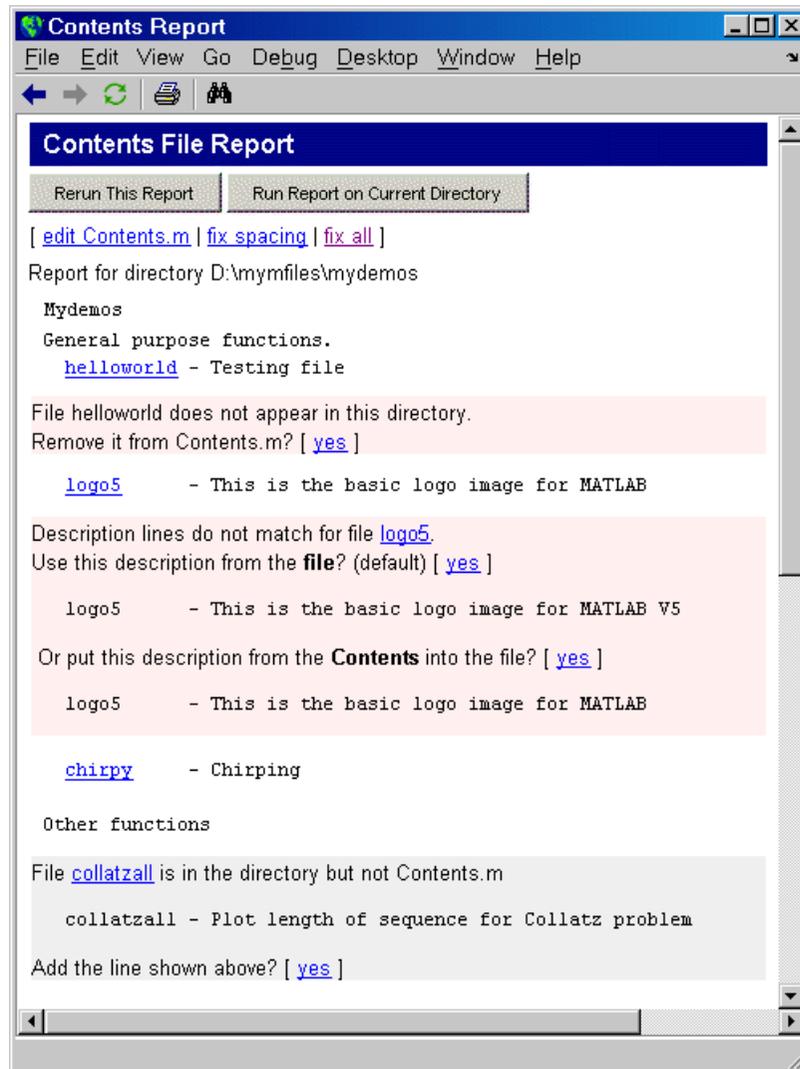
Use the links displayed for each line, or edit the Contents.m file directly, or edit the M-files to make the changes. To make all of the suggested changes at once, click **fix all**. To automatically align the filenames and descriptions in the Contents.m file, click **fix spacing**.

If you always want the Contents.m file to reflect all files in the directory, you can automatically generate a new Contents.m file rather than changing the file based on the Contents Report. To do this, first delete the existing Contents.m file, run the Contents Report, and click yes when prompted for MATLAB to automatically create one.

Messages in the Contents File Report

No Contents File. This message appears if there is no Contents.m file in the directory. Click **yes** to automatically create a Contents.m file, which contains the filenames and descriptions for all M-files in the directory.

```
No Contents.m file. Make one? [ yes ]
```

File Not Found. This message appears when a file included in Contents.m is not in the directory. These messages are highlighted in pink. For example, a message such as

```
File helloworld does not appear in this directory.
Remove it from Contents.m? [ yes ]
```

means the Contents.m file includes an entry for helloworld, but that file is not in the directory. This might be because you removed the file helloworld, or you manually added it to Contents.m because you planned to create the file but have not as yet, or you renamed helloworld.

Description Lines Do Not Match. This message appears when the description line in the M-file help does not match the description provided for the M-file in Contents.m. These messages are highlighted in pink. Click **yes** to replace the description in the Contents.m file with the description from the M-file. Or select the option to replace the description line in the M-file help using the description for that file in Contents.m.

```
Description lines do not match for file logo5.
Use this description from the file? (default) [ yes ]
 logo5      - This is the basic logo image for MATLAB V5
Or put this description from the Contents into the file? [ yes ]
 logo5 - This is the basic logo image for MATLAB
```

Files Not In Contents.m. This message appears when a file in the directory is not in `Contents.m`. These messages are highlighted in gray. Click **yes** to add the filename and its description line from the M-file help to the `Contents.m` file.

```
collatzall is in the directory but not Contents.m
 collatzall - Plot length of sequence for Collatz problem
Add the line shown above? [ yes ]
```

Dependency Report

The Dependency Report shows dependencies among M-files in a directory. This helps you determine all the M-files you need to provide when you tell someone to run a particular M-file. If you do not provide all the dependent M-files along with the M-file you want them to run, they will not be able run the file. The report does not list as dependencies the M-files in the `toolbox/matlab` directory because every MATLAB user already has those files.

To access this report, follow the instructions in "Accessing and Using Directory Reports" on page 7-2. You can also access the report from the Editor/Debugger **Tools** menu.

Select **Show child functions** to see a list of all M-files (children) called by each M-file in the directory (parent). The report also indicates where each child function resides, for example, in a specified toolbox. If a child function's location is listed as unknown, it could be because the child function is not on the search path or in the current directory.

The file `chirpy.m` calls two M-files in the Signal Processing Toolbox and one in the Image Processing Toolbox.

The file `go.m` calls `moebius.m`, located in the current directory.
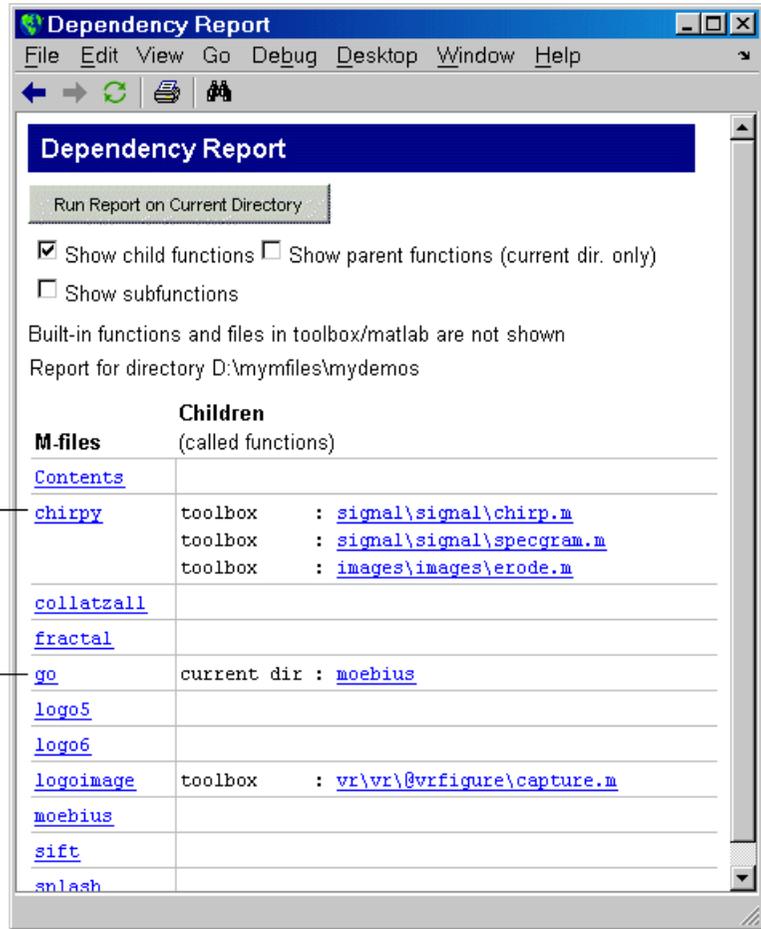
The Dependency Report is similar to running the `depfun` function, although the two do not provide the exact same results. For performance purposes, the Dependency Report limits the functions considered.

Select **Show parent functions** to list the M-files that call each M-file. The report limits the parent (calling) functions to those in the current directory. Select **Show subfunctions** to include subfunctions in the report. Subfunctions are listed directly after the main function and are highlighted in gray.

Coverage Report

Run the Coverage Report after you run the Profiler to identify how much of a file ran when it was profiled. For example, when you have an `if` statement in your code, that section might not run during profiling, depending on conditions.

You can run the Coverage Report from the Profiler, or follow these steps:

1 In the MATLAB desktop, select **Desktop > Profiler**. Profile an M-file in the Profiler. For detailed instructions, see "Profiling for Improving Performance" on page 7-27.

2 In the Current Directory browser, select `Coverage Report`. The **Coverage Report** appears, providing a summary of coverage for the M-file you profiled.

The Coverage Report shows the percentage of a file that ran when it was profiled.
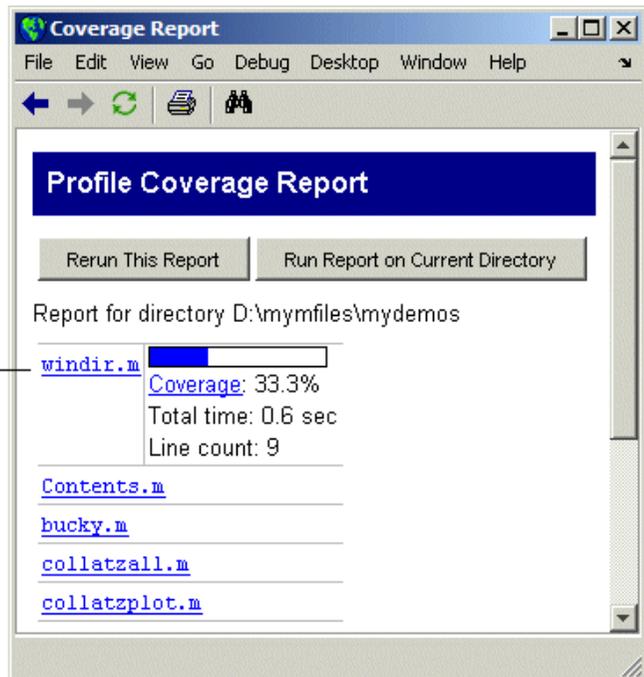
3 Click the **Coverage** link to see the Profile Detail Report for the file.

M-Lint Code Check Report

In this section...

Running the M-Lint Code Check Directory Report

The M-Lint Code Check Report displays potential errors and problems, as well as opportunities for improvement in your code. The term "lint" is the name given to similar tools used with other programming languages such as C. In MATLAB, M-Lint produces a message for each line of an M-file that it determines might be improved. For example, a common M-Lint message is that a variable foo in line 12 is defined but never used in the M-file.

To run the M-Lint code check directory report, follow these steps:

1 In the Current Directory browser, navigate to the directory that contains the M-files you want to check with M-Lint. To use the example shown in this documentation, lengthofline.m, you can change the current directory by running

```
cd(fullfile(matlabroot,'help','techdoc','matlab_env','examples'))
```

2 If you plan to modify the example, save the file to a directory for which you have write access, and then make that directory the current MATLAB directory. In this example, the file is saved to I:\MATLABFiles\mymfiles.

3 In the Current Directory browser toolbar, select the M-Lint Code Check Report from the Directory Reports listing—for details, see "Accessing and Using Directory Reports" on page 7-2.

The M-Lint Code Check Report displays in the MATLAB Web Browser, showing those M-files that M-Lint identified as having potential problems or opportunities for improvement.

Line number and message describing a potential problem or improvement opportunity.

Click a line number to open the M-file in the Editor/Debugger at the line.
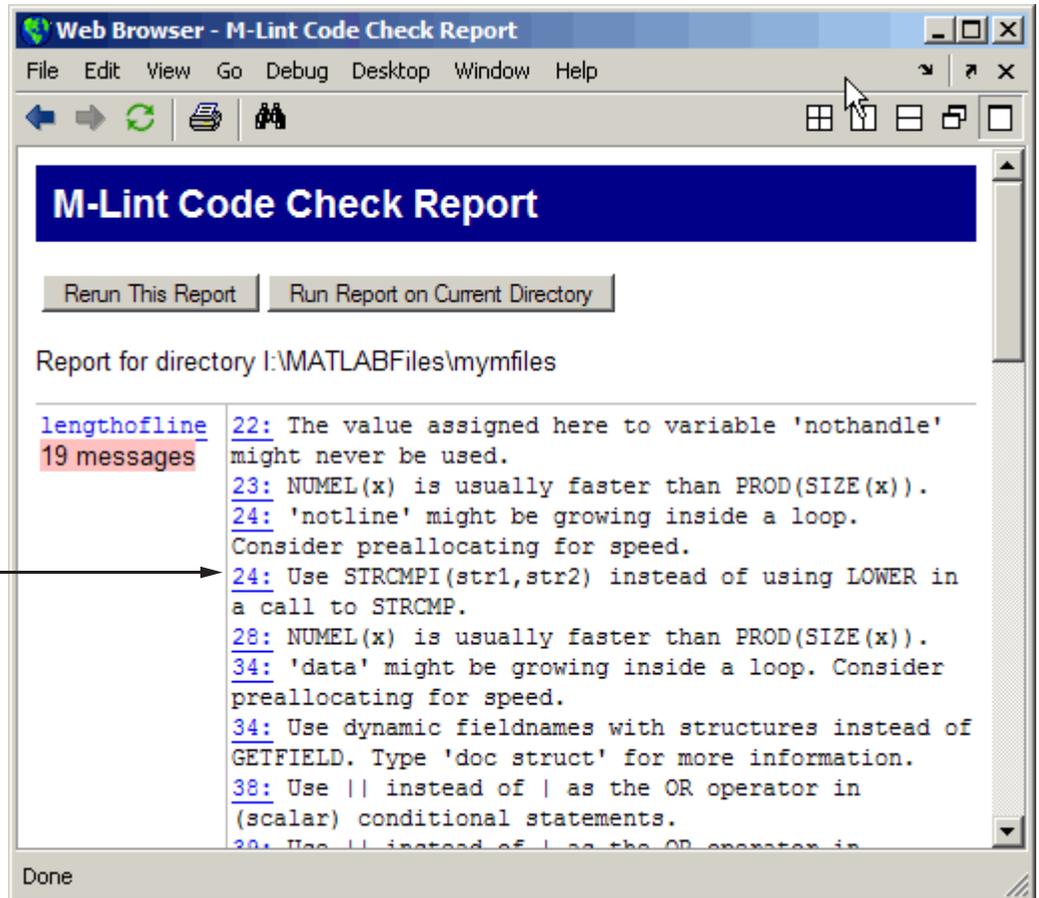
4 For each message, review the suggestion and your code, click the line number to open the M-file in the Editor/Debugger at that line, and make changes based on the message. Use the following general advice:

- If you are not sure what a message means or what to change in the code as a result, use the Help browser to look for related topics in the online documentation. For examples of messages and what to do about them, including specific changes to make for the example, lengthofline.m, see "Making Changes Based on M-Lint Messages" on page 7-18.

- M-Lint does not provide perfect information about every situation and in some cases, you might not want to make any changes based on the M-Lint message. In the event you do not want to change the code but you also do not want to see the M-Lint message for that line in the M-Lint Report, instruct M-Lint to ignore a line by adding %#ok to the end of the line in the M-file. (You can override the %#ok by running the mlint function with the '-notok' tag.)

- If there are certain messages or types of messages you do not want to see, you can set a preference so that M-Lint does not report them. Select **File > Preferences > M-Lint**. In **Select messages to enable**, clear the check box for messages you do not want to see. Review the settings for all messages to ensure you are seeing those pertinent to your file. Click **OK**. For more information, click the **Help** button in the **M-Lint Preferences** pane. The next time you run the report, the messages will not appear. You can use %#ok with a specific message ID so that only that type of message is suppressed—for more information, see the reference page for mlint.

5 After making changes, save the M-file. Consider saving the file to a different name if you made significant changes that might introduce errors. Then you can refer to the original file if needed to resolve problems with the updated file. Use **Tools > Compare Against** in the Editor/Debugger to help you identify the changes you made to the file. For more information, see "Comparing Files — File Comparison Tool" on page 6-54.

6 Run and debug the file(s) again to be sure you have not introduced any inadvertent errors.

7 If the M-Lint Code Check Report is already displayed, click **Rerun This Report** to update the report based on the changes you made to the file, or run the report from the Current Directory browser toolbar. Ensure the M-Lint messages are gone, based on the changes you made to the M-files.

Making Changes Based on M-Lint Messages

For information on how to correct the potential problems presented by M-Lint, use the following resources:

- Look for relevant topics in the MATLAB Programming and "Programming Tips" documentation.

- Use the Help browser **Search** and **Index** panes to find documentation about terms presented in the M-Lint messages.

Other techniques to help you identify problems in and improve your M-files are in these topics:

- "Syntax Highlighting" on page 6-28 in the Command Window and Editor/Debugger
- "Examining Errors" on page 3-9 generated when you run the M-file
- "Finding Errors, Debugging, and Correcting M-Files" on page 6-84, namely the Editor/Debugger and debugging functions
- "Profiling for Improving Performance" on page 7-27 for improving performance

Example Using M-Lint Messages to Improve Code

An example file, lengthofline.m, is included with MATLAB in *matlabroot*/matlab/help/techdoc/matlab_env/examples.

To run the M-Lint Code Check Report for lengthofline.m, change the current directory to its location by running

```
cd(fullfile(matlabroot,'help','techdoc','matlab_env','examples'))
```

In the Current Directory browser, select the **M-Lint Code Check Report** from the list of directory reports on the toolbar.

The M-Lint Code Check Report appears, with its list of messages suggesting improvements you can make to lengthofline.m and any other files in the directory.

M-Lint Code Check Report

File   Edit   View   Go   Debug   Desktop   Window   Help

M-Lint Code Check Report

[Rerun This Report]   [Run Report on Current Directory]

Report for directory I:\MATLABFiles\mymfiles

19 messages

22: The value assigned here to variable 'nothandle' might never be used.
23: NUMEL(x) is usually faster than PROD(SIZE(x)).
24: 'notline' might be growing inside a loop. Consider preallocating for speed.
24: Use STRCMPI(str1,str2) instead of using LOWER in a call to STRCMP.
28: NUMEL(x) is usually faster than PROD(SIZE(x)).
34: 'data' might be growing inside a loop. Consider preallocating for speed.
34: Use dynamic fieldnames with structures instead of GETFIELD. Type 'doc struct' for more information.
38: Use || instead of | as the OR operator in (scalar) conditional statements.
39: Use || instead of | as the OR operator in (scalar) conditional statements.
40: Use || instead of | as the OR operator in (scalar) conditional statements.
42: 'data' might be growing inside a loop. Consider preallocating for speed.
43: 'dim' might be growing inside a loop. Consider preallocating for speed.
45: 'dim' might be growing inside a loop. Consider preallocating for speed.
48: There may be a parenthesis imbalance around here.
48: There may be a parenthesis imbalance around here.
48: There may be a parenthesis imbalance around here.
48: There may be a parenthesis imbalance around here.
49: Terminate statement with semicolon to suppress output (in functions).
49: Use of brackets [] is unnecessary. Use parentheses to group, if needed.

Done

Messages and Resulting Changes for the lengthofline Example. The
following table describes each message and demonstrates a way to change the
file, based on the message.

Message — Code (Original Line Numbers)	Explanation and Updated Code (New Line Numbers)
22: The value assigned here to variable 'nothandle' might never be used. — — — — — — — — — — — — — — — — 22 nothandle = ~ishandle(hline); 23 for nh = 1:prod(size(hline)) 24 notline(nh) = ~ishandle(hline(nh)) ...	In line 22, nothandle is assigned a value, but nothandle is probably not used anywhere after that in the file. The line might be extraneous and you could delete it. But it might be that you actually intended to use the variable, which is the case for the lengthofline example. Update line 24 to use nothandle, which is faster than computing ~ishandle for each iteration of the loop, as shown here. 22 nothandle = ~ishandle(hline); 23 for nh = 1:numel(hline) 24 notline(nh) = nothandle(nh) ...
23: NUMEL(x) is usually faster than PROD(SIZE(x)). — — — — — — — — — — — — — — — — 23 for nh = 1:prod(size(hline))	While prod(size(x)) returns the number of elements in a matrix, the numel function was designed to do just that, and therefore is usually more efficient. Type doc numel to see the numel reference page. Change the line to 23 for nh = 1:numel(hline)

Message — Code (Original Line Numbers)	Explanation and Updated Code (New Line Numbers)
24: 'notline' might be growing inside a loop. Consider preallocating for speed. — — — — — — — — — — — — — — — — — 22 nothandle = ~ishandle(hline); 23 for nh = 1:numel(hline) 24 notline(nh) = ~ishandle(hline(nh)) ...	When you increase the size of an array within a loop, it is inefficient. Before the loop, preallocate the array to its maximum size to improve performance. For more information, see "Preallocating Memory" in the MATLAB Programming documentation. In the example, add a new line to preallocate notline before the loop. 23 notline = false(size(hline)); 24 for nh = 1:numel(hline) 25 notline(nh) = nothandle(nh) ...
24: Use STRCMPI(str1,str2) instead of using LOWER in a call to STRCMP. — — — — — — — — — — — — — — — — — 24 notline(nh)=~ishandle(hline(nh)) || ~strcmp('line',lower(get(hline(nh), 'type')));	While strcmp ('line',lower(get(hline(nh)'type')) converts the result of the get function to a lowercase string before doing the comparison, the strcmpi function ignores the case while performing the comparison, with advantages that include more efficiency. Change line 25 to notline(nh) = nothandle(nh) || ~strcmpi('line',get(hline(nh),'type'));
28: NUMEL(x) is usually faster than PROD(SIZE(x)). — — — — — — — — — — — — — — — — — 28 for nl = 1:prod(size(hline))	See the same message and explanation reported for line 23. Change the line 29 to for nl = 1:numel(hline)

Message — Code (Original Line Numbers)	Explanation and Updated Code (New Line Numbers)
34: 'data' might be growing inside a loop. Consider preallocating for speed. —————————————— 33 for nd = 1:length(fdata) 34 data{nd} = getfield(flds,fdata{nd});	See the same message and explanation reported for line 24. Add this line, 34, before the loop `data = cell(size(fdata));`
34: Use dynamic fieldnames with structures instead of GETFIELD. Type 'doc struct' for more information. —————————————— 34 data{nd} = getfield(flds,fdata{nd});	You can access a field in a structure as a variable expression that MATLAB evaluates at run-time. This is more efficient than using `getfield`. For more information, type `doc struct` to see the reference page for structures, or see "Using Dynamic Field Names" in the MATLAB Programming documentation. Change line 37 to `data{nd} = flds.(fdata{nd});`

38: Use || instead of | as the OR operator in (scalar) conditional statements.
39: Use || instead of | as the OR operator in (scalar) conditional statements.
40: Use || instead of | as the OR operator in (scalar) conditional statements.
 	
