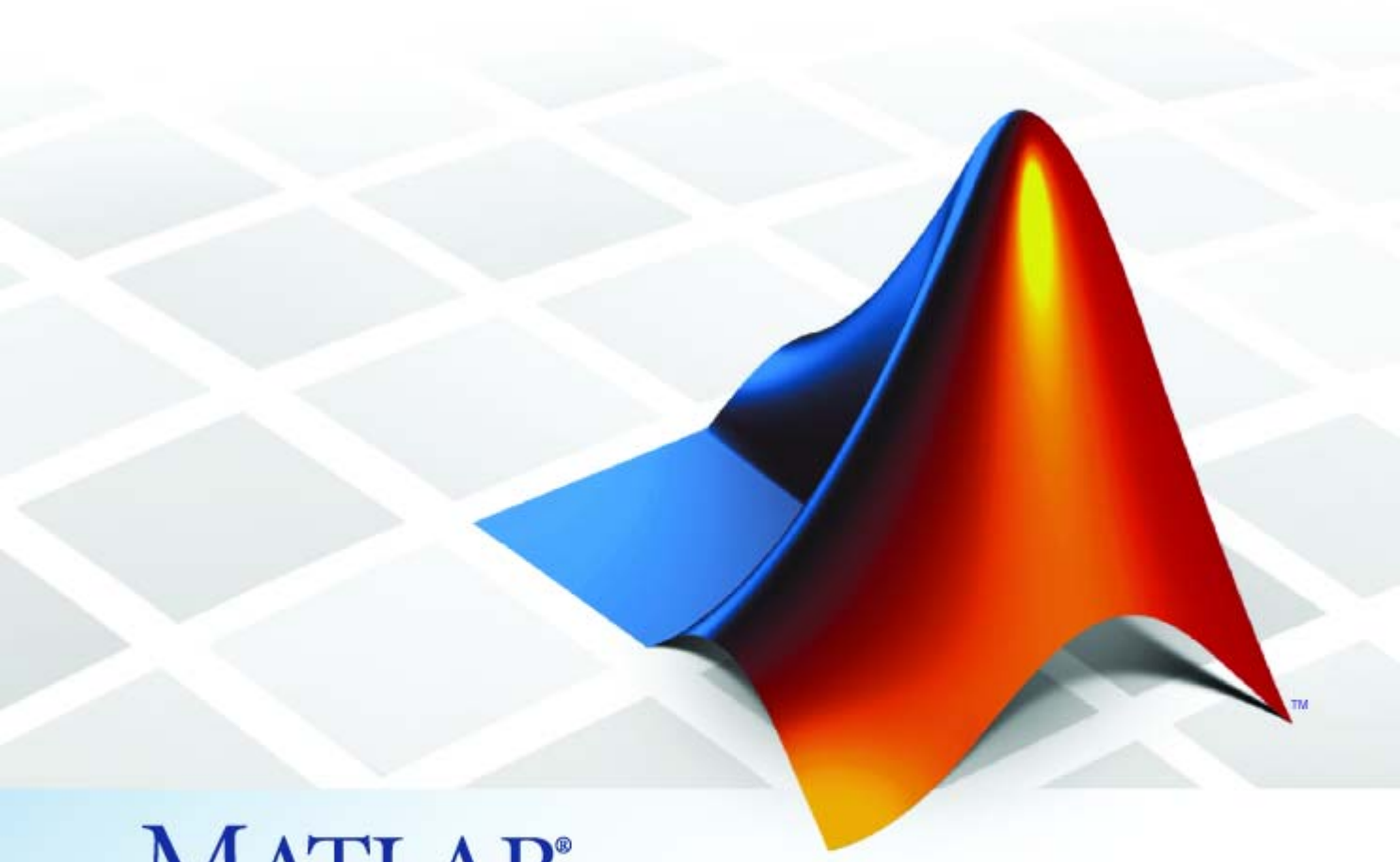
MATLAB® 7
Desktop Tools and Development Environment



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 and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be 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

June 2004	First printing	New for MATLAB 7.0 (Release 14). Formerly part of Using MATLAB.
October 2004	Online only	Revised for MATLAB 7.0.1 (Release 14SP1)
March 2005	Online only	Revised for MATLAB 7.0.4 (Release 14SP2)
March 2005	Second printing	Revised for MATLAB 7.0.4 (Release 14SP2)
June 2005	Third printing	Minor revision for MATLAB 7.0.4 (Release 14SP2)
September 2005	Online only	Revised for MATLAB 7.1 (Release 14SP3)
March 2006	Online only	Revised for MATLAB 7.2 (Release 2006a)
September 2006	Online only	Revised for MATLAB 7.3 (Release 2006b)
March 2007	Online only	Revised for MATLAB 7.4 (Release 2007a)
September 2007	Online only	Revised for MATLAB 7.5 (Release 2007b)
March 2008	Online only	Revised for MATLAB 7.6 (Release 2008a)
October 2008	Online only	Revised for MATLAB 7.7 (Release 2008b)
March 2009	Online only	Revised for MATLAB 7.8 (Release 2009a)
September 2009	Online only	Revised for MATLAB 7.8 (Release 2009b)

Contents

1 Startup and Shutdown

2 Desktop

Running Functions — Command Window and History

3

4 Help, Demos, and Related Resources

5 Workspace Browser and Variable Editor

6 Managing Files in MATLAB

7 File Exchange — Finding and Getting Files Created by Other Users

8 Editing and Debugging M-Files

9 Tuning and Managing M-Files

10 Publishing M-Files

11 Using Notebook to Publish to Microsoft Word

12 Source Control Interface

13 Internationalization

Index

1

Startup and Shutdown

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

Overview of Starting the MATLAB Program

The way you start the MATLAB program depends on the platform you use:

- On Microsoft® Windows® platforms, start MATLAB by double-clicking the MATLAB shortcut  on your Windows desktop.
- On Apple® Macintosh® platforms, start MATLAB by double-clicking the MATLAB icon in the `Applications` folder.
- On platforms that run the UNIX®[1] operating system, start MATLAB by typing `matlab` at the operating system prompt.

When you start MATLAB, all the product software from The MathWorks™ that you are licensed to use, is available. You do not have to start each product separately.

There are alternative ways to start MATLAB, and you can customize startup. For example, you can change the folder in which MATLAB starts or automatically execute MATLAB statements upon startup.

Starting the MATLAB Program on Windows Platforms

In this section...
"Starting the MATLAB Program from the Windows Desktop or a DOS Window" on page 1-3
"Starting the MATLAB Program using File Associations on Windows Platforms" on page 1-3
"Utility to Change File Associations for Windows Platforms" on page 1-6
"Changing File Associations for the MATLAB Program from the Windows Environment" on page 1-6

1. UNIX is a registered trademark of The Open Group in the United States and other countries.

Starting the MATLAB Program from the Windows Desktop or a DOS Window

To start the MATLAB program on a Microsoft Windows platform, select **Start > Programs > MATLAB > R2009b > MATLAB R2009b**, or double-click the MATLAB shortcut  on your Windows desktop. The installer created this shortcut 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 folder in which you want to start MATLAB and type `matlab` at the DOS prompt.

You can specify the current folder upon startup as well as other options—for more information, see "Startup Folder for the MATLAB Program" on page 1-11 and "Startup Options" on page 1-17.

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 the MATLAB Program using File Associations on Windows Platforms

On