——————————————
 	
38 if isempty(data{3}) | ...
 	
39 (length(unique(data{1}(:)))==1 | ...
 	
40 length(unique(data{2}(:)))==1 | ...
 	
41 length(unique(data{3}(:)))==1)
	While | (the element-wise logical OR operator) performs the comparison correctly, use the || (short circuit OR operator) for efficiency. For details, see "Logical Operators" in the MATLAB Programming documentation. Change lines 40, 41, and 42 to
	
	`if isempty(data{3}) || ...`
	
	`(length(unique(data{1}(:)))==1 || ...`
	
	`length(unique(data{2}(:)))==1 || ...`

Message — Code (Original Line Numbers)	Explanation and Updated Code (New Line Numbers)

`42: 'data' might be growing inside a loop. Consider preallocating for speed.`
 	
— — — — — — — — — — — — — — — — — —
 	
`42 data{3} = zeros(size(data{1}));`
	This message no longer appears due to the change made to line 34 `data{nd} = getfield(flds,fdata{nd});`. Sometimes fixing code in one line automatically clears a message for another line. If the reason for a message or the action to take for a message is not obvious at first, it could be because another line is causing the message. Address the issues that are easy to fix first and rerun the report. Do not make any changes to line 44.

`43: 'dim' might be growing inside a loop. Consider preallocating for speed.`
 	
`43 dim(nl) = 2;`
	See the same message and explanation reported for line 24. Add this line before the first line of the loop
	
	`dim = len;`

`48: There may be a parenthesis imbalance around here.` `48: There may be a parenthesis imbalance around here.` `48: There may be a parenthesis imbalance around here.` `48: There may be a parenthesis imbalance around here.`	There is an error in this line, which you would see by running `lengthofline`. M-Lint suggests that it might be due to a parenthesis imbalance. You can check that by moving the arrow key over each of the delimiters, to see if MATLAB indicates a mismatch. This requires that **File > Preferences > Keyboard > Delimiter Matching** has the **Match on arrow key** option selected. There are no mismatched delimiters. The actual problem is the semicolon in parentheses, `data{3}(:)` is incorrect and should be a colon. In line 51, change `data{3}(;)` to `data{3}(:)`. That single change addressed the issues in all the messages for that line.

Message — Code (Original Line Numbers)	Explanation and Updated Code (New Line Numbers)
49: Terminate statement with semicolon to suppress output (in functions).	Adding a semicolon to the end of a statement suppresses output and is a common practice. M-Lint alerts you to lines that produce output but lack the terminating semicolon. If you want to view output from this line, do not add the semicolon. You can instruct M-Lint to ignore all messages on this line so that the messages on it will not appear by adding %#ok to the end of the line. However, because there is currently another message on the line, do not add %#ok until you have addressed the other message.
	Alternatively, you can add %#ok with the message ID for the specific message you want to suppress. To determine the message ID, run mlint('lengthofline.m', '-id'), which indicates the ID is NOPRT—for more information, see the mlint function reference page.
	For this example, assume you want to display the output and suppress the M-Lint message. To do so, add %#ok<NOPRT> to the end of the line.
	Note that there is a similar message for M-file scripts. This is so you can suppress the message for M-files that are cell-mode scripts, because they are often intended as demos and the display of output is intentional.
49: Use of brackets [] is unnecessary. Use parentheses to group, if needed. ——————————————————— 49 len(nl) = sum([sqrt(dot(temp',temp'))])	For more information about the use of brackets and parentheses, see the Special Characters reference page. In this example, remove the brackets because they are not needed. They add processing time because MATLAB concatenates unnecessarily. Change line 52 to `len(nl) = sum(sqrt(dot(temp',temp')))` %#ok

Updated M-Lint Code Check Report after making changes to the lenghtofline file based on M-Lint messages. Now, no messages are reported.
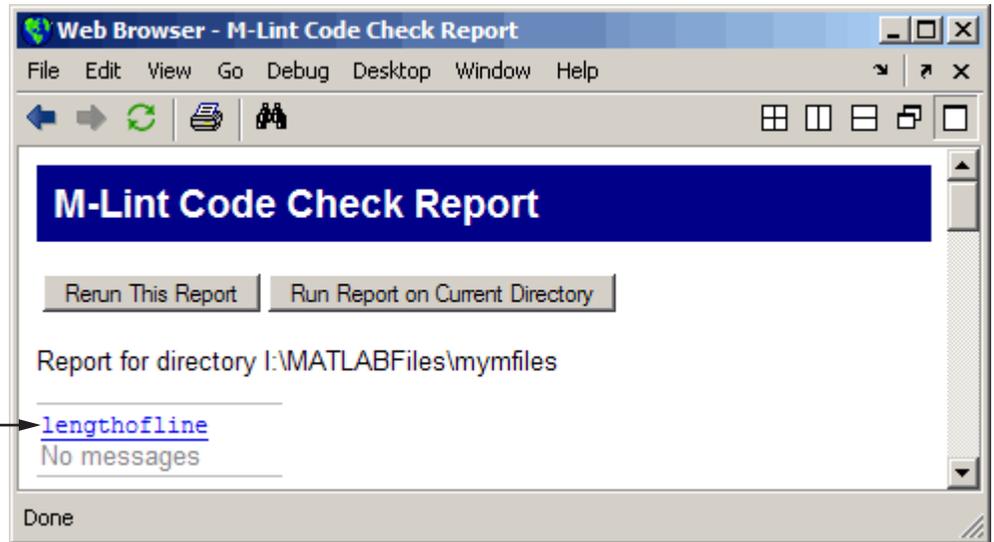
The M-file that includes all of the changes suggested by M-Lint is `lengthofline2.m`. To view it, run

```
edit(fullfile(matlabroot,'help','techdoc','matlab_env',...
'examples','lengthofline2.m')).
```

Other Ways to Access M-Lint

You can get M-Lint messages using any of the following methods. Each provides the same M-Lint messages, but in a different format:

- Access the M-Lint Code Check report for an M-file from the Editor/Debugger Tools menu or from the Profiler detail report.

- Run the `mlint` function, which analyzes the specified file and displays messages in the Command Window, or `mlintrpt`, which runs `mlint` and displays the messages in the Web Browser.

- Use automatic M-Lint analysis and code correction while you work on a file in the Editor/Debugger — see "M-Lint Code Analyzer" on page 6-87.

Profiling for Improving Performance

What Is Profiling?

Profiling is a way to measure where a program spends time. To assist you in profiling, MATLAB provides a graphical user interface, called the Profiler, which is based on the results returned by the profile function. Once you identify which functions are consuming the most time, you can determine why you are calling them and look for ways to minimize their use and thus improve performance. It is often helpful to decide whether the number of times a particular function is called is reasonable. Because programs often have several layers, your code may not explicitly call the most time-consuming functions. Rather, functions within your code might be calling other time-consuming functions that can be several layers down in the code. In this case it is important to determine which of your functions are responsible for such calls.

Profiling helps to uncover performance problems that you can solve by

- Avoiding unnecessary computation, which can arise from oversight

- Changing your algorithm to avoid costly functions

- Avoiding recomputation by storing results for future use

When you reach the point where most of the time is spent on calls to a small number of built-in functions, you have probably optimized the code as much as you can expect.

Profiling Process and Guidelines

Here is a general process you can follow to use the Profiler to improve performance in your M-files. This section also describes how you can use profiling as a debugging tool and as a way to understand complex M-files.

Note Premature optimization can increase code complexity unnecessarily without providing a real gain in performance. Your first implementation should be as simple as possible. Then, if speed is an issue, use profiling to identify bottlenecks.

---

1 In the summary report produced by the Profiler, look for functions that used a significant amount of time or were called most frequently. See "Profile Summary Report" on page 7-33 for more information.

2 View the detail report produced by the Profiler for those functions and look for the lines that use the most time or are called most often. See "Profile Detail Report" on page 7-35 for more information.

You might want to keep a copy of your first detail report to use as a reference to compare with after you make changes, and then profile again.

3 Determine whether there are changes you can make to the lines most called or the most time-consuming lines to improve performance.

For example, if you have a `load` statement within a loop, `load` is called every time the loop is called. You might be able to save time by moving the `load` statement so it is before the loop and therefore is only called once.

4 Click the links to the files and make the changes you identified for potential performance improvement. Save the files and run `clear all`. Run the Profiler again and compare the results to the original report. Note that there are inherent time fluctuations that are not dependent on your code. If you profile the exact same code twice, you can get slightly different results each time.

5 Repeat this process to continue improving the performance.

Using Profiling as a Debugging Tool

The Profiler is a useful tool for isolating problems in your M-files.

For example, if a particular section of the file did not run, you can look at the detail reports to see what lines did run, which might point you to the problem.

You can also view the lines that did not run to help you develop test cases that exercise that code.

If you get an error in the M-file when profiling, the Profiler provides partial results in the reports. You can see what ran and what did not to help you isolate the problem. Similarly, you can do this if you stop the execution using **Ctrl+C**, which might be useful when a file is taking much more time to run than expected.

Using Profiling for Understanding an M-File

For lengthy M-files that you did not create or that you have not used for awhile and are unfamiliar with, you can use the Profiler to see how the M-file actually worked. Use the Profiler detail reports to see the lines actually called.

If there is an existing GUI tool (or M-file) similar to one that you want to create, start profiling, use the tool, then stop profiling. Look through the Profiler detail reports to see what functions and lines ran. This helps you determine the lines of code in the file that are most like the code you want to create.

Using the Profiler

Use the Profiler to help you determine where you can modify your code to make performance improvements. The Profiler is a tool that shows you where an M-file is spending its time. This section covers

- "Opening the Profiler" on page 7-30

- "Running the Profiler" on page 7-30

- "Profiling a Graphical User Interface" on page 7-32

- "Profiling Statements from the Command Window" on page 7-33

- "Changing Fonts for the Profiler" on page 7-33

For information about the reports generated by the Profiler, see "Profile Summary Report" on page 7-33 and "Profile Detail Report" on page 7-35.

Opening the Profiler

You can use any of the following methods to open the Profiler:

- Select **Desktop > Profiler** from the MATLAB desktop.

- Click the Profiler button in the MATLAB desktop toolbar.

- With a file open in the MATLAB Editor/Debugger, select **Tools > Open Profiler**.

- Select one or more statements in the Command History window, right-click to view the context menu, and choose **Profile Code**.

- Enter the following function in the Command Window:

```
profile viewer
```

Running the Profiler

The following illustration summarizes the steps for profiling.

1 Type profile viewer to open the Profiler.

2 Type the statement to run.

3 Click **Start Profiling**.
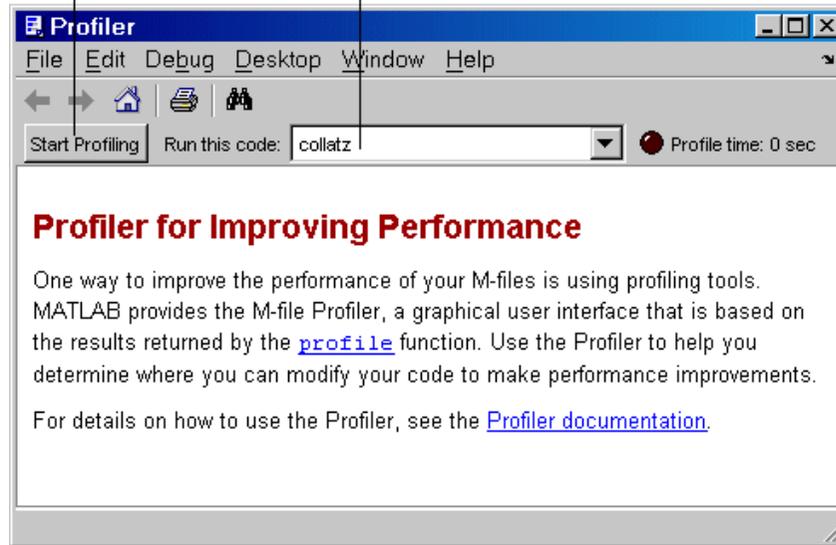
To profile an M-file or a line of code, follow these steps:

1 In the **Run this code** field in the Profiler, type the statement you want to run.

You can run this example

```
[t,y] = ode23('lotka',[0 2],[20;20])
```

as the code is provided with MATLAB demos. It runs the Lotka-Volterra predator-prey population model. For more information about this model, type lotkademo, which runs the demonstration.

To run a statement you previously profiled in the current MATLAB session, select the statement from the list box — MATLAB automatically starts profiling the code, so skip to step 3.

2 Click **Start Profiling** (or press **Enter** after typing the statement).

While the Profiler is running, the **Profile time** indicator (at the top right of the Profiler window) is green and the number of seconds it reports increases.

Profile time: 1 sec

When the profiling is finished, the **Profile time** indicator becomes black and shows the length of time the Profiler ran. The statements you profiled are shown as having been executed in the Command Window.

Profile time: 6 sec

This is not the actual time that your statements took to run; it is the wall clock (or `tic`/`toc`) time elapsed from when you clicked **Start Profiling** until profiling stops. If the time reported is much different from what you expected (for example, hundreds of seconds for a simple statement), you might have had profiling on longer than you realized. This also does not match the time reported in Profiler statistics, which is based on `cpu` time by default, not wall clock time.

3 When profiling is complete, the Profile Summary report appears in the Profiler window. For more information about this report, see "Profile Summary Report" on page 7-33.

Profiling a Graphical User Interface

You can run the Profiler for a graphical user interface, such as the Filter Design and Analysis tool included with Signal Processing Toolbox. You can also run the Profiler for an interface you created, such as one built using GUIDE.

To profile a graphical user interface, follow these steps:

1 In the Profiler, click **Start Profiling**. Make sure that no code appears in the **Run this code** field.

2 Start the graphical user interface. (If you do not want to include its startup process in the profile, do not click **Start Profiling**, step 1, until after you have started the graphical interface.)

3 Use the graphical interface. When you are finished, click **Stop Profiling** in the Profiler.

The Profile Summary report appears in the Profiler.

Profiling Statements from the Command Window

To profile more than one statement, follow these steps:

1 In the Profiler, clear the **Run this code** field and click **Start Profiling**.

2 In the Command Window, enter and run the statements you want to profile.

3 After running all the statements, click **Stop Profiling** in the Profiler.

The Profile Summary report appears in the Profiler.

Changing Fonts for the Profiler

To change the fonts used in the Profiler, follow these steps:

1 Select **File > Preferences > Fonts** to open the **Font** Preferences dialog box.

2 In the Font Preferences dialog box, select the code or text font that you want to use in the Profiler. The Profiler is an HTML Proportional Text tool. For more information, click the **Help** button in the dialog box.

3 Click **Apply** or **OK**. The Profiler font reflects the changes.

Profile Summary Report

The Profile Summary report presents statistics about the overall execution of the function and provides summary statistics for each function called. The report formats these values in four columns.

• **Function Name** — A list of all the functions and subfunctions called by the profiled function. When first displayed, the functions are listed in

order by the amount of time they took to process. To sort the functions alphabetically, click the **Function Name** link at the top of the column.

- **Calls** — The number of times the function was called while profiling was on. To sort the report by the number of times functions were called, click the **Calls** link at the top of the column.

- **Total Time** — The total time spent in a function, including all child functions called, in seconds. The time for a function includes time spent on child functions. To sort the functions by the amount of time they consumed, click the **Total Time** link at the top of the column. By default, the summary report displays profiling information sorted by **Total Time**. Note that the Profiler itself uses some time, which is included in the results. Also note that total time can be zero for files whose running time was inconsequential.

- **Self Time** — The total time spent in a function, *not* including time for any child functions called, in seconds. To sort the functions by this time value, click the **Self Time** link at the top of the column.

- **Total Time Plot** — Graphic display showing self time compared to total time.

Following is the summary report for the Lotka-Volterra model described in "Example: Using the profile Function" on page 7-43.

To print a summary report, click the Print button ⎙.

To get more detailed information about a particular function, click its name in the **Function Name** column. See "Profile Detail Report" on page 7-35 for more information.

Click any column label to sort by that column.

Click any function name to display the detailed report.

Profile Detail Report

The Profile Detail report shows profiling results for a selected function that was called during profiling. A Profile Detail report is made up of seven sections, summarized below. By default, the Profile Detail report includes all seven sections, although, depending on the function, not every section contains data. To return to the Profile Summary report from the Profile Detail report, click the Home button 🏠 in the toolbar. The following sections provide more detail:

Controlling the Contents of the Detail Report Display

You can determine which sections are included in the display by selecting
them and then clicking the **Refresh** button. The following sections provide
more detail about each section of this report.

Select
report
options to
display, and
then click
Refresh.
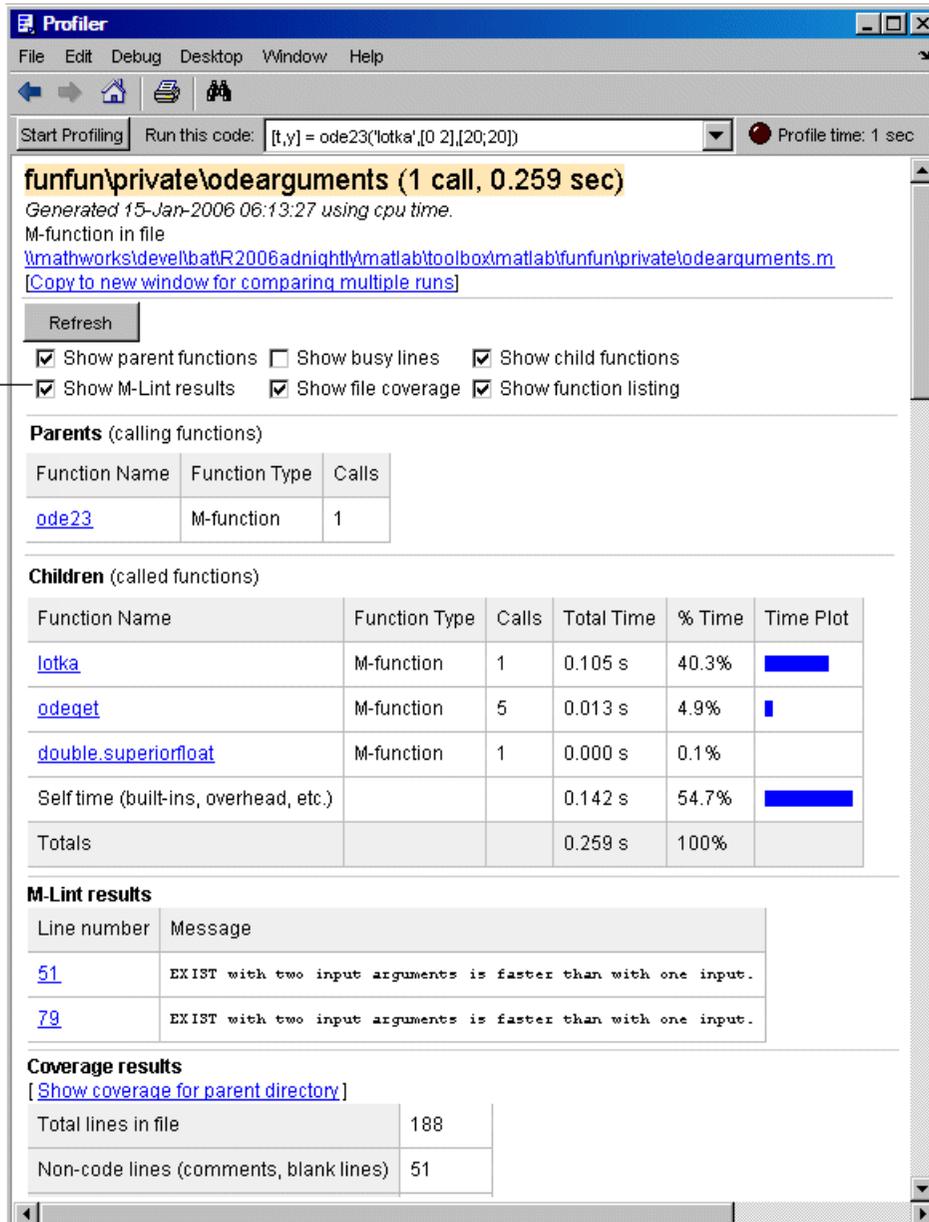
funfun\private\odearguments (1 call, 0.259 sec)

Generated 15-Jan-2006 06:13:27 using cpu time.
M-function in file
\\mathworks\devel\bat\R2006adnightly\matlab\toolbox\matlab\funfun\private\odearguments.m
[Copy to new window for comparing multiple runs]

Refresh

☑ Show parent functions ☐ Show busy lines ☑ Show child functions
☑ Show M-Lint results ☑ Show file coverage ☑ Show function listing

Parents (calling functions)

Function Name	Function Type	Calls
ode23	M-function	1

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
lotka	M-function	1	0.105 s	40.3%	■■■■
odeget	M-function	5	0.013 s	4.9%	■
double.superiorfloat	M-function	1	0.000 s	0.1%	
Self time (built-ins, overhead, etc.)			0.142 s	54.7%	■■■■
Totals			0.259 s	100%	

M-Lint results

Line number	Message
51	EXIST with two input arguments is faster than with one input.
79	EXIST with two input arguments is faster than with one input.

Coverage results
[Show coverage for parent directory]

Total lines in file	188
Non-code lines (comments, blank lines)	51

Profile Detail Report Header

The detail report header includes the name of the function that was profiled, the number of times it was called in the parent function, and the amount of time it used.

The header includes a link that opens the function in your default text editor.

The header also includes a link that copies the report to a separate window. Creating a copy of the report can be helpful when you make changes to the file, run the Profiler for the updated file, and compare the Profile Detail reports for the two runs. Do not make changes to M-files provided with MathWorks products, that is, files in *matlabroot*/toolbox directories.

Parent Functions

To include the **Parents** section in the detail report, select the **Show parent functions** check box. This section of the report provides information about the parent functions, with links to their detail reports.

Busy Lines

To include information about the lines of code that used the most amount of processing time in the detail report, select the **Show busy lines** check box.

Note that this was not selected in the example. Click a line number to view that line of code in the source listing.

Click a line number to go to that line in the file.

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
110	f0 = feval(ode,t0,y0,args{:}); ...	1	7.05 s	47.6%	▬▬▬
146	rtol = odeget(options,'RelTol'...	1	0.067 s	25.8%	▬▬
60	if (nargin(ode) == 2) ...	1	7.019 s	7.2%	▮
94	end	1	0.003 s	3.6%	▮
79	if (exist(ode)==2)	1	0.005 s	1.9%	▏
Other lines & overhead			0.054 s	23.9%	▬
Totals			0.253 s	100%	

Child Functions

To include the **Children** section of the detail report, select the **Show child functions** check box. This section of the report lists all the functions called by the profiled function. If the called function is an M-file, you can view the source code for the function by clicking its name.

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
Click to view detail report for functions.					
Children (called functions)					
lotka	M-function	1	0.105 s	40.3%	
odeget	M-function	5	0.013 s	4.9%	
double.superiorfloat	M-function	1	0.000 s	0.1%	
Self time (built-ins, overhead, etc.)			0.142 s	54.7%	
Totals			0.259 s	100%	

M-Lint Results

To include the **M-Lint results** section in the detail report display, select the **Show M-Lint results** check box. This section of the report provides information about problems and potential improvements, generated by M-Lint about the function. For more information about M-Lint, see "M-Lint Code Check Report" on page 7-16.

M-Lint results

Click a line number to go to line in code.

Line number	Message
51	EXIST with two input arguments is faster than with one input.
79	EXIST with two input arguments is faster than with one input.

File Coverage

To include the **Coverage results** section in the detail report display, select the **Show file coverage** check box. This section of the report provides statistical information about the number of lines in the code that executed during the profile run.

Function Listing

To include the **Function listing** section in the detail report display, select the **Show function listing** check box. If the file is an M-file, the Profile Detail report includes a column listing the execution time for each line, a column listing the number of times the line was called, and the source code for the function.

In the function listing, comment lines appear in green, lines of code that executed appear in black, and lines of code that did not execute appear in gray. If you click a function name in the listing, you can view its detail report.

By default, the Profile Detail report uses the color red to highlight the lines of code with the longest execution time. The darker the shade of red, the longer the line of code took to execute. Using the menu in this section of the detail report you can change this default and choose to highlight lines of code based on other criteria such as the lines called the most, lines called out by M-Lint, or lines of code that were (or were not) executed. Using this menu, you can also turn off highlighting completely.

Select the criteria for highlighting lines. To turn highlighting off, select none.

Function listing

Color highlight code according to | coverage | ▼ |

| time |
| numcalls |
| coverage |
| noncoverage |
| mlint |
| none |

```
  time   calls  line
                  1  function [...     n, next, t0, tfinal, tdir, y0, f0, args,
                  2                     d, rtol, normcontrol, normy, hmax, htry,
                  3                     .
                  4      odearg         Used, solver, ode, tspan, y0, options, ex
                  5  %ODEARGUME         tion that processes arguments for all ODE
                  6  %
                  7  %   See also ODE113, ODE15I, ODE15S, ODE23, ODE23S, ODE23T, ODE23TB,
                  8
                  9  %   Mike Karr, Jacek Kierzenka
                 10  %   Copyright 1984-2005 The MathWorks, Inc.
                 11  %   $Revision: 1.12.4.9 $  $Date: 2005/07/29 11:41:32 $
                 12
  < 0.01      1   13  if strcmp(solver,'ode15i')
                 14      FcnHandlesUsed = true;    % no MATLAB v. 5 legacy for ODE15I
                 15  end
                 16
  < 0.01      1   17  if FcnHandlesUsed  % function handles used
                 18      msg = ['When the first argument to ', solver,' is a function handl
                 19      if isempty(tspan) || isempty(y0)
                 20        error('MATLAB:odearguments:Tspan0rY0NotSupplied',...
                 21            [msg 'the tspan and y0 arguments must be supplied.']);
                 22      end
```

Highlighted lines

The profile Function

The Profiler is based on the results returned by the `profile` function. This section describes

- "profile Function Syntax Summary" on page 7-43
- "Example: Using the profile Function" on page 7-43
- "Accessing Profiler Results" on page 7-45
- "Saving Profile Reports" on page 7-47

profile Function Syntax Summary

Here is a summary of some of the main forms of profile. For details about these and other options, type doc profile. Some people use profile simply to see the child functions; see also depfun for that purpose.

Syntax	Description
profile on	Starts profile, clearing previously recorded statistics.
profile on -detail *level*	Specifies the level of function to be profiled, where *level* can be either:
	'mmex' — M-functions, M-subfunctions, and MEX-functions 'builtin' — M-functions, M-subfunctions, MEX-functions, and built-ins
profile on -history	Specifies that the exact sequence of function calls is to be recorded.
profile off	Suspends profile.
profile resume	Restarts profile without clearing previously recorded statistics.
profile clear	Clears the statistics recorded by profile.
profile viewer	Opens the Profiler, a graphical user interface and displays the information gathered as an HTML-formatted report.
	Note: If you run the obsoleted syntax profile report, the profile function calls this syntax.
s = profile('status')	Displays a structure containing the current profile status.
stats = profile('info')	Suspends profile and displays a structure containing profile results.

Example: Using the profile Function

This example demonstrates how to run profile:

1 To start `profile`, type in the Command Window

```
profile on
```

2 Execute an M-file. This example runs the Lotka-Volterra predator-prey population model. For more information about this model, type `lotkademo`, which runs a demonstration.

```
[t,y] = ode23('lotka',[0 2],[20;20]);
```

3 Generate the profile report and display it in the Profiler window. This suspends `profile`.

```
profile viewer
```

4 Restart `profile`, without clearing the existing statistics.

```
profile resume
```

The `profile` function is now ready to continue gathering statistics for any more M-files you run. It will add these new statistics to those generated in the previous steps.

5 Stop `profile` when you finish gathering statistics.

```
profile off
```

6 To view the profile data, call `profile` specifying the `'info'` argument. The profile function returns data in a structure.

```
p = profile('info')

p =
        FunctionTable: [10x1 struct]
      FunctionHistory: [2x0 double]
       ClockPrecision: 3.3333e-010
           ClockSpeed: 3.0000e+009
                 Name: 'MATLAB'
```

The `FunctionTable` indicates that statistics were gathered for 10 functions.

7 To save the profile report, use the `profsave` function. This function stores the profile information in separate HTML files, for each function listed in `FunctionTable` of p.

```
profsave(p)
```

By default, `profsave` puts these HTML files in a subdirectory of the current directory named `profile_results`, and displays the summary report in your system browser. You can specify another directory name as an optional second argument to `profsave`.

Accessing Profiler Results

The `profile` function returns results in a structure. This example illustrates how you can access these results:

1 To start `profile`, specifying the `detail` and `history` options, type in the Command Window.

```
profile on -detail builtin -history
```

The detail option specifies that built-ins should be included in the profile data. The history option specifies that the report include information about the sequence of functions as they are entered and exited during profiling.

2 Execute an M-file. This example runs the Lotka-Volterra predator-prey population model. For more information about this model, type `lotkademo`, which runs a demonstration.

```
[t,y] = ode23('lotka',[0 2],[20;20]);
```

3 Stop profiling.

```
profile off
```

4 Get the structure containing profile results.

```
stats = profile('info')
stats =
        FunctionTable: [43x1 struct]
      FunctionHistory: [2x754 double]
       ClockPrecision: 3.3333e-010
```

```
                    ClockSpeed: 3.0000e+009
                          Name: 'MATLAB'
```

5 The FunctionTable field is an array of structures, where each structure
represents an M-function, M-subfunction, MEX-function, or, because the
builtin option is specified, a MATLAB built-in function.

```
stats.FunctionTable

ans =

41x1 struct array with fields:
    CompleteName
    FunctionName
    FileName
    Type
    NumCalls
    TotalTime
    TotalRecursiveTime
    Children
    Parents
    ExecutedLines
    IsRecursive
    PartialData
```

6 View the second structure in FunctionTable.

```
stats.FunctionTable(2)

ans =
              CompleteName: [1x79 char]
              FunctionName: 'ode23'
                  FileName: [1x73 char]
                      Type: 'M-function'
                  NumCalls: 1
                 TotalTime: 0.3902
        TotalRecursiveTime: 0
                  Children: [20x1 struct]
                   Parents: [0x1 struct]
             ExecutedLines: [139x3 double]
```

```
          IsRecursive: 0
          PartialData: 0
```

7 To view the history data generated by `profile`, view the `FunctionHistory`, for example, `stats.FunctionHistory`. The history data is a 2-by-n array. The first row contains Boolean values, where `0` (zero) means entrance into a function and `1` means exit from a function. The second row identifies the function being entered or exited by its index in the `FunctionTable` field. To see how to create a formatted display of history data, see the example on the `profile` reference page.

Saving Profile Reports

To save the profile report, use the `profsave` function.

This function stores the profile information in separate HTML files, for each function listed in the `FunctionTable` field of the structure, `stats`.

```
profsave(stats)
```

By default, `profsave` puts these HTML files in a subdirectory of the current directory named `profile_results`. You can specify another directory name as an optional second argument to `profsave`.

```
profsave(stats,'mydir')
```

8

Publishing Results

MATLAB provides two different approaches for publishing: using cells and with the Notebook features for Microsoft Word.

Publishing to HTML, XML, LaTeX, Word, and PowerPoint Using Cells

In this section...
"About Publishing M-Files" on page 8-2
"Publishing Scripts and Functions—Differences" on page 8-3
"Example of Publishing Without Text Markup" on page 8-4
"Example of Publishing with Text Markup" on page 8-6

About Publishing M-Files

When you have completed writing and debugging an M-file, use the M-file cell features in the Editor/Debugger to quickly publish the M-file and its results in any of several presentation formats: HTML, XML, LaTeX, or, when the applications are installed, Microsoft Word or PowerPoint. This allows you to share your work with others, presenting not only the code, but also commentary on the code and results from running the file.

Publishing features evaluate M-files cell-by-cell and display the contents of a cell in a presentation quality document. For M-file scripts, publishing also displays the results. For example, published documents include bold headings for each section of the file. For M-file scripts, published documents include output to the Command Window and figures. The cells in the Editor/Debugger that you use for publishing are the same ones you might already have used for rapid code iteration—see "Using Cells for Rapid Code Iteration and Publishing Results" on page 6-133.

If you have an active Internet connection, you can watch the Publishing M Code from the Editor/Debugger video demo for an overview of the major functionality.

This is the overall process to publish an M-file using cell features in the Editor/Debugger:

1 Enable cell mode and define cells as described in steps 1 through 3 in "Using Cells for Rapid Code Iteration and Publishing Results" on page 6-133. When you publish the file without adding any text markup, comments at

the start of a cell appear as plain text. Comments appearing after code in a cell appear as unformatted M-file comments in the published document.

2 Use **Cell > Insert Text Markup** to insert markup symbols in the M-file comments to stylize the text for the output, for example, to display specified text as bold or monospaced. For details, see "Marking Up Text in Cells for Publishing" on page 8-11.

3 When publishing an M-file function, select **File > Preferences > Editor/Debugger > Publishing**, and clear the **Evaluate code** option.

4 Select **File > Publish To**, and select the format in which you want to publish the M-file: HTML, XML, LaTeX, Word, or PowerPoint. For details, see "Publishing M-Files Using Cells" on page 8-24.

5 Change the **Editor/Debugger Publishing** and **Publishing Images** preferences to adjust the output. For example, you can choose to include or exclude the code from the output. For details, see "Modifying Published Output Using Preferences" on page 8-26.

MATLAB publishes the M-file by writing the cell titles, comment text, and code to a file using the specified format. For M-file scripts, MATLAB also evaluates the cells and writes the results of the evaluation to the output file. Any figures created or modified during the evaluation are saved as graphics files, and are shown with the results.

Publishing Scripts and Functions—Differences

When you publish M-file scripts, MATLAB runs the code and the published output includes the results.

When you publish M-file functions, MATLAB does not run the code, so the published output does not include results. Publishing an M-file function allows you to effectively save the M-file code along with formatted M-file comments. The instructions for publishing M-files apply to M-file functions, with the exception of steps that involve evaluating code and displaying results from running the file.

If you want to publish an M-file function that includes results, change the M-file function to a script. Remove the function declaration statement and

within the M-file script, supply any input values that you had passed when you ran the function.

Example of Publishing Without Text Markup

This is based on the M-file script used in "Example — Evaluate Cells" on page 6-141, as shown here. Instructions for preparing and publishing the file follow.

Select **File > Publish to HTML** to produce the following result.

M-file published to HTML format.

Publishing automatically generates a cell contents listing with links to all cell (section) titles.

Automatic formatting makes cell titles bold and shows code in a monospace font. Executable code appears with a gray background.

Results from running the M-file are included in the published document.

Contents

- Calculate and Plot Sine Wave
- Modify Plot Properties

Calculate and Plot Sine Wave

Define the range for x. Calculate and plot y = sin(x).

```
x = 0:0.125:6*pi;
y = sin(x);
plot(x,y)
```
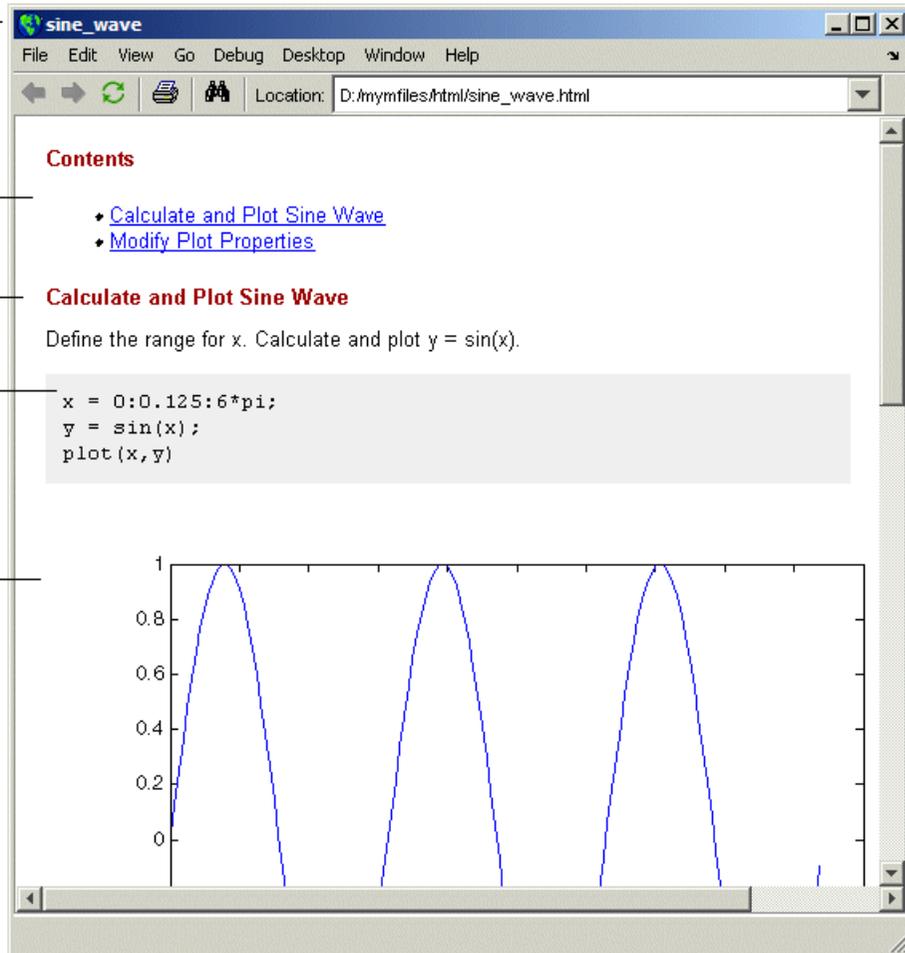
Example of Publishing with Text Markup

This simple example adds text markup to the sine_wave.m file used in
"Example of Publishing Without Text Markup" on page 8-4 to produce the
following published HTML document.

Published file after applying text markup.

Add a title for the document.

Display comment lines using TeX format.

Make selected comment text appear in monospace.

Reduce the size of the figure output in the document using **Preferences for Publishing Images**.

MATLAB code appears with gray background to distinguish it from results.

Plot Sine Wave

File   Edit   View   Go   Debug   Desktop   Window   Help

Location: D:/mymfiles/html/sine_wave

Plot Sine Wave

Calculate and plot a sine wave.

Contents

- Calculate and Plot Sine Wave
- Modify Plot Properties

Calculate and Plot Sine Wave

Define the range for x.

$$0 \le x \le 16\pi$$

Calculate and plot $y = sin(x)$.

```
x = 0:0.125:6*pi;
y = sin(x);
plot(x,y)
```

Modify Plot Properties

```
title('Sine Wave','FontWeight','bold')
xlabel('x')
```

1 Add an overall title and introduction for the published document

   a Select **Cell > Insert Text Markup > > Document Title and Introduction**. MATLAB adds the following at the top of the file.

```
%% DOCUMENT TITLE
% INTRODUCTORY TEXT
```

     The two percent signs (`%%`) indicate the start of a new cell, where a cell is a section of an M-file. A single percent sign indicates a comment line.

   b Replace `DOCUMENT TITLE` with `Plot Sine Wave`.

   c On line 2, after the single percent sign, replace `INTRODUCTORY TEXT` with a comment about the overall file. For example, `Calculate and plot a sine wave`.

   d On line 3, insert a blank line for better readability.

   You can add any overall comments about the file in the lines following the title. You cannot add executable code between document title and before the next cell (a line starting with `%%`) if you want the document title to appear as the overall document title.

2 Display equations in comments with symbols and Greek characters using the TeX format. For a list of symbols you can display and the character sequence to create them, see the `String` property on the MATLAB reference page for graphics `Text Properties`. In this example, to create a comment containing the following equation in the published document: $0 \leq x \leq 6\pi$, use text markup as follows:

   a Position the cursor at the end of line 5, where the text`Define the range for x` appears.

   b Select **Cell > Insert Text Markup > TeX Equation**.

     MATLAB inserts the following lines:

```
%
% $$e^{\pi i} + 1 = 0$$
%
```

     The sample equation, which is the text between the set of two dollar signs (`$$`), is highlighted.

c Replace the sample equation with the following TeX equation:

```
O \leq x \leq 6\pi
```

The three lines that display the TeX equation in the published document now appear as follows in the M-file.

```
%
% $$O \leq x \leq 6\pi$$
%
```

3 Display a selected comment text in a monospace font, as follows:

a Position the cursor in the following comment, which appears in line 9.

```
% Calculate and plot y = sin(x).
```

b To make the equation y = sin(x) appear in monospace font in the published document, select the equation and then select **Cell> Insert Text Markup> Monospaced Text**. Line 9 should appear as follows:

```
% Calculate and plot |y = sin(x)|.
```

4 To reduce the size of the published figure, select **File > Preferences > Editor/Debugger > Publishing Images**. In the **Preferences** dialog box, for **Resize image**, select **Restrict height to** and enter 200. Click **OK** to close the dialog box.

5 Select **File > Save and Publish to HTML**.

The HTML file displays in the MATLAB Web Browser, as shown at the start of this example, "Example of Publishing with Text Markup" on page 8-6.

6 By default, MATLAB stores the HTML document, sine_wave.html, and the associated image files in d:/mymfiles/html for this example.

The file sine_wave.m now appears as shown in the following illustration.

```
Editor - D:\mymfiles\sine_wave.m
File  Edit  Text  Go  Cell  Tools  Debug  Desktop  Window  Help

 1      %% Plot Sine Wave
 2      % Calculate and plot a sine wave.
 3
 4      %% Calculate and Plot Sine Wave
 5      % Define the range for x.
 6      %
 7      % $$0 \leq x \le1 6\pi$$
 8      %
 9      % Calculate and plot |y = sin(x)|.
10 -    x = 0:0.125:6*pi;
11 -    y = sin(x);
12 -    plot(x,y)
13      %% Modify Plot Properties
14 -    title('Sine Wave','FontWeight','bold')
15 -    xlabel('x')
16 -    ylabel('sin(x)')
17 -    set(gca,'Color','w')
18 -    set(gcf, 'MenuBar', 'none')

                                script        Ln  9      Col  34    OVR
```

Marking Up Text in Cells for Publishing

In this section...
"Overview of Text Markup" on page 8-11
"Text Markup for Cell Breaks, Headings, and Formatted Comments" on page 8-12
"Text Markup for Indented Text, Lists, and Graphics" on page 8-14
"Text Markup for HTML, LaTeX, and TeX Equation Output Types" on page 8-17
"Text Markup for Bold, Italic, and Monospaced Text Formats" on page 8-20
"Text Markup for Inline Links" on page 8-22

Overview of Text Markup

When you publish an M-file and results, you can mark up the file using cell features—this adds and formats comments in the published file. You can include the markup as you write the basic code, mark up the file after you've written the code, or do both. The markup applies to any of the available publishing options: HTML, XML, LaTeX, Word, and PowerPoint.

Any cell features you use for evaluating and improving your code are used for publishing purposes as well. The "Example of Publishing Without Text Markup" on page 8-4 shows how the cells used for improving an M-file appear when the M-file is published. You might want to change the existing cells for publishing purposes, but note that this changes the cells for evaluation purposes as well. For example, to have text markup and formatted comments in the output document, the comments must appear at the start of a cell, before any code.

Mark up comment text in one of two ways:

• Use **Cell > Insert Text Markup** menu items to format the code, which automatically inserts the markup symbols for you. This is not available for all markup options.

- Type the markup symbols directly in the code. Note that what you type is the same as the code that results if you instead use the equivalent menu item.

The following tables describe each markup option and how to use it. The tables refer to "Example of Publishing with Text Markup" on page 8-6:

Text Markup for Cell Breaks, Headings, and Formatted Comments

Cell breaks, headings, and formatted comments are the structural elements that control the overall output format of the published document.

Format	How to Produce Format	Resulting Code	Published Results

Overall document heading and introductory text	1 Position the cursor anywhere in the editor.
	
	2 Select **Cell > Insert Text Markup > Document Title and Introduction.**
	
	3 Replace **DOCUMENT TITLE** in the resulting code with your desired cell heading.
	
	4 Replace INTRODUCTORY TEXT in the resulting code with text that introduces the M-file.
	
	See step in the example.
		`%% DOCUMENT TITLE`
		`% INTRODUCTORY TEXT`
		In the example, the overall heading is
		`%% Plot Sine Wave`
			Document title is formatted as a top-level heading (h1 in HTML), using a large size, bold font.
			
			Introductory text appears as formatted text.

Section title, formatted comments, and a cell break	1 Position the cursor where you want to insert a new cell.
	
	2 Select **Cell > Insert Text Markup > Section Title with Cell Break.**
	
	3 Replace **SECTION TITLE** with your desired title.
	
	4 Replace DESCRIPTIVE TEXT with text that describes the cell.
		`%% SECTION TITLE`
		`% DESCRIPTIVE TEXT`
		In the example, the section titles are
		`%% Calculate and Plot Sine Wave`
		`%% Modify Plot Properties`
		Note that descriptive text must appear before the first line of executable code in a cell.
			Formatted as a heading (h2 in HTML), using a medium size, bold font.
			
			Descriptive text appears as formatted text in the published output.

Text Markup for Indented Text, Lists, and Graphics

Indented text, lists, and graphics are types of block styles. Block styles control the appearance of large sections of text within the final published document. A block is a series of comment lines within a cell's descriptive text that starts and ends with a blank comment line, or the beginning or end of the descriptive text.

Format	How to Produce Format	Resulting Code and Explanation	Published Results
Indented text	**1** Position the cursor before the line where you want to add indented text. **2** Select **Cell > Insert Text Markup > Preformatted Text**. **3** Replace the resulting code with your desired text, including tabs and spaces. Be careful, however, to keep the percent signs (%) at the beginning of each line.	``` % PREFORMATTED % TEXT ``` More than one space at the start of a block distinguishes it as preformatted.	The indents, spacing, and line breaks in the M-file are preserved in the output: ``` PREFORMATTED TEXT ```

Format	How to Produce Format	Resulting Code and Explanation	Published Results
Image	**1** Position the cursor before the line where you want to add a graphic. **2** Select **Cell > Insert Text Markup > Image**. **3** Replace the sample text inserted, FILENAME.PNG, with filename of the graphic you want to insert. Keep the percent sign (%) and angle brackets (<< >>) . The image file must be in the same directory as the output file, or you must specify a relative path to the image file, from the output file.	`% <<FILENAME.PNG>>`	If you replace FILENAME.PNG with surfpeaks.jpg, the published results appear as follows:

Format	How to Produce Format	Resulting Code and Explanation	Published Results

Bulleted list	1 Position the cursor before the line where you want to add a bulleted list.
	
	2 Select **Cell > Insert Text Markup > Bulleted List**.
	
	3 Replace the sample text inserted, ITEM 1 and ITEM 2, with your desired text. Be careful, however, to keep the percent signs (%) and asterisk (*) at the beginning of each line.
		```% * ITEM1```
		```% * ITEM2```
		
		The asterisk (*) at the start of a line indicates it is a bulleted list item.
			• ITEM1
			
			• ITEM2

Numbered list	1 Position the cursor before the line where you want to add a numbered list.
	
	2 Select **Cell > Insert Text Markup > Numbered List**.
	
	3 Replace the inserted text , ITEM 1 and ITEM 2, with your desired text. Do not, however, replace the percent sign (%) and number sign (#) that appear at the beginning of each line.
		```% # ITEM1```
		```% # ITEM2```
		
		The number sign (#) at the start of a line indicates it is a numbered list item.
			1 ITEM1
			
			2 ITEM2

Text Markup for HTML, LaTeX, and TeX Equation Output Types

You can specify the output type for a published M-file as HTML, LaTeX or TeX equation. When you publish the M-file, use the **File > Publish to** menu options accordingly. For details, see "Publishing M-Files Using Cells" on page 8-24.

Note When you markup text for the HTML or LaTex output type, that text is published only when the specified output type matches the markup type. For example, if you add HTML markup, but then select **File > Publish To > LaTeX**, the text enclosed within the HTML markup is not published. Similarly, if you add LaTeX markup, but then select **File > Publish To >**

HTML or click the Publish to HTML button ( ), the text enclosed within the LaTeX markup is not published.

Format	How to Produce Format	Resulting Code	Published Results
HTML markup	**1** Position the cursor before the line where you want to add HTML markup. **2** Select **Cell > Insert Text Markup > HTML Markup**. **3** Replace the inserted HTML markup (as shown in the next column) with your desired text. Be careful, however, to keep the percent signs (%) at the beginning of each line.	```% <table border=1><tr> <td>one</td><td>two </td></tr></table>% </html>```	

Format	How to Produce Format	Resulting Code	Published Results
LaTeX markup	**1** Position the cursor before the line where you want to add LaTeX markup. **2** Select **Cell > Insert Text Markup > LaTeX Markup**. **3** Replace the inserted LaTeX markup (as shown in the next column) with your desired text. Be careful, however, to keep the percent signs (%) at the beginning of each line.	```% <latex>% \begin{tabular}{|r|r}% \hline n&&$n!$\\ \\hline 1&1\\ 2&2\\ 3&6\\ \\hline% \end{tabular}% </latex>```	

TeX Equations and symbols	**1** Position the cursor before the line where you want to add an equation or symbols.
	
	2 Select **Cell > Insert Text Markup > TeX Equation**.
	
	3 Replace the inserted text inserted, `e^{\pi i} + 1 = 0`, with your desired TeX equation. See step in the example.
		```% $$e^{\pi i} + 1 = 0$$%``` The $$ at the start and end of a block description distinguish the TeX equation.
		
		For a list of symbols you can display and the character sequence to create them, see the `String` property on MATLAB graphics reference page for `Text Properties`.
		
		In the example, line 7 includes a TeX equation:
		
		`% $$0\leq 6\pi$$`
			$e^{\pi i} + 1 = 0$

Text Markup for Bold, Italic, and Monospaced Text Formats

You can mark up selected strings in the M-file comments so that they appear in bold, italic or monospaced text formats when you publish the M-file.

Format	How to Produce Format	Resulting Code	Published Results
Bold text	Follow these steps to bold existing text: **1** Within a comment, select text that you want to be bold. **2** Select **Cell > Insert Text Markup > Bold Text**. To insert sample text, that you will replace with your desired text, follow these steps: **1** Select **Cell > Insert Text Markup > Bold Text**. **2** Replace the inserted text with the text that you want to be bold.	`% *BOLD TEXT*`	**BOLD TEXT**

Format	How to Produce Format	Resulting Code	Published Results
Italic text	Follow these steps to bold existing text: **1** Within a comment, select text that you want to be italic. **2** Select **Cell > Insert Text Markup > Italic Text**. To insert sample text, that you will replace with your desired text, follow these steps: **1** Select **Cell > Insert Text Markup > Italic Text**. **2** Replace the inserted text with the text you want to be italic.	`% _ITALIC TEXT_`	*ITALIC TEXT*

Monospaced text	Follow these steps to bold existing text:
	
	1 Within a comment, select text that you want to be monospaced.
	
	2 Select **Cell > Insert Text Markup > Monospaced Text**.
	
	To insert sample text, that you will replace with your desired text, follow these steps:
	
	1 Select **Cell > Insert Text Markup > Monospaced Text**.
	
	2 Replace the inserted text with the text you want to be monospaced.
		`% |MONOSPACED TEXT|`
		
		In the example, monospaced text is added in line 9:
		
		`% Calculate and plot |y=sin(x)|.`
			`MONOSPACED TEXT`

Text Markup for Inline Links

When you specify hypertext links within an M-file, when you publish the document the hypertext links become active links to a URL on the web.

Format	How to Produce Format	Resulting Code	Published Results
URL as hyperlinked text	**1** Within a comment, position the cursor where you want to insert the hypertext link. **2** Select **Cell > Insert Text Markup > Hyperlinked Text**. The Editor/Debugger inserts the following code: `<http://www.mathworks.com The MathWorks>` **3** Replace www.mathworks.com with your desired URL. **4** Delete the string, The MathWorks.	`% <http://www.mathworks.com>>`	http://www.mathworks.com
Hyperlinked text without a printed URL	**1** Within a comment, position the cursor where you want to insert the hypertext link. **2** Select **Cell > Insert Text Markup > Hyperlinked Text**. The Editor/Debugger inserts the following code: `<http://www.mathworks.com The MathWorks>` **3** Replace www.mathworks.com with your desired URL. **4** Replace The MathWorks with the text that you want to appear as the hyperlinked text.	`% <http://www.mathworks.com The MathWorks>>`	The MathWorks

8-23

Publishing M-Files Using Cells

In this section...
"How to Publish an M-File" on page 8-24
"About Published M-Files" on page 8-25
"Modifying Published Output Using Preferences" on page 8-26

How to Publish an M-File

When you publish an M-file that contains cells and text markup, MATLAB produces an output document consisting of the M-file code, comments, and results.

1 When you publish an M-file function, MATLAB does not run the code or include results from running the code in the output document—for M-file functions, clear the **Evaluate code** preference prior to publishing.

2 After adding cells and text markup to an M-file, select **File > Publish To** and select an output format from those listed in the menu: **HTML**, **XML**, **LaTeX**, **Word**, or **PowerPoint**. If the M-file contains unsaved changes, the menu item becomes **Save and Publish To**.

 You can also publish to the default output format specified in **Preferences** using the Publish button in the Editor/Debugger toolbar.

MATLAB displays the published document in the appropriate tool for the selected output format:

- HTML displays in the MATLAB Web Browser.

- XML displays in the MATLAB Editor/Debugger.

- LaTeX displays in the MATLAB Editor/Debugger.

- Word displays in Microsoft Word.

- PowerPoint displays in Microsoft PowerPoint.

> **Note** Publishing to Microsoft Word and to PowerPoint features are available only on Windows systems that have the applications installed. Supported Word and PowerPoint versions are 2000, 2002, 2003, and 2007.

The published file contains the formatted comments, code with syntax highlighting and a gray background to distinguish it from results, and results for each cell. It also contains a **Contents** heading at the top of the file with a bulleted list of links to the named cells in the rest of the document.

When code in a cell creates or modifies a figure, the published file includes an image of the figure. If the code in a given cell modifies a figure more than once, the published file includes only one image of the figure, that being the last version of the figure.

When publishing to HTML, the M-file code is included at the end of published HTML file as comments. Use the `grabcode` function to extract the code from the HTML file.

Function Alternative

From the Command Window, run the `publish` function to run the M-file and publish the results. See the `publish` function reference page for options you can set.

About Published M-Files

Published Filenames and Locations

MATLAB names the published file the same as the M-file that produced it, adding the relevant extension for the selected output format: `.html`, `.xml`, `.tex`, `.doc`, or `.ppt`. MATLAB stores this output file, along with supporting files such as images of figure windows, in the `html` subdirectory under the directory containing the M-file you published.

For example, when you publish `d:/mymfiles/sine_wave.m` to HTML, MATLAB creates a directory `d:/mymfiles/html` that includes the published document `sine_wave.html`. Any figure windows produced by running the M-file appear as image files in the directory, for example, `sine_wave_img.png`.

TeX equations are image files as well; in the example, the equation file is `sine_wave_eq_eq####.png`. MATLAB creates a thumbnail file for the document, `sine_wave_img_thumbnail.png` in the example, if that preference is selected — see in the online documentation.

Publishing Code that Displays Hyperlinks in Command Window

If the M-file you publish contains statements that display hyperlinks in the MATLAB Command Window, the published document shows the code rather than the hyperlinks.

For example

```
disp('<a href="http://www.mathworks.com">Link to MathWorks</a>')
```

displays

Link to MathWorks

in the Command Window. You can click the link to go to the MathWorks Web site. When that `disp` statement is in an M-file you publish, the hyperlink tag and the text between it, that is,

```
<a href="http://www.mathworks.com">Link to MathWorks</a>
```

rather than the link, appears in the published document.

Similar results occur if you include

```
help matlab_functioname
```

in an M-file.

Modifying Published Output Using Preferences

Use preferences to control execution, output, and options related to images created during publishing. For details about these preferences, click the **Help** button in the **Preferences** dialog box for those panes.

Notebook for Publishing to Word

Note Notebook is available only on Windows systems that have Microsoft Word installed. For supported versions of Word, see "Configuring Notebook" on page 8-53.

Using Notebook to Create an M-book

Notebook enables you to access the numeric computation and visualization software of MATLAB from within the word processing environment, Microsoft Word. Using Notebook, you can create a document, called an *M-book*, that contains text, MATLAB commands, and the output from MATLAB commands.

You can think of an M-book as a record of an interactive MATLAB session annotated with text, or as a document embedded with live MATLAB commands and output. Notebook is useful for creating electronic or printed records of MATLAB sessions, class notes, textbooks or technical reports.

See Also Publishing Using Cells

As an alternative to Notebook, consider publishing using cells. For more information, see "Publishing to HTML, XML, LaTeX, Word, and PowerPoint Using Cells" on page 8-2.

Creating or Opening an M-Book

Creating an M-Book from MATLAB

To create a new M-book from within MATLAB, type

```
notebook
```

in the Command Window. If you are running Notebook for the first time, you might need to configure it. See "Configuring Notebook" on page 8-53 for more information.

Notebook starts Microsoft Word on your system and creates a new M-book, called `Document1`.

When Word is opening, if a dialog box appears asking you to enable or disable macros, choose to enable macros. Notebook defines Microsoft Word macros that enable MATLAB to interpret the different types of cells that hold MATLAB commands and their output. For more information on macro security, see "Configuring Notebook" on page 8-53.

Depending on the version of Word you are using, one of the following occurs:

- In Word 2000, 2002, and 2003, Notebook adds the **Notebook** menu to the Word menu bar, as shown in the following illustration. Use this menu to access Notebook features.
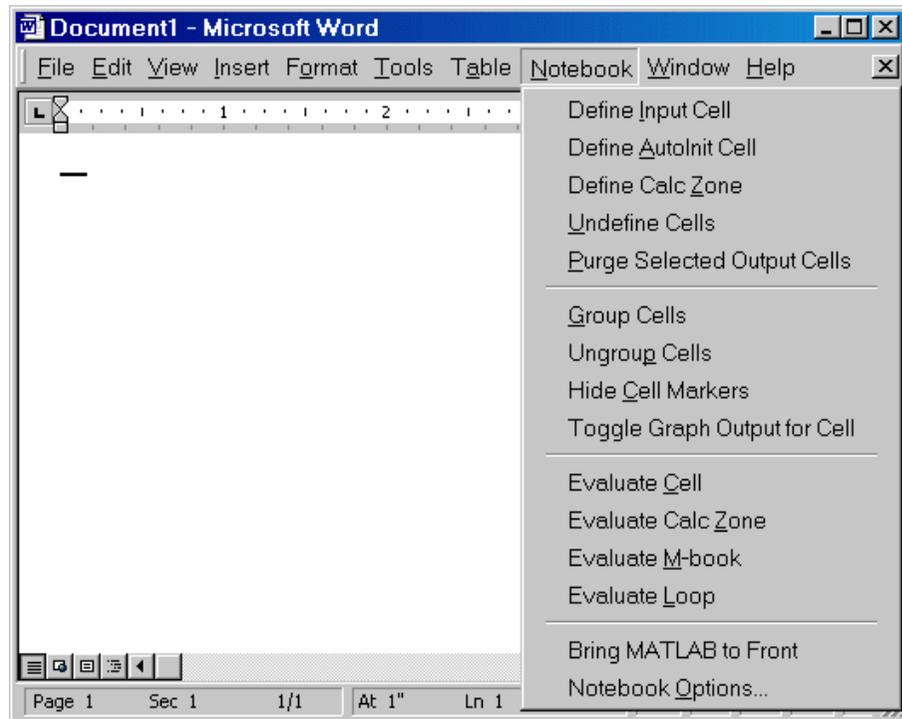
- In Word 2007, Notebook adds the **Notebook** menu to the Word **Add-Ins** tab, as shown in the following illustration. Use this menu to access Notebook features.
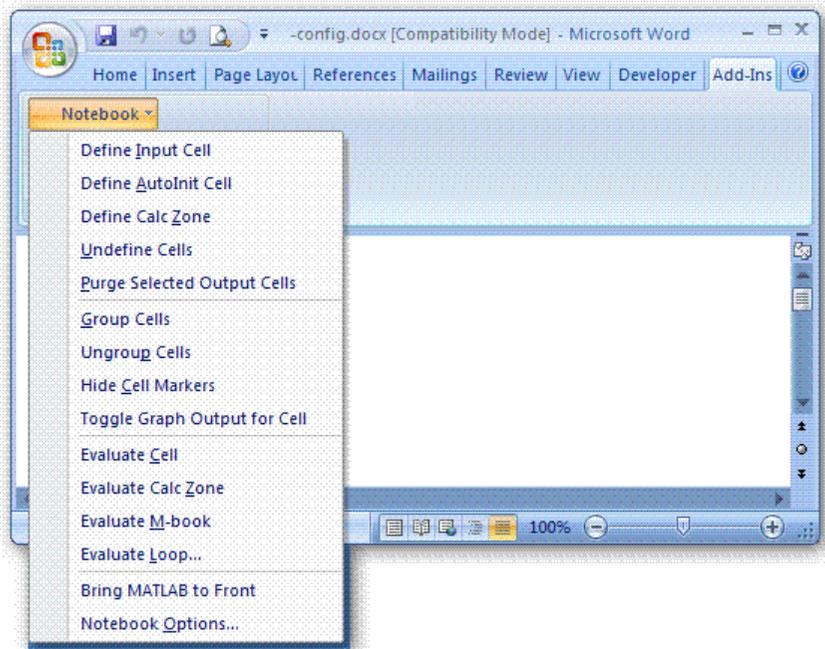
Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Creating an M-Book While Running Notebook

With Notebook running, you can create a new M-book as follows:

- In Word 2000, 2002, and 2003, select **File > New M-book**

- In Word 2007, select **Add-Ins > New M-book**, as shown in the following figure:
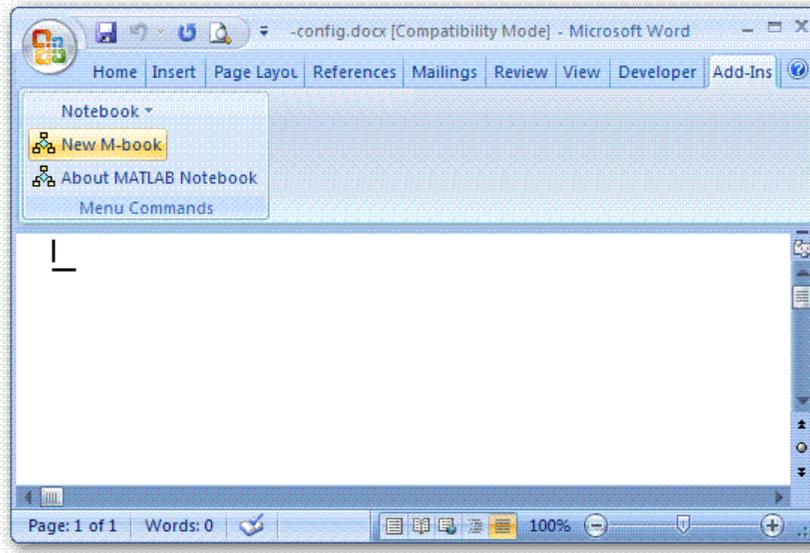
Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Opening an Existing M-Book

You can use the notebook command to open an existing M-book

```
notebook filename
```

where filename is the M-book you want to open. Or you can double-click an M-book file in a Windows file management tool, such as Explorer.
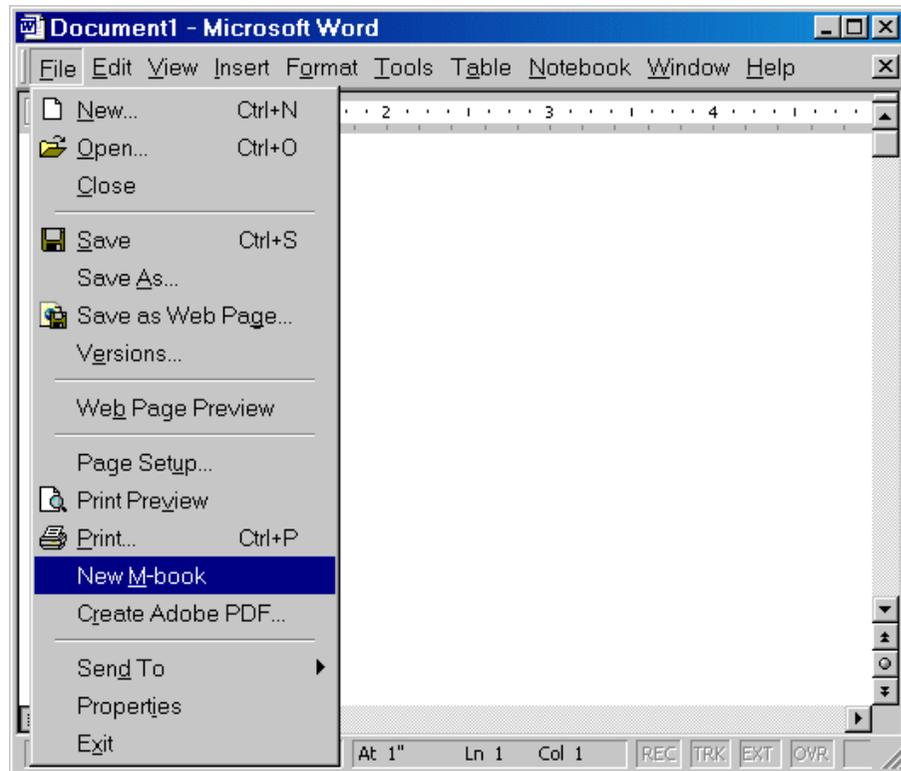
When you double-click on an M-book, Microsoft Word opens the M-book and starts MATLAB if it is not already running. Notebook adds the **Notebook** menu to the Word menu bar and adds **New M-book** to the **File** menu.

Converting a Word Document to an M-Book

To convert a Word document to an M-book, follow these steps, depending on the version of Word you are using:

- Microsoft Word 2000, 2002, or 2003:

  1 Create a new M-book.

  2 From the **Insert** menu, select **File**.

3 Select the file you want to convert.

4 Click **OK**.

- Microsoft Word 2007:

  1 Create a new M-book.

  2 From the **Insert** tab, in the **Text** group, click the arrow next to **Object** and then click **Text from File**, as shown in the image that follows.

    The Insert File dialog box opens.

  3 In the Insert File dialog box, select the file that you want to convert, and then click **OK**.
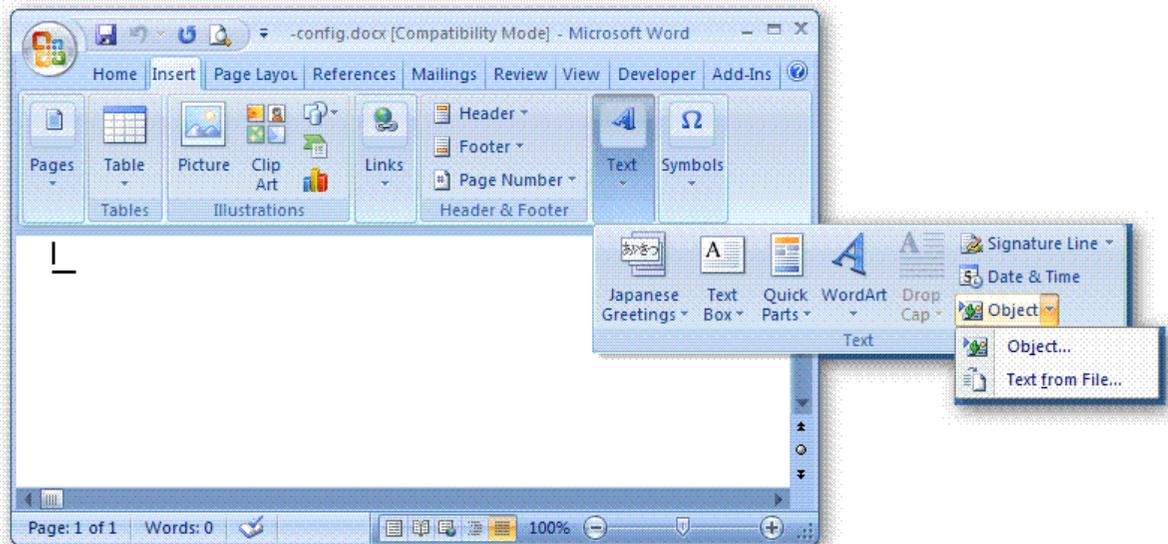
Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Entering MATLAB Commands in an M-Book

Note A good way to learn how to use Notebook is to open the sample M-book, `Readme.doc`, and try out the various techniques described in this section. You can find this file in the *matlabroot*`/notebook/pc` directory.

You enter MATLAB commands in an M-book the same way you enter text in any other Word document. For example, you can enter the following text in a Word document. The example uses text in Courier Font but you can use any font:

```
Here is a sample M-book.

a = magic(3)
```

To execute the MATLAB `magic` command in this document, you must

- Define the command as an input cell

- Evaluate the input cell

MATLAB displays the output of the command in the Word document in an output cell.

Protecting the Integrity of Your Workspace in M-Books

When you work on more than one M-book in a single word processing session, note that:

- Each M-book uses the same "copy" of MATLAB.

- All M-books share the same workspace.

If you use the same variable names in more than one M-book, data used in one M-book can be affected by another M-book. You can protect the integrity of your workspace by specifying the `clear` command as the first autoinit cell in the M-book.

Ensuring Data Consistency in M-Books

An M-book can be thought of as a sequential record of a MATLAB session. When executed in order, from the first MATLAB command to the last, the M-book accurately reflects the relationships among these commands.

If, however, you change an input cell or output cell as you refine your M-book, Notebook does not automatically recalculate input cells that depend on either the contents or the results of the changed cells. As a result, the M-book may contain inconsistent data.

When working on an M-book, you might find it useful to select **Evaluate M-book** periodically to ensure that your M-book data is consistent. You could also use calc zones to isolate related commands in a section of the M-book. You can then use **Evaluate Calc Zone** to execute only those input cells contained in the calc zone.

Debugging and Notebook

Do not use debugging functions or use the Editor/Debugger while evaluating cells with Notebook. Instead debug M-files from within MATLAB, and then after completing debugging, clear all the breakpoints and access the M-file via Notebook. If you debug while evaluating from Notebook, you might experience problems with MATLAB.

Defining MATLAB Commands as Input Cells for Notebook

In this section...
"Defining Commands as Input Cells for Notebook" on page 8-36
"Defining Cell Groups for Notebook" on page 8-37
"Defining Autoinit Input Cells for Notebook" on page 8-38
"Defining Calc Zones for Notebook" on page 8-38
"Converting an Input Cell to Text with Notebook" on page 8-39

For information about evaluating the input cells you define, see "Evaluating MATLAB Commands with Notebook" on page 8-41.

Defining Commands as Input Cells for Notebook

To define a MATLAB command in a Word document as an input cell,

1 Type the command into the M-book as text. For example,

```
This is a sample M-book.

a = magic(3)
```

2 Position the cursor anywhere in the command and select **Notebook > Define Input Cell** or press **Alt+D**. If the command is embedded in a line of text, use the mouse to select it. Notebook defines the MATLAB command as an input cell:

```
This is a sample M-book.

[a = magic(3)]
```

Note how Notebook changes the character font of the text in the input cell to a bold, dark green color and encloses it within *cell markers*. Cell markers are bold, gray brackets. They differ from the brackets used to enclose matrices by their size and weight. For information about changing these default formats, see "Modifying Styles in the M-Book Template" on page 8-47.

Defining Cell Groups for Notebook

You can collect several input cells into a single input cell. This is called a *cell group*. Because all the output from a cell group appears in a single output cell that Notebook places immediately after the group, cell groups are useful when several MATLAB commands are needed, such as, to fully define a graphic.

For example, if you define all the MATLAB commands that produce a graphic as a cell group and then evaluate the cell group, Notebook generates a single graphic that includes all the graphic components defined in the commands. If instead you define all the MATLAB commands that generate the graphic as separate input cells, evaluating the cells generates multiple graphic output cells.

See "Evaluating Cell Groups with Notebook" on page 8-42 for information about evaluating a cell group. For information about ungrouping a cell group, see "Ungroup Cells" on page 8-61.

Creating a Cell Group for Notebook

To create a cell group,

1 Use the mouse to select the input cells that are to make up the group.

2 Select **Notebook > Group Cells** or press **Alt+G**.

Notebook converts the selected cells into a cell group and replaces cell markers with a single pair that surrounds the group:

```
This is a sample cell group.

[date
a = magic(3) ]
```

Note the following:

- A cell group cannot contain output cells. If the selection includes output cells, Notebook deletes them.

- A cell group cannot contain text. If the selection includes text, Notebook places the text after the cell group. However, if the text precedes the first input cell in the selection, Notebook leaves it where it is.

- If you select part or all of an output cell but not its input cell, Notebook includes the input cell in the cell group.

When you create a cell group, Notebook defines it as an input cell unless its first line is an autoinit cell, in which case Notebook defines the group as an autoinit cell.

Defining Autoinit Input Cells for Notebook

You can use *autoinit cells* to specify MATLAB commands to be automatically evaluated each time an M-book is opened. This is a quick and easy way to initialize the workspace. *Autoinit cells* are simply input cells with the following additional characteristics:

- Notebook evaluates the autoinit cells when it opens the M-book.

- Notebook displays the commands in autoinit cells using dark blue characters.

Autoinit cells are otherwise identical to input cells.

Creating an Autoinit Cell for Notebook

You can create an autoinit cell in two ways:

- Enter the MATLAB command as text, then convert the command to an autoinit cell by selecting **Notebook > Define AutoInit Cell**.

- If you already entered the MATLAB command as an input cell, you can convert the input cell to an autoinit cell. Either select the input cell or position the cursor in the cell, then select **Notebook > Define AutoInit Cell**.

See "Evaluating MATLAB Commands with Notebook" on page 8-41 for information about evaluating autoinit cells.

Defining Calc Zones for Notebook

You can partition an M-book into self-contained sections, called *calc zones*. A calc zone is a contiguous block of text, input cells, and output cells. Notebook inserts Microsoft Word section breaks before and after the section to define

the calc zone. The section break indicators include bold, gray brackets to distinguish them from standard Word section breaks.

You can use calc zones to prepare problem sets, making each problem a separate calc zone that can be created and tested on its own. An M-book can contain any number of calc zones.

Note Using calc zones does not affect the scope of the variables in an M-book. Variables used in one calc zone are accessible to all calc zones.

Creating a Calc Zone

After you create the text and cells you want to include in the calc zone, you define the calc zone by following these steps:

1 Select the input cells and text to be included in the calc zone.

2 Select **Notebook > Define Calc Zone**.

Note You must select an input cell and its output cell in their entirety to include them in the calc zone.

See "Evaluating a Calc Zone with Notebook" on page 8-44 for information about evaluating a calc zone.

Converting an Input Cell to Text with Notebook

To convert an input cell (or an autoinit cell or a cell group) to text,

1 Select the input cell with the mouse or position the cursor in the input cell.

2 Select **Notebook > Undefine Cells** or press **Alt+U**.

When Notebook converts the cell to text, it reformats the cell contents according to the Microsoft Word Normal style. For more information about M-book styles, see "Modifying Styles in the M-Book Template" on page

8-47. When you convert an input cell to text, Notebook also converts the corresponding output cell to text.

Evaluating MATLAB Commands with Notebook

In this section...
"Evaluating Input Commands with Notebook" on page 8-41
"Evaluating Cell Groups with Notebook" on page 8-42
"Evaluating a Range of Input Cells with Notebook" on page 8-43
"Evaluating a Calc Zone with Notebook" on page 8-44
"Evaluating an Entire M-Book" on page 8-44
"Using a Loop to Evaluate Input Cells Repeatedly with Notebook" on page 8-45
"Converting Output Cells to Text with Notebook" on page 8-46
"Deleting Output Cells with Notebook" on page 8-46

Evaluating Input Commands with Notebook

After you define a MATLAB command as an input cell, or as an autoinit cell, you can evaluate it in your M-book. Use the following steps to define and evaluate a MATLAB command:

1 Type the command into the M-book as text. For example:

```
This is a sample M-book

a = magic(3)
```

2 Position the cursor anywhere in the command. If the command is embedded in a line of text, use the mouse to select it. Then select **Notebook > Define Input Cell** or press **Alt+D**.

Notebook defines the MATLAB command as an input cell. For example:

```
This is a sample M-book

[a = magic(3)]
```

3 Specify the input cell to be evaluated by selecting it with the mouse or by placing the cursor in it. Then select **Notebook > Evaluate Cell** or press **Ctrl+Enter**.

Notebook evaluates the input cell and displays the results in a output cell immediately following the input cell. If there is already an output cell, Notebook replaces its contents, wherever it is in the M-book. For example:

```
This is a sample M-book.

[a = magic(3) ]

[a =
     8     1     6
     3     5     7
     4     9     2  ]
```

The text in the output cell is blue and is enclosed within cell markers. Cell markers are bold, gray brackets. They differ from the brackets used to enclose matrices by their size and weight. Error messages appear in red. For information about changing these default formats, see "Modifying Styles in the M-Book Template" on page 8-47.

Evaluating Cell Groups with Notebook

You evaluate a cell group the same way you evaluate an input cell (because a cell group is an input cell):

1 Position the cursor anywhere in the cell or in its output cell.

2 Select **Notebook > Evaluate Cell** or press **Ctrl+Enter**.

For information about creating a cell group, see "Defining Cell Groups for Notebook" on page 8-37.

When MATLAB evaluates a cell group, the output for all commands in the group appears in a single output cell. By default, Notebook places the output cell immediately after the cell group the first time the cell group is evaluated. If you evaluate a cell group with an existing output cell, Notebook places the results in the output cell wherever it is located in the M-book.

Note Text or numeric output always comes first, regardless of the order of the commands in the group.

The illustration shows a cell group and the figure created when you evaluate the cell group.

This is a sample M-book with a cell group.

```
[t = 0:pi/100:2*pi;
y = sin(t);
plot(t,y) ]
```
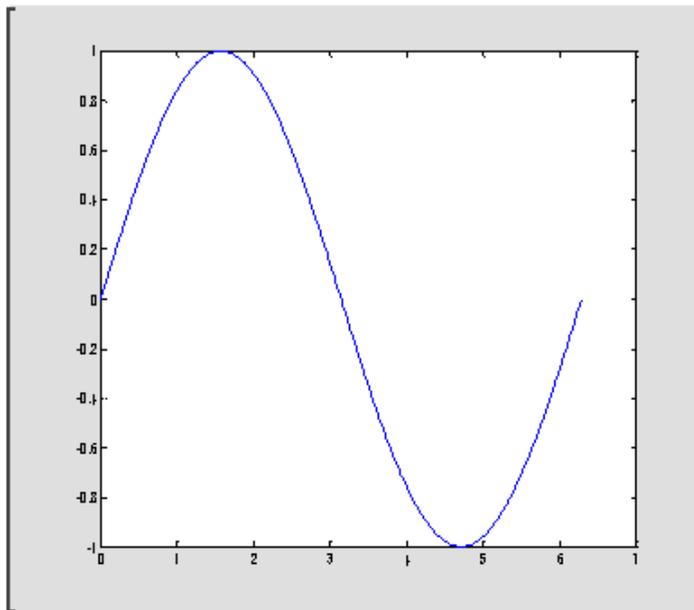

Evaluating a Range of Input Cells with Notebook

To evaluate more than one MATLAB command contained in different but contiguous input cells,

1 Select the range of cells that includes the input cells you want to evaluate. You can include text that surrounds input cells in your selection.

2 Select **Notebook > Evaluate Cell** or press **Ctrl+Enter**.

Notebook evaluates each input cell in the selection, inserting new output cells or replacing existing ones.

Evaluating a Calc Zone with Notebook

To evaluate a calc zone,

1 Position the cursor anywhere in the calc zone.

2 Select **Notebook > Evaluate Calc Zone** or press **Alt+Enter**.

For information about creating a calc zone, see "Defining Calc Zones for Notebook" on page 8-38.

By default, Notebook places the output cell immediately after the calc zone the first time the calc zone is evaluated. If you evaluate a calc zone with an existing output cell, Notebook places the results in the output cell wherever it is located in the M-book.

Evaluating an Entire M-Book

To evaluate the entire M-book, either select **Notebook > Evaluate M-book** or press **Alt+R**.

Notebook begins at the top of the M-book regardless of the cursor position and evaluates each input cell in the M-book. As it evaluates the M-book, Notebook inserts new output cells or replaces existing output cells.

Controlling Execution of Multiple Commands

When you evaluate an entire M-book, and an error occurs, evaluation continues. If you want to stop evaluation if an error occurs, follow this procedure:

1 Select **Notebook > Notebook Options**.

The **Notebook Options** dialog box opens.

2 Select the **Stop evaluating on error** check box and click **OK**.

Using a Loop to Evaluate Input Cells Repeatedly with Notebook

To evaluate a sequence of MATLAB commands repeatedly,

1 Use the mouse to select the input cells, including any text or output cells located between them.

2 Select **Notebook > Evaluate Loop** or press **Alt+L**. Notebook displays the **Evaluate Loop** dialog box.
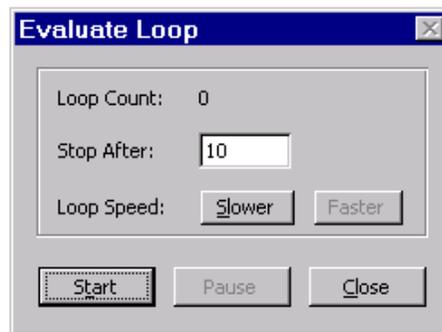
3 Enter the number of times you want MATLAB to evaluate the selected commands in the **Stop After** field, then click **Start**. The button changes to **Stop**. Notebook begins evaluating the commands and indicates the number of completed iterations in the **Loop Count** field.

You can increase or decrease the delay at the end of each iteration by clicking **Slower** or **Faster**. Slower increases the delay. Faster decreases the delay.

To suspend evaluation of the commands, click **Pause**. The button changes to **Resume**. Click **Resume** to continue evaluation.

To stop processing the commands, click **Stop**. To close the **Evaluate Loop** dialog box, click **Close**.

8-45

Converting Output Cells to Text with Notebook

You can convert an output cell to text by undefining cells. If the output is numeric or textual, Notebook removes the cell markers and converts the cell contents to text according to the Microsoft Word Normal style. If the output is graphical, Notebook removes the cell markers and dissociates the graphic from its input cell, but does not alter its contents.

Note Undefining an output cell does not affect the associated input cell.

To undefine an output cell,

1 Select the output cell you want to undefine.

2 Select **Notebook > Undefine Cells** or press **Alt+U**.

Deleting Output Cells with Notebook

To delete output cells,

1 Select an output cell, using the mouse, or place the cursor in the output cell.

2 Select **Notebook > Purge Selected Output Cells** or press **Alt+P**.

If you select a range of cells, Notebook deletes all the output cells in the selected range, but any associate input cells remain intact.

Printing and Formatting an M-Book

In this section...
"Printing an M-Book" on page 8-47
"Modifying Styles in the M-Book Template" on page 8-47
"Choosing Loose or Compact Format for Notebook" on page 8-48
"Controlling Numeric Output Format for Notebook" on page 8-49
"Controlling Graphic Output for Notebook" on page 8-49

Printing an M-Book

You can print all or part of an M-book by doing one of the following, depending on the version of Microsoft Word you are using:

- In Microsoft Word 2000, 2002, 2003 — Select **File > Print**.

- In Microsoft Word 2007 — Select **Microsoft Office Button > Print**

Word follows these rules when printing M-book cells and graphics:

- Cell markers are not printed.

- Input cells, autoinit cells, and output cells (including error messages) are printed according to their defined styles. If you prefer to print these cells using black type instead of colors or shades of gray, you can modify the styles.

Modifying Styles in the M-Book Template

You can control the appearance of the text in your M-book by modifying the predefined styles stored in the M-book template, `m-book.dot`. These styles control the appearance of text and cells. By default, M-books use the Word Normal style for all other text.

For example, if you print an M-book on a color printer, input cells appear dark green, output and autoinit cells appear dark blue, and error messages appear red. If you print the M-book on a grayscale printer, these cells appear as

shades of gray. To print these cells using black type, you need to modify the color of the Input, Output, AutoInit, and Error styles in the M-book template.

The table below describes the default styles used by Notebook. If you modify styles, you can use the information in the tables below to help you return the styles to their original settings. For general information about using styles in Word documents, see the Word documentation.

Style	Font	Size	Weight	Color
Normal	Times New Roman	10 points	N/A	Black
AutoInit	Courier New	10 points	Bold	Dark blue
Error	Courier New	10 points	Bold	Red
Input	Courier New	10 points	Bold	Dark green
Output	Courier New	10 points	N/A	Blue

When you change a style, Word applies the change to all characters in the M-book that use that style and gives you the option to change the template. Be cautious about making changes to the template. If you choose to apply the changes to the template, you will affect all new M-books you create using the template. See the Word documentation for more information.

Choosing Loose or Compact Format for Notebook

You can specify whether a blank line appears between the input and output cells by selecting the loose or compact format:

1 Select **Notebook > Notebook Options**.

2 In the **Notebook Options** dialog box, select either **Loose** or **Compact**. Loose format adds an empty line. Compact format does not.

3 Click **OK**.

Note Changes you make using the **Notebook Options** dialog box take effect for output generated *after* you click **OK**. To affect existing input or output cells, you must reevaluate the cells.

Controlling Numeric Output Format for Notebook

To change how Notebook displays numeric output,

1 Select **Notebook > Notebook Options**.

2 In the **Notebook Options** dialog box, select a format from the **Numeric Format** list. These settings correspond to the choices available with the MATLAB `format` command.

3 Click **OK**.

Note Changes you make using the **Notebook Options** dialog box take effect for output generated *after* you click **OK**. To affect existing input or output cells, you must reevaluate the cells.

Controlling Graphic Output for Notebook

This section describes how to control several aspects of the graphic output produced by MATLAB commands in an M-book, including

- "Embedding Graphic Output in the M-Book" on page 8-49
- "Suppressing Graphic Output for Individual Input Cells in Notebook" on page 8-50
- "Sizing Graphic Output in Notebook" on page 8-51
- "Cropping Graphic Output in Notebook" on page 8-51
- "Adding White Space Around Graphic Output in Notebook" on page 8-52

Embedding Graphic Output in the M-Book

By default, graphic output is embedded in an M-book. To display graphic output in a separate figure window,

1 Select **Notebook > Notebook Options**.

2 In the **Notebook Options** dialog box, clear the **Embed Figures in M-book** check box.
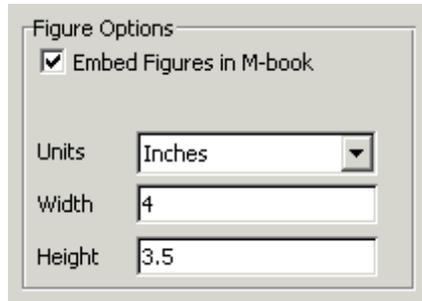
3 Click **OK**.

Note Embedded figures do not include Handle Graphics objects generated by the `uicontrol` and `uimenu` functions.

Notebook determines whether to embed a figure in the M-book by examining the value of the figure object's `Visible` property. If the value of the property is `off`, Notebook embeds the figure. If the value of this property is `on`, all graphic output is directed to the current figure window.

Suppressing Graphic Output for Individual Input Cells in Notebook

If an input or autoinit cell generates figure output that you want to suppress,

1 Place the cursor in the input cell.

2 Select **Notebook > Toggle Graph Output for Cell**.

Notebook suppresses graphic output from the cell, inserting the string `(no graph)` after the input cell.

To allow graphic output for a cell, repeat the procedure. Notebook removes the (no graph) marker and allows graphic output from the cell.

Note **Toggle Graph Output for Cell** overrides the **Embed Figures in M-book** option, if that option is set.

Sizing Graphic Output in Notebook

To set the default size of embedded graphics in an M-book,

1 Select **Notebook > Notebook Options**.

2 In the **Notebook Options** dialog box, use the **Units**, **Width** and **Height** fields to set the size of graphics generated by the M-book.

3 Click **OK**.

Note Changes you make using the **Notebook Options** dialog box take effect for graphic output generated *after* you click **OK**. To affect existing input or output cells, you must reevaluate the cells.

You change the size of an existing embedded figure by selecting the figure, clicking the left mouse button anywhere in the figure, and dragging the resize handles of the figure. If you resize an embedded figure using its resize handles and then regenerate the figure, its size reverts to its original size.

Cropping Graphic Output in Notebook

To crop an embedded figure to cut off areas you do not want to show,

1 Select the graphic by clicking the left mouse button anywhere in the figure.

2 Hold down the **Shift** key.

3 Drag a sizing handle toward the center of the graphic.

Adding White Space Around Graphic Output in Notebook

You can add white space around an embedded figure by moving the boundaries of a graphic outward. Select the graphic, then hold down the **Shift** key and drag a sizing handle away from the graphic.

Configuring Notebook

After you install Notebook but before you begin using it, you must configure it. (Notebook is installed as part of the MATLAB installation process on Windows platforms. For more information, see the MATLAB installation documentation for your platform.)

Before configuring Notebook, you must specify that Word can use the Notebook macros.

- In Word 2000, 2002, and 2003 do either of the following:
  - Set the macro security level to medium: in Word, select **Tools > Macros > Security**, and in the resulting dialog box, choose **Medium**.
  - After starting Notebook, when Word first opens, a security warning dialog box appears. In the dialog box, select **Always trust macros from this source**. This allows you to use Notebook, but still maintain a high security level for other macros you use in Word.
- In Word 2007, follow the Word help instructions in the topic entitled "Enable or disable macros in Office documents."

To configure Notebook, type the following in the MATLAB Command Window:

```
notebook ('-setup')
```

MATLAB accesses the Windows system registry to locate Microsoft Word and the Word templates directory, and to identify the version of Word. MATLAB then copies Notebook's m-book.dot template to the Word templates directory. MATLAB Notebook supports Word versions 2000, 2002, 2003, and 2007.

When Notebook setup successfully finishes, MATLAB displays the message

```
Setup complete
```

Notebook Feature Reference

This section provides reference information about each of the Notebook features, listed alphabetically. To use these features, select them from the **Notebook** menu in Microsoft Word. (In Word 2007, the **Notebook** menu is on the **Add-Ins** tab.)

Bring MATLAB to Front

Bring MATLAB to Front brings the MATLAB Command Window to the foreground.

Define Autoinit Cell

Define AutoInit Cell creates an autoinit cell by converting the current paragraph, selected text, or input cell. An autoinit cell is an input cell that is automatically evaluated whenever you open an M-book.

Result

If you select this feature while the cursor is in a paragraph of text, Notebook converts the entire paragraph to an autoinit cell. If you select this feature while text is selected, Notebook converts the text to an autoinit cell. If you select this feature while the cursor is in an input cell, Notebook converts the input cell to an autoinit cell.

Format

Notebook formats the autoinit cell using the AutoInit style, defined as bold, dark blue, 10-point Courier New.

See Also

For more information about autoinit cells, see "Defining Autoinit Input Cells for Notebook" on page 8-38.

Define Calc Zone

Define Calc Zone defines the selected text, input cells, and output cells as a calc zone. A calc zone is a contiguous block of related text, input cells, and output cells that describes a specific operation or problem.

Result

Notebook defines a calc zone as a Word document section, placing section breaks before and after the calc zone. However, Word does not display section breaks at the beginning or end of a document.

See Also

For information about evaluating calc zones, see "Evaluating a Calc Zone with Notebook" on page 8-44. For more information about document sections, see the Microsoft Word documentation.

Define Input Cell

Define Input Cell creates an input cell by converting the current paragraph, selected text, or autoinit cell. An input cell contains a MATLAB command.

Result

If you select this feature while the cursor is in a paragraph of text, Notebook converts the entire paragraph to an input cell. If you select this feature while text is selected, Notebook converts the text to an input cell. If you select this feature while the cursor is in an autoinit cell, Notebook converts the autoinit cell to an input cell.

Format

Notebook encloses the text in cell markers and formats the cell using the Input style, defined as bold, dark green, 10-point Courier New.

See Also

For more information about creating input cells, see "Defining MATLAB Commands as Input Cells for Notebook" on page 8-36. For information about evaluating input cells, see "Evaluating MATLAB Commands with Notebook" on page 8-41.

Evaluate Calc Zone

Evaluate Calc Zone sends the input cells in the current calc zone to MATLAB to be evaluated. The current calc zone is the Word section that contains the cursor.

Result

As Notebook evaluates each input cell, it generates an output cell. When you evaluate an input cell for which there is no output cell, Notebook places the output cell immediately after the input cell that generated it. If you evaluate an input cell for which there is an output cell, Notebook replaces the results in the output cell wherever it is in the M-book.

See Also

For more information, see "Evaluating a Calc Zone with Notebook" on page 8-44.

Evaluate Cell

Evaluate Cell sends the current input cell or cell group to MATLAB to be evaluated. An input cell contains a MATLAB command. A cell group is a single, multiline input cell that contains more than one MATLAB command. Notebook displays the output or an error message in an output cell.

Result

If you evaluate an input cell for which there is no output cell, Notebook places the output cell immediately after the input cell that generated it. If you evaluate an input cell for which there is an output cell, Notebook replaces the results in the output cell wherever it is in the M-book. If you evaluate a cell group, all output for the cell appears in a single output cell.

An input cell or cell group is the current input cell or cell group if

- The cursor is in the input cell or cell group.

- The cursor is at the end of the line that contains the closing cell marker for the input cell or cell group.

- The cursor is in the output cell for the input cell or cell group.

- The input cell or cell group is selected.

Note Evaluating a cell that involves a lengthy operation may cause a time-out. If this happens, Word displays a time-out message and asks whether you want to continue waiting for a response or terminate the request. If you choose to continue, Word resets the time-out value and continues waiting for a response. Word sets the time-out value; you cannot change it.

See Also

For more information, see "Evaluating MATLAB Commands with Notebook" on page 8-41. For information about evaluating the entire M-book, see "Evaluating an Entire M-Book" on page 8-44.

Evaluate Loop

Evaluate Loop evaluates the selected input cells repeatedly.

For more information, see "Using a Loop to Evaluate Input Cells Repeatedly with Notebook" on page 8-45.

Evaluate M-Book

Evaluate M-book evaluates the entire M-book, sending all input cells to MATLAB to be evaluated. Notebook begins at the top of the M-book regardless of the cursor position.

Result

As Notebook evaluates each input cell, it generates an output cell. When you evaluate an input cell for which there is no output cell, Notebook places the output cell immediately after the input cell that generated it. If you evaluate an input cell for which there is an output cell, Notebook replaces the results in the output cell wherever it is in the M-book.

See Also

For more information, see "Evaluating an Entire M-Book" on page 8-44.

Group Cells

Group Cells converts the input cells in the selection into a single multiline input cell called a cell group. You evaluate a cell group using **Evaluate Cell**. When you evaluate a cell group, all of its output follows the group and appears in a single output cell.

Result

If you include text in the selection, Notebook moves it after the cell group. However, if text precedes the first input cell in the group, the text will remain before the group.

If you include output cells in the selection, Notebook deletes them. If you select all or part of an output cell before selecting this feature, Notebook includes its input cell in the cell group.

If the first line in the cell group is an autoinit cell, the entire group acts as a sequence of autoinit cells. Otherwise, the group acts as a sequence of input cells. You can convert an entire cell group to an autoinit cell by using **Define AutoInit Cell**.

See Also

For more information, see "Defining Cell Groups for Notebook" on page 8-37. For information about converting a cell group to individual input cells, see the description of the "Ungroup Cells" on page 8-61.

Hide Cell Markers

Hide Cell Markers hides cell markers in the M-book.

When you select this feature, it changes to **Show Cell Markers**.

Note Notebook does not print cell markers whether you choose to hide them or show them on the screen.

Notebook Options

Notebook Options allows you to examine and modify display options for numeric and graphic output.

See Also

See "Printing and Formatting an M-Book" on page 8-47 for more information.

Purge Selected Output Cells

Purge Selected Output Cells deletes all output cells from the current selection.

See Also

For more information, see "Deleting Output Cells with Notebook" on page 8-46.

Toggle Graph Output for Cell

Toggle Graph Output for Cell suppresses or allows graphic output from an input cell.

If an input or autoinit cell generates figure output that you want to suppress, place the cursor in the input cell and choose this feature. The string (no graph) will be placed after the input cell to indicate that graph output for that cell will be suppressed.

To allow graphic output for that cell, place the cursor inside the input cell and choose **Toggle Graph Output for Cell** again. The (no graph) marker will be removed. This feature overrides the **Embed Figures in M-book** option, if that option is set in the **Notebook Options** dialog box.

See Also

See "Embedding Graphic Output in the M-Book" on page 8-49 and "Suppressing Graphic Output for Individual Input Cells in Notebook" on page 8-50 for more information.

Undefine Cells

Undefine Cells converts the selected cells to text. If no cells are selected but the cursor is in a cell, Notebook undefines that cell. Notebook removes the cell markers and reformats the cell according to the Normal style.

If you undefine an input cell, Notebook automatically undefines its output cell. However, if you undefine an output cell, Notebook does not undefine its input cell. If you undefine an output cell containing an embedded graphic, the graphic remains in the M-book but is no longer associated with an input cell.

See Also

For information about the Normal style, see "Modifying Styles in the M-Book Template" on page 8-47. For information about deleting output cells, see the description of the "Purge Selected Output Cells" on page 8-60.

Ungroup Cells

Ungroup Cells converts the current cell group into a sequence of individual input cells or autoinit cells. If the cell group is an input cell, Notebook converts the cell group to input cells. If the cell group is an autoinit cell, Notebook converts the cell group to autoinit cells. Notebook deletes the output cell for the cell group.

A cell group is the current cell group if

- The cursor is in the cell group.
- The cursor is at the end of a line that contains the closing cell marker for the cell group.
- The cursor is in the output cell for the cell group.
- The cell group is selected.

See Also

For information about creating cell groups, see the description of the "Defining Cell Groups for Notebook" on page 8-37.

Source Control Interface

The source control interface provides access to your source control system from MATLAB. Source control systems, also known as version control, revision control, configuration management, and file management systems, are platform dependent — the topics for the Windows platforms appear first, followed by the topics for the UNIX platforms.

Source Control Interface on Windows

If you use source control systems to manage your files, you can interface with the systems to perform source control actions from within MATLAB, Simulink, and Stateflow®. Use menu items in MATLAB, Simulink, or Stateflow, or run functions in the MATLAB Command Window to interface with your source control systems.

The source control interface on Windows works with any source control system that conforms to the Microsoft Common Source Control standard, Version 1.1. If your source control system does not conform to the standard, use a Microsoft Source Code Control API wrapper product for your source control system so that you can interface with it from MATLAB, Simulink, and Stateflow.

Perform most source control interface actions from the Current Directory browser. You can also perform many of these actions for a single file from the MATLAB Editor/Debugger, a Simulink model window, or a Stateflow chart window — for more information, see "Performing Source Control Actions from the Editor/Debugger, Simulink, or Stateflow on Windows" on page 9-23. Another way to access many of the source control actions is with the `verctrl` function.

Setting Up the Source Control Interface on Windows

In this section...
"Create Projects in Source Control System" on page 9-4
"Specify Source Control System in MATLAB" on page 9-6
"Register Source Control Project with MATLAB" on page 9-7
"Add Files to Source Control" on page 9-9

Create Projects in Source Control System

In your source control system, create the projects that your directories and files will be associated with.

All files in a directory must belong to the same source control project. Be sure the working directory for the project in the source control system specifies the correct pathname to the directory on disk.

Example of Creating Source Control Project

This example uses the project my_thesis_files in Microsoft Visual SourceSafe. This illustration of the Current Directory browser shows the pathname to the directory on disk, D:\my_thesis_files.
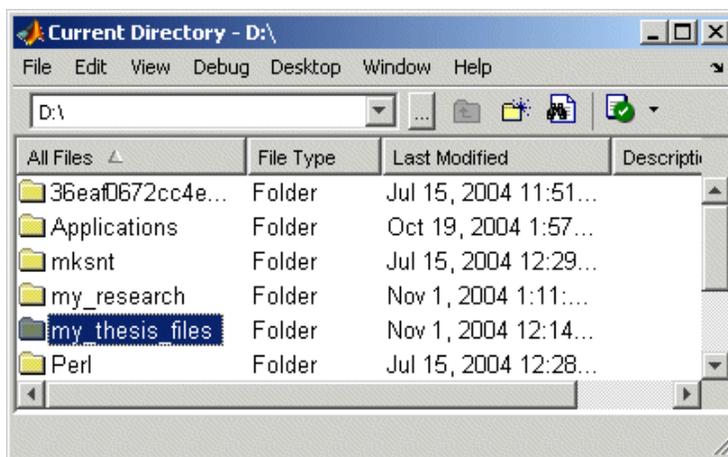
The following illustration shows the example project in the source control system.
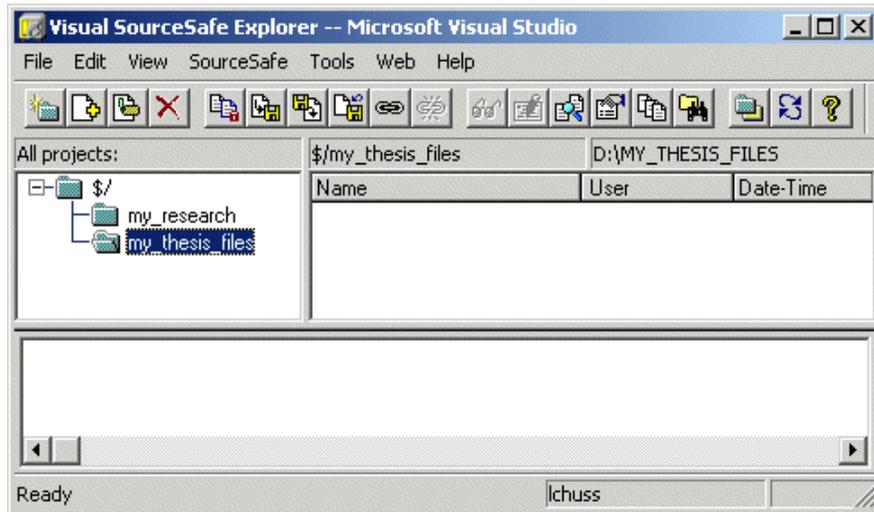
To set the working directory in Microsoft Visual SourceSafe for this example, select my_thesis_files, right-click, select **Set Working Folder** from the context menu, and specify D:\my_thesis_files in the resulting dialog box.
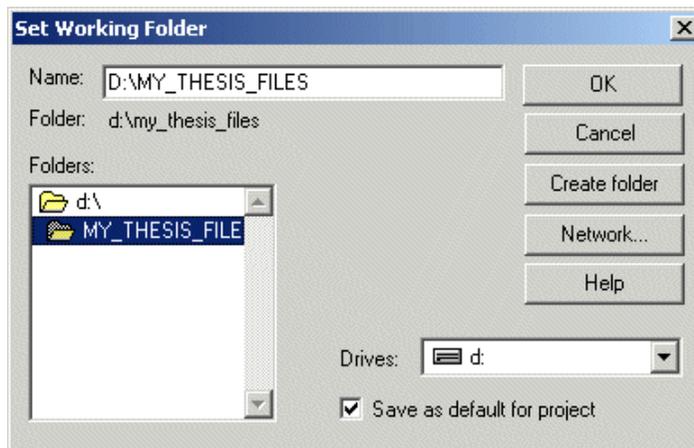
Specify Source Control System in MATLAB

In MATLAB, specify the source control system you want to access. Select **File > Preferences > General > Source Control**.

The currently selected system is shown in the Preferences dialog box. The list includes all installed source control systems that support the Microsoft Common Source Control standard.

Select the source control system you want to interface with and click **OK**.
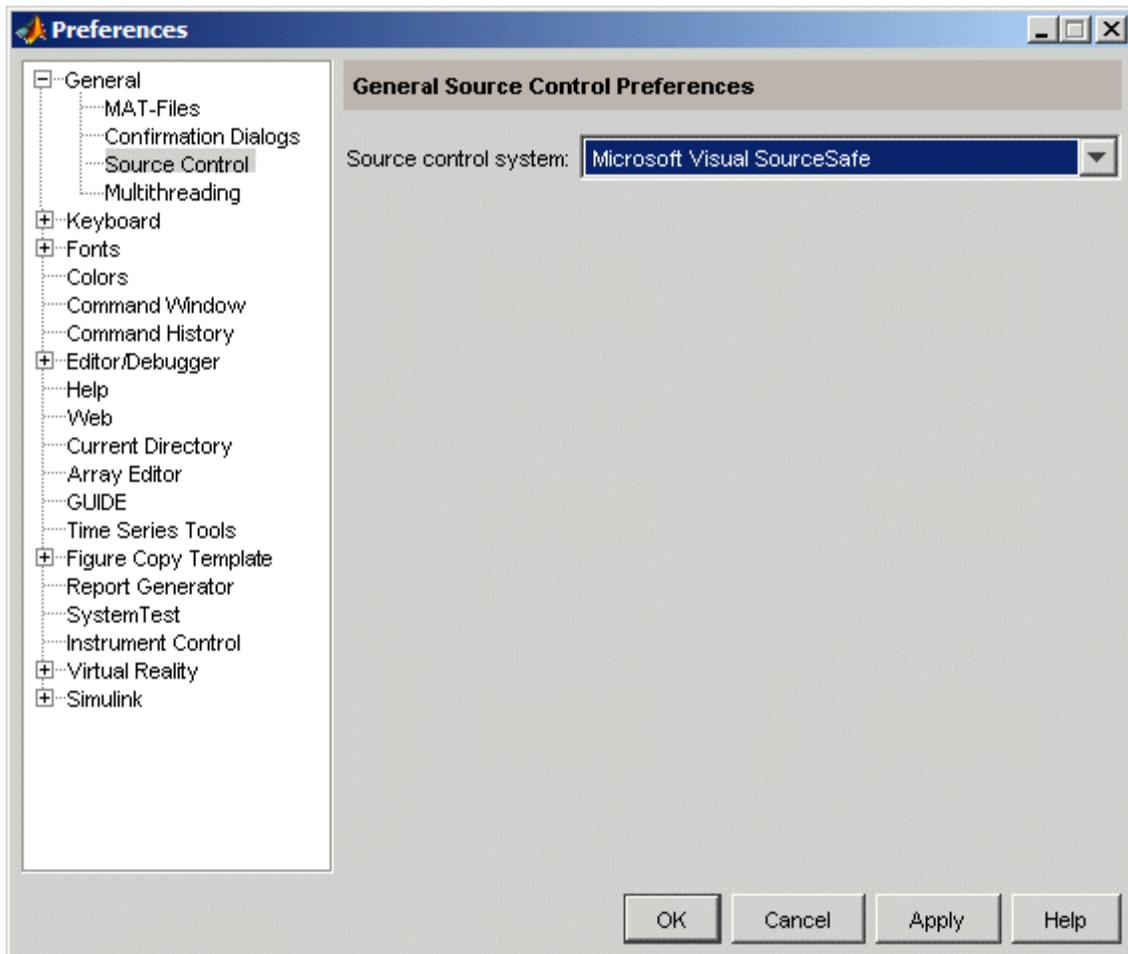
MATLAB remembers preferences between sessions, so you only need to perform this action again when you want to access a different source control system.

Function Alternative

A function alternative to select a source control system is not available, but you can list all available source control systems using `verctrl` with the `all_systems` argument. Use `cmopts` to display the name of the currently selected source control system.

Register Source Control Project with MATLAB

Register a source control system project with a directory in MATLAB, that is, associate a source control system project with a directory and all files in that directory. Do this only one time for any file in the directory, which registers all files in that directory:

1 In the MATLAB Current Directory browser, select a file that is in the directory you want to associate with a project in your source control system. For example, select `D:\my_thesis_files\wind.m`. This will associate all files in the `my_thesis_files` directory.

2 Right-click, and from the context menu, select **Source Control > Register Name_of_Source_Control_System Project with MATLAB**. The **Name_of_Source_Control_System** is the source control system you selected using preferences as described in "Specify Source Control System in MATLAB" on page 9-6.

The following example shows Microsoft Visual SourceSafe.
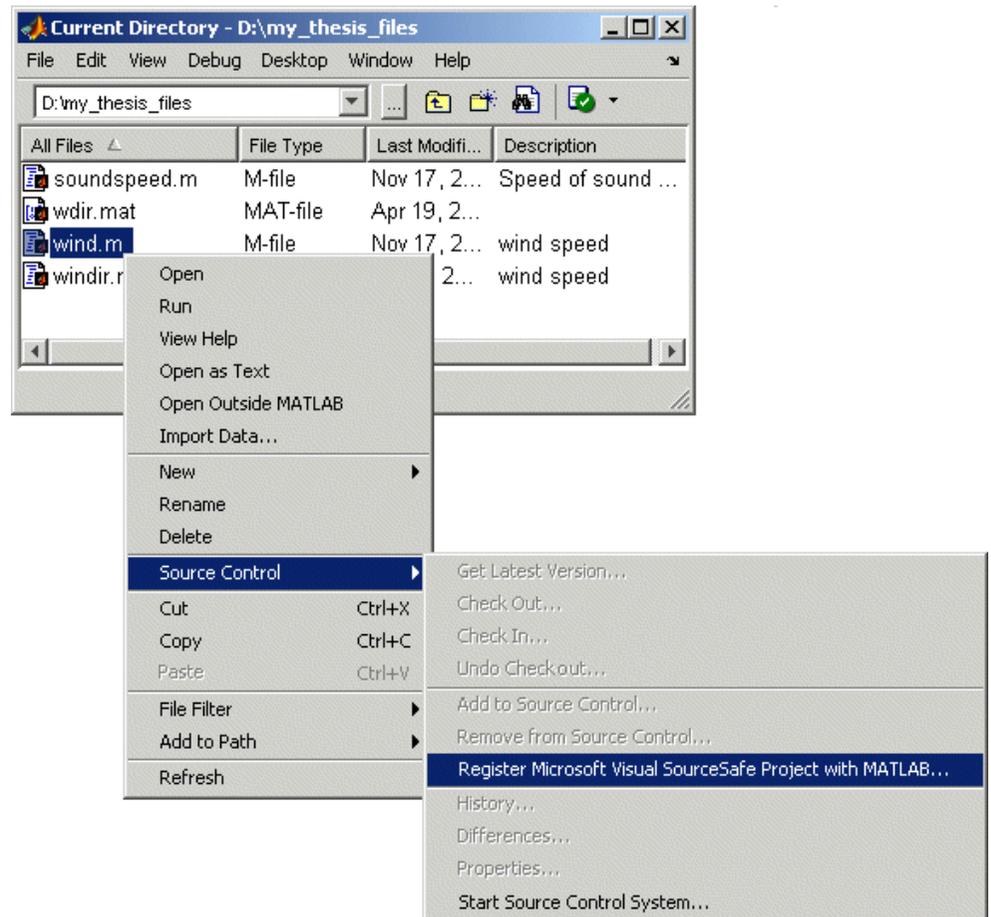
3 In the resulting **Name_of_Source_Control_System Login** dialog box, provide the username and password you use to access your source control system, and click **OK**.

Visual SourceSafe Login

Username:	my_name
Password:	*******
Database:	Microsoft Visual Studio Browse...

OK   Cancel   Help

4 In the resulting **Choose project from
Name_of_Source_Control_System** dialog box, select
the source control system project to associate with the directory and click
OK. This example shows my_thesis_files.

Choose project from SourceSafe

Working folder:

D:\my_thesis_files

SourceSafe project to use:

$/thesis

OK

Cancel

Help

$/
  my_research
  my_thesis_files
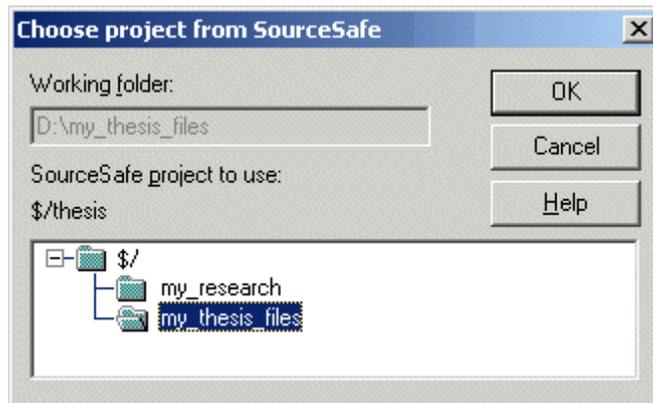
The selected file, its directory, and all files in the directory, are associated
with the source control system project you selected. For the example,
MATLAB associates all files in D:\my_thesis_files with the source
control project my_thesis_files.

Add Files to Source Control

Add files to the source control system. Do this only once for each file:

1 In the Current Directory browser, select files you want to add to the source
control system.

2 Right-click, and from the context menu, select **Source Control > Add to Source Control**.

3 The resulting **Add to source control** dialog box lists files you selected to add. You can add text in the **Comments** field. If you expect to use the files soon, select the **Keep checked out** check box (which is selected by default). Click **OK**.
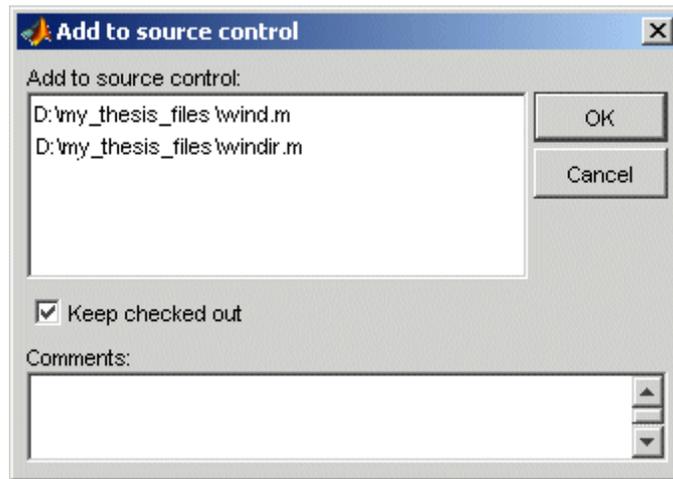
If you try to add an unsaved file, the file is automatically saved upon adding.

Function Alternative

The function alternative is `verctrl` with the `add` argument.

Checking Files Into and Out of Source Control from MATLAB on Windows

In this section...
"Check Files Into Source Control" on page 9-11
"Check Files Out of Source Control" on page 9-12
"Undoing the Checkout" on page 9-13

Before checking files into and out of your source control system from MATLAB, be sure to set up your system for use with MATLAB as described in "Setting Up the Source Control Interface on Windows" on page 9-4.

Check Files Into Source Control

After creating or modifying files in MATLAB or related products, check the files into the source control system by performing these steps:

1 In the Current Directory browser, select the files to check in. A file can be open or closed when you check it in, but it must be saved, that is, it cannot contain unsaved changes.

2 Right-click, and from the context menu, select **Source Control > Check In**.

3 In the resulting **Check in file(s)** dialog box, you can add text in the **Comments** field. If you want to continue working on the files, select the check box **Keep checked out**. Click **OK**.

If a file contains unsaved changes when you try to check it in, you will be prompted to save the changes to complete the checkin. If you did not keep the file checked out and you keep the file open, note that it is a read-only version.

Function Alternative

The function alternative is verctrl with the checkin argument.

Check Files Out of Source Control

From MATLAB, to check out the files you want to modify, perform these steps:

1 In the Current Directory browser, select the files to check out.

2 Right-click, and from the context menu, select **Source Control > Check Out**.

3 The resulting **Check out file(s)** dialog box lists files you selected to check out. Enter comment text in the **Comments** field, which appears if your source control system supports comments on checkout. Click **OK**.
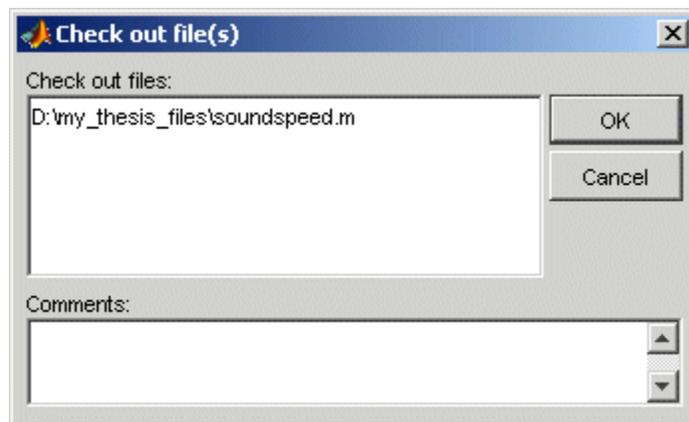
After checking out a file, make changes to it in MATLAB or another product, and save the file. For example, edit an M-file in the Editor/Debugger.

If you try to change a file without first having checked it out, the file is read-only, as seen in the title bar, and you will not be able to save any changes. This protects you from accidentally overwriting the source control version of the file.

If you end the MATLAB session, the file remains checked out. You can check in the file from within MATLAB during a later session, or directly from your source control system.

Function Alternative

The function alternative is `verctrl` with the `checkout` argument.

Undoing the Checkout

You can undo the checkout for files. The files remain checked in, and do not have any of the changes you made since you last checked them out. To save any changes you have made since checking out a particular file select **File > Save As**, and supply a different filename before you undo the checkout.

To undo a checkout, follow these steps:

1 In the MATLAB **Current Directory** browser, select the files for which you want to undo the checkout.

2 Right-click, and from the context menu, select **Source Control > Undo Checkout**.

The MATLAB **Undo checkout** dialog box opens, listing the files you selected.

3 Click **OK**.

Function Alternative

The function alternative is `verctrl` with the `undocheckout` argument.

Additional Source Control Actions on Windows

In this section...
"Getting the Latest Version of Files for Viewing or Compiling" on page 9-14
"Removing Files from the Source Control System" on page 9-15
"Showing File History" on page 9-16
"Comparing the Working Copy of a File to the Latest Version in Source Control" on page 9-18
"Viewing Source Control Properties of a File" on page 9-20
"Starting the Source Control System" on page 9-21

Getting the Latest Version of Files for Viewing or Compiling

You can get the latest version of a file from the source control system for viewing or running. Getting a file differs from checking it out. When you get a file, it is write protected so you cannot edit it, but when you check out a file, you can edit it.

To get the latest version, follow these steps:

1 In the MATLAB **Current Directory** browser, select the directories or files that you want to get. If you select files, you cannot also select directories.

2 Right-click, and from the context menu, select **Source Control > Get Latest Version**.

The MATLAB **Get latest version** dialog box opens, listing the files or directories you selected.
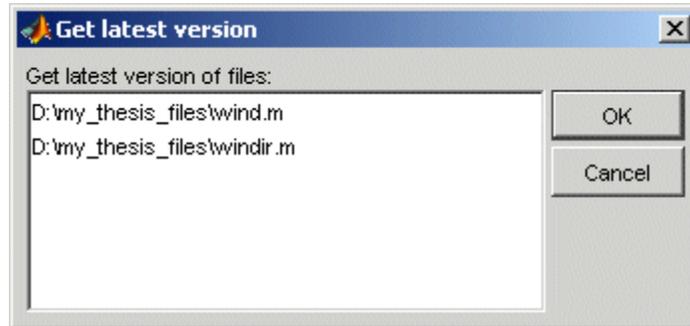
3 Click **OK**.

You can now open the file to view it, run the file, or check out the file for editing.

Function Alternative

The function alternative is `verctrl` with the `get` argument.

Removing Files from the Source Control System

To remove files from the source control system, follow these steps:

1 In the MATLAB **Current Directory** browser, select the files you want to remove.

2 Right-click, and from the context menu, select **Source Control > Remove from Source Control**.

The MATLAB **Remove from source control** dialog box opens, listing the files you selected.

3 Click **OK**.

Function Alternative

The function alternative is `verctrl` with the `remove` argument.

Showing File History

To show the history of a file in the source control system, follow these steps:

1 In the MATLAB **Current Directory** browser, select the file for which you want to view the history.

2 Right-click, and from the context menu, select **Source Control > History**.

A dialog box, which is specific to your source control system, opens. For Microsoft Visual SourceSafe, the **History Options** dialog box opens, as shown in the following example illustration.
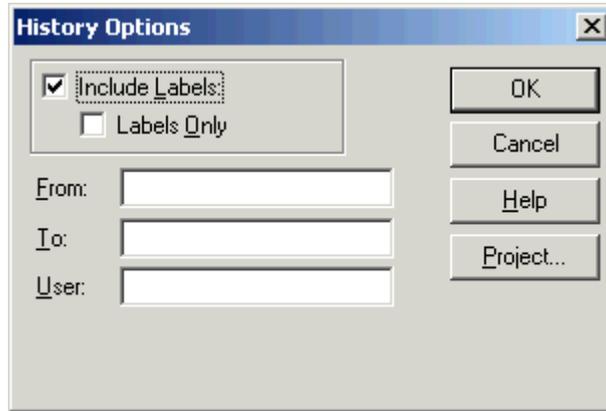
3 Complete the dialog box to specify the range of history you want for the selected file and click **OK**. For example, enter my_name for **User**.

The history presented depends on your source control system. For Microsoft Visual SourceSafe, the **History** dialog box opens for that file, showing the file's history in the source control system.
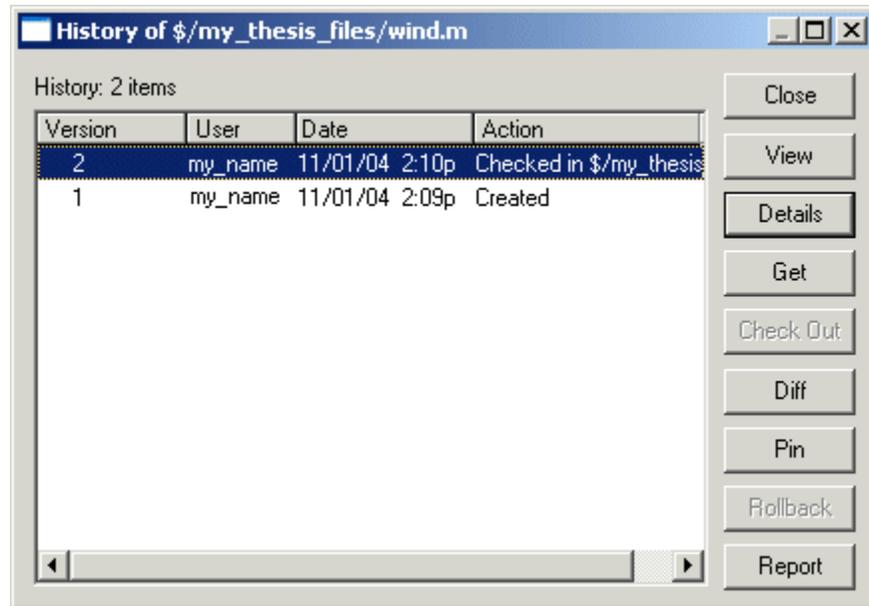
Function Alternative

The function alternative is verctrl with the history argument.

Comparing the Working Copy of a File to the Latest Version in Source Control

You can compare the current working copy of a file with the latest checked-in version of the file in the source control system. This highlights the differences between the two files, showing the changes you made since you checked out the file.

To view the differences, follow these steps:

1 In the MATLAB **Current Directory** browser, select the file for which you want to view differences. This is a file that has been checked out and edited.

2 Right-click, and from the context menu, select **Source Control > Differences**.

A dialog box, which is specific to your source control system, opens. For Microsoft Visual SourceSafe, the **Difference Options** dialog box opens.

3 Review the default entries in the dialog box, make any needed changes, and click **OK**. The following example is for Microsoft Visual Source Safe.
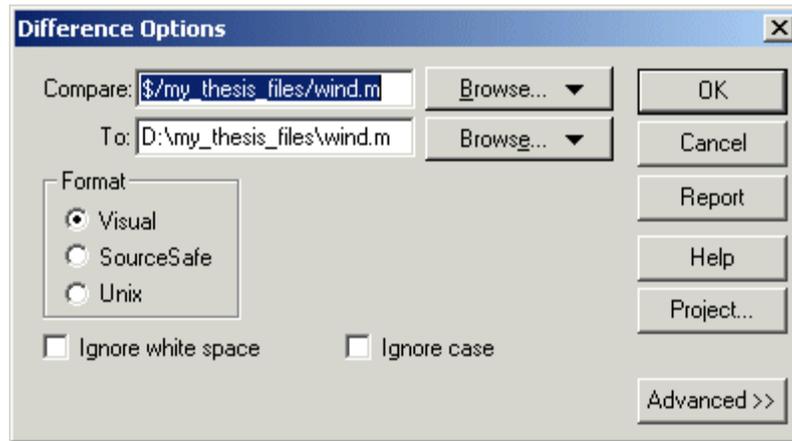
The method of presenting differences depends on your source control system. For Microsoft Visual SourceSafe, the **Differences for** dialog box opens. This highlights the differences between the working copy of the file and the latest checked-in version of the file.
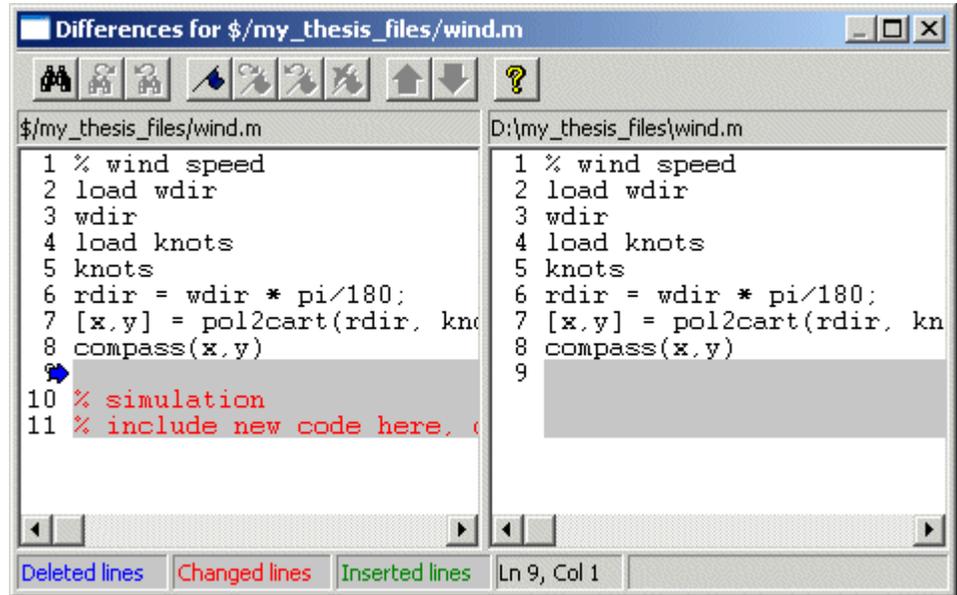
Function Alternative

The function alternative is verctrl with the showdiff or isdiff argument.

Viewing Source Control Properties of a File

To view the source control properties of a file, follow these steps:

1 In the MATLAB **Current Directory** browser, select the file for which you want to view properties.

2 Right-click, and from the context menu, select **Source Control > Properties**.

A dialog box, which is specific to your source control system, opens. The following example shows the Microsoft Visual SourceSafe properties dialog box.
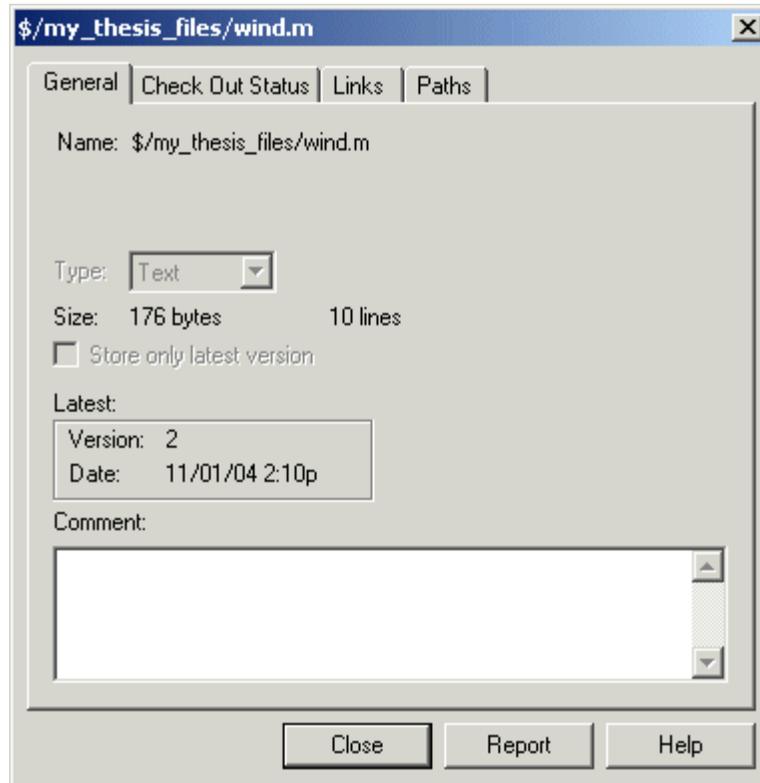
Function Alternative

The function alternative is verctrl with the properties argument.

Starting the Source Control System

All the MATLAB source control actions automatically start the source control system to perform the action, if the source control system is not already open. If you want to start the source control system from MATLAB without performing a specific action source control action,

1 Right-click any directory or file in the MATLAB Current Directory browser

2 From the context menu, select **Source Control > Start Source Control System**.

The interface to your source control system opens, showing the source control project associated with the current directory in MATLAB. The following example shows the Microsoft **Visual SourceSafe Explorer** interface.
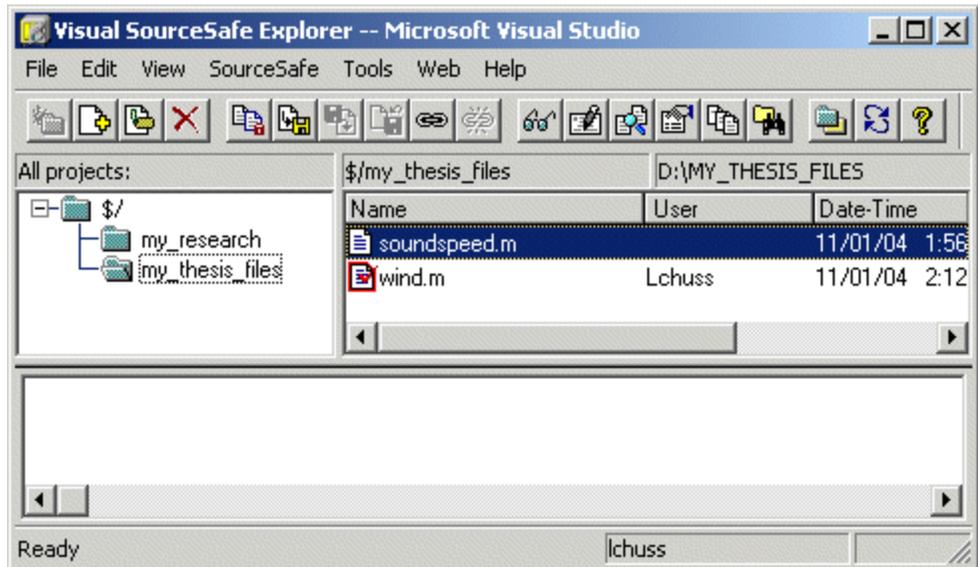
Function Alternative
The function alternative is verctrl with the runscc argument.

Performing Source Control Actions from the Editor/Debugger, Simulink, or Stateflow on Windows

You can create or open a file in the Editor/Debugger, Simulink, or Stateflow and perform most source control actions from their **File > Source Control** menus, rather than from the Current Directory browser as described in previous sections. Following are some differences in the source control interface process when you use the Editor/Debugger, Simulink, or Stateflow:

- You can perform actions on only one file at time.

- Some of the dialog boxes have a different icon in the title bar. For example, the **Check out file(s)** dialog box uses an M-file Editor/Debugger document icon instead of the MATLAB icon.

- You cannot add a new (Untitled) file, but must instead first save the file.

- You cannot register projects from Simulink or Stateflow. Instead, register a project using the Current Directory browser, as described in "Register Source Control Project with MATLAB" on page 9-7.

Troubleshooting Source Control Problems on Windows

In this section...
"Source Control Error: Provider Not Present or Not Installed Properly" on page 9-24
"Restriction Against @ Character" on page 9-25
"Add to Source Control Is the Only Action Available" on page 9-25
"More Solutions for Source Control Problems" on page 9-25

Source Control Error: Provider Not Present or Not Installed Properly

In some cases, MATLAB recognizes your source control system but you cannot use source control features for MATLAB. Specifically, when you select **File > Preferences > General > Source Control**, or run cmopts, MATLAB lists your source control system, but you cannot perform any source control actions. Only the **File > Source Control > Start Source Control System** menu item is available, and when you select it, MATLAB displays this error:

```
Source control provider is not present or not installed properly.
```

Often, this error occurs because a registry key that MATLAB requires from the source control application is not present. Make sure this registry key is present:

```
HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\
InstalledSCCProviders
```

The registry key refers to another registry key that is similar to

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SourceSafe\SccServerPath
```

This registry key has a path to a DLL-file in the file system. Make sure the DLL-file exists in that location. If you are not familiar with registry keys, ask your system administrator for help.

If this does not solve the problem and you use Microsoft Visual SourceSafe, try running a client setup for your source control application. When SourceSafe is

installed on a server for a group to use, each machine client can run a setup but is not required to do so. However, some applications that interface with SourceSafe, including MATLAB, require you to run the client setup. Run the client setup, which should resolve the problem.

If the problem persists, access source control outside of MATLAB.

Restriction Against @ Character

Some source control systems, such as Perforce and Synergy, reserve the @ character. Perforce, for example, uses it as a revision specifier. Therefore, you might experience problems if you use these source control systems with MATLAB files and directories that include the @ character in the directory or filename.

You might be able to work around this restriction by quoting nonstandard characters in filenames, such as with an escape sequence, which some source control systems allow. Consult your source control system documentation or technical support resources for a workaround.

Add to Source Control Is the Only Action Available

To use source control features for a file in Simulink or Stateflow, the file's source control project must first be registered with MATLAB. When a file's source control project is *not* registered with MATLAB, all **File > Source Control** menu items are disabled except **Add to Source Control**. You can select **Add to Source**, which registers the project with MATLAB, or you can register the project using the Current Directory browser, as described in "Register Source Control Project with MATLAB" on page 9-7. You can then perform source control actions for all files in that project (directory).

More Solutions for Source Control Problems

The latest solutions for problems interfacing MATLAB with a source control system appear on the MathWorks Web page for support at `http://www.mathworks.com/support/`. Search Solutions and Technical Notes for "source control."

Source Control Interface on UNIX

If you use a source control system to manage your files, you can check M-files and Simulink models, and Stateflow charts into and out of the source control system from within MATLAB, Simulink, and Stateflow.

The source control interface supports four popular source control systems, as well as a custom option:

- ClearCase from IBM Rational

- Concurrent Version System (CVS)

- ChangeMan from Serena (also used for PVCS from Merant)

- Revision Control System (RCS)

- Custom option — Allows you to build your own interface if you use a different source control system. For details, see the reference page for customverctrl.

Perform source control interface actions for a single file using menu items in the MATLAB Editor/Debugger, a Simulink model window, or a Stateflow chart window. To perform source control actions on multiple files, use the Current Directory browser. Alternatively, run source control functions in the Command Window, which provide some options not supported with the menu items.

Specifying the Source Control System on UNIX

MATLAB Alternative

In MATLAB, specify the source control system you want to access. Select **File > Preferences > General > Source Control**.

The currently selected system is shown in the Preferences dialog box. The default selection is None.

Select the source control system you want to interface with and click **OK**.
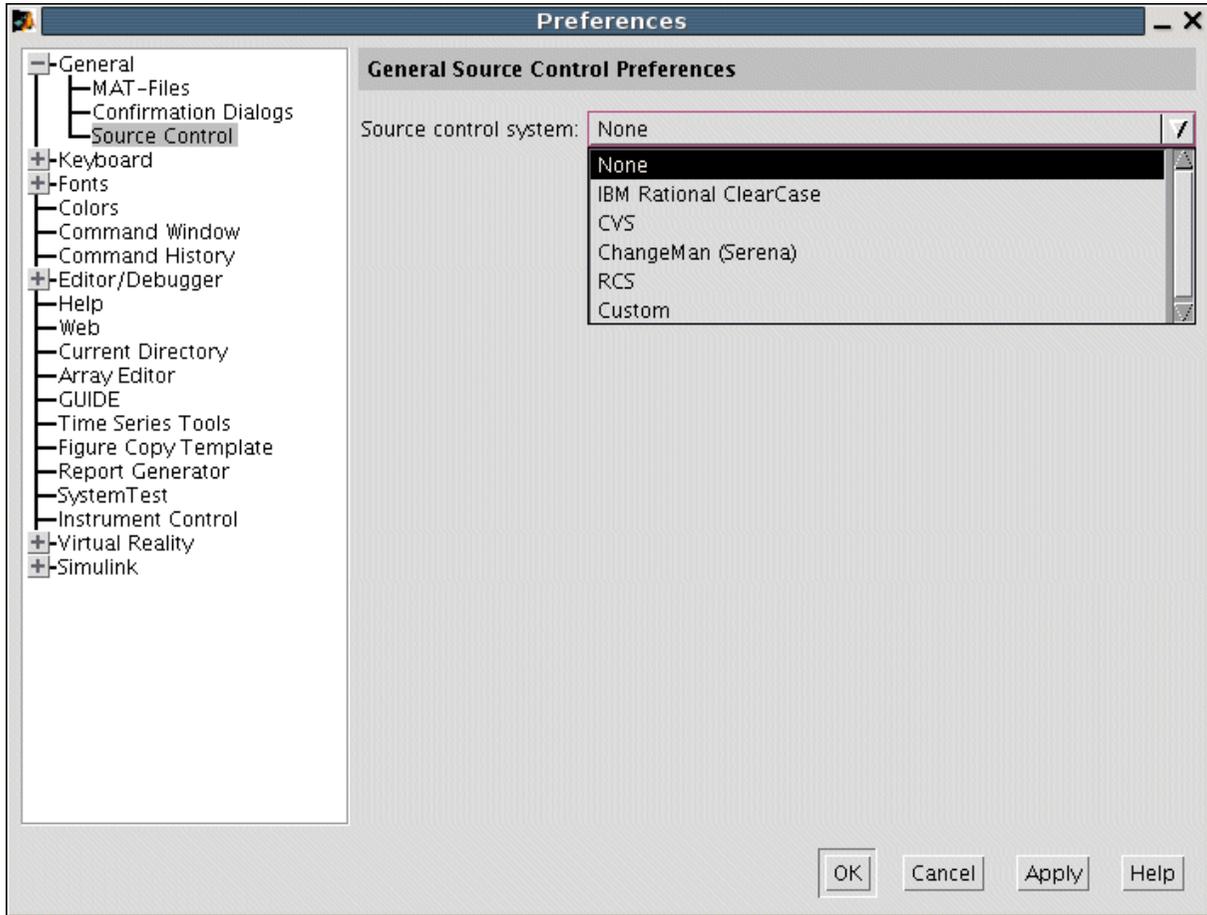
MATLAB remembers preferences between sessions, so you only need to perform this action when you want to access a different source control system.

Function Alternative

A function alternative to select a source control system is not available, but you can list the currently selected source control system by running cmopts.

Setting a View and Checking Out a Directory with ClearCase on UNIX

If you use ClearCase on a UNIX platform, perform the following from ClearCase:

1 Set a view.

2 Check out the directory that contains files you want to save, check in, or check out.

You can now use the MATLAB, Simulink, or Stateflow source control interfaces to ClearCase.

Checking Files Into the Source Control System on UNIX

In this section...

Checking In One or More Files Using the Current Directory Browser

1 From the Current Directory browser, select the file or files to check in. A file can be open or closed when you check it in, but it must be saved, that is, it cannot contain unsaved changes.

2 Right-click, and from the context menu, select **Source Control > Check In**.

3 In the resulting **Check in file(s)** dialog box, you can add text in the **Comments** field. If you want to continue working on the files, select the check box **Keep checked out**. Click **OK**.
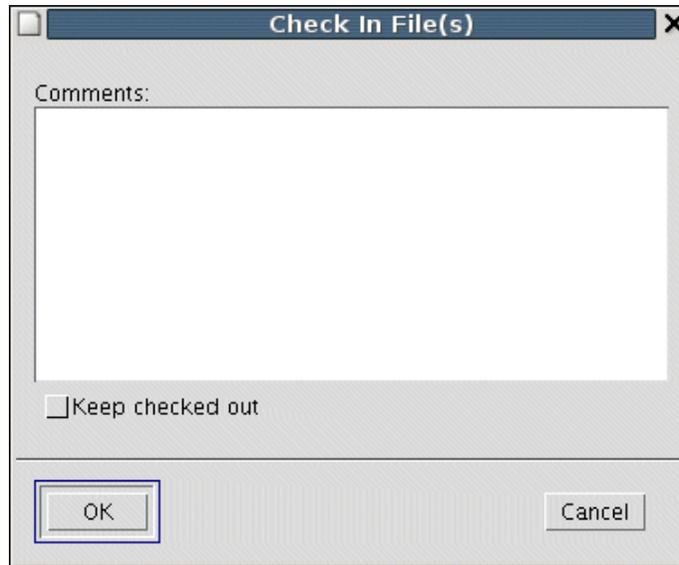
The files are checked into the source control system. If any file contains unsaved changes when you try to check it in, you will be prompted to and must then save the changes to complete the checkin.

An error appears in the Command Window if a file is already checked in.

If you did not keep a file checked out and you keep that file open, note that it is a read-only version.

Checking In One File Using the Editor/Debugger, Simulink, or Stateflow

1 From the Editor/Debugger, Simulink, or Stateflow, with the file open and saved, select **File > Source Control > Check In**.

2 In the resulting **Check in file(s)** dialog box, you can add text in the **Comments** field. If you want to continue working on the files, select the check box **Keep checked out**. Click **OK**.

Function Alternative

Use `checkin` to check files into the source control system. The files can be open or closed when you use `checkin`. The `checkin` function takes this form:

```
checkin({'file1','file2', ...},'comments','comment_text',...
'option','value')
```

For `filen`, use the complete path and include the file extension. You must supply the `comments` argument and a comments string with `checkin`.

Use the `option` argument to

- Check in a file and keep it checked out — set the `lock` option value to `on`.

- Check in a file even though it has not changed since the previous check in — set the `force` option value to `on`.

The `comments` argument and the `lock` and `force` options apply to all files checked in.

Example Using checkin Function

To check in the file `clock.m` with the comment `Adjustment for leap year`, type

```
checkin('\myserver\mymfiles\clock.m','comments', ...
'Adjustment for leap year')
```

For other examples, see the reference page for `checkin`.

Checking Files Out of the Source Control System on UNIX

Checking Out One or More Files Using the Current Directory Browser

1 In the Current Directory browser, select the file or files to check out.

2 Right-click, and from the context menu, select **Source Control > Check Out**. The **Check out file(s)** dialog box opens.
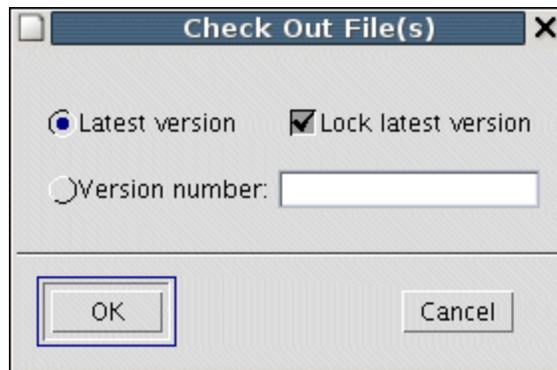
3 Complete the dialog box:

   a To check out the versions that were most recently checked in, select
   the **Latest version** option.

   b To check out a specific version of the files, select the **Version number**
   option and type the version number in the field.

     c To prevent others from checking out the files while you have them checked out, select **Lock latest version**. To check out read-only versions of the file, clear **Lock latest version**.

4 Click **OK**.

An error appears in the Command Window if a file is already checked out.

After checking out files, make changes to them in MATLAB or another product, and save the files. For example, edit an M-file in the Editor/Debugger.

If you try to change a file without first having checked it out, the file is read-only, as seen in the title bar, and you will not be able to save any changes. This protects you from accidentally overwriting the source control version of the file.

If you end the MATLAB session, the file or files remain checked out. You can check in files from within MATLAB during a later session, or directly from your source control system.

Checking Out a Single File Using the Editor/Debugger, Simulink, or Stateflow

1 Open the M-file, Simulink model, or Stateflow chart you want to check out. The title bar indicates the file is read-only.

2 Select **File > Source Control > Check Out**. The **Check out file(s)** dialog box opens.

3 Complete the dialog box as described in step of "Checking Out One or More Files Using the Current Directory Browser" on page 9-33, and click **OK**.

Function Alternative

Use checkout to check out a file from the source control system. You can check out multiple files at once and specify checkout options. The checkout function takes this form:

```
checkout({'file1','file2', ...},'option','value')
```

For `filen`, use the complete path and include the file extension.

Use the `option` argument to

- Check out a read-only version of the file — set the `lock` option value to `off`.
- Check out the file even if you already have it checked out — set the `force` option value to `on`.
- Check out a specific version of the file — use the `revision` option, and assign the version number to the `value` argument.

The options apply to all files being checked out. The files can be open or closed when you use `checkout`.

Example Using checkout Function—Check Out a Specific Version of a File

To check out the 1.1 version of the file `clock.m`, type

```
checkout('\myserver\mymfiles\clock.m','revision','1.1')
```

For other examples, see the reference page for `checkout`.

Undoing the Checkout on UNIX

In this section...
"Impact of Undoing a File Checkout" on page 9-36
"Undoing the Checkout for One or More Files Using the Current Directory Browser" on page 9-36
"Undoing the Checkout for a Single File Using the Editor/Debugger, Simulink, or Stateflow" on page 9-36
"Function Alternative" on page 9-37

Impact of Undoing a File Checkout

When you undo the checkout for a file, the file remains checked in, and does not have any of the changes you made since you checked it out. To save any changes you have made since checking out a file, select **File > Save As**, and supply a different filename before you undo the checkout. Undo the checkout using the Current Directory browser for one or more files. For only one file, you can also use the Editor/Debugger, Simulink, or Stateflow.

Undoing the Checkout for One or More Files Using the Current Directory Browser

1 In the MATLAB **Current Directory** browser, select the file or files for which you want to undo the checkout.

2 Right-click, and from the context menu, select **Source Control > Undo Checkout**. MATLAB undoes the checkout.

An error appears in the Command Window if the file is not checked out.

Undoing the Checkout for a Single File Using the Editor/Debugger, Simulink, or Stateflow

1 Open the M-file, Simulink model, or Stateflow chart for which you want to undo the checkout.

2 Select **File > Source Control > Undo Checkout**. MATLAB undoes the checkout.

Function Alternative

The undocheckout function takes this form:

```
undocheckout({'file1','file2', ...})
```

Use the complete path for filen and include the file extension. For example, to undo the checkout for the files clock.m and calendar.m, type

```
undocheckout({'\myserver\mymfiles\clock.m',...
'\myserver\mymfiles\calendar.m'})
```

Index

F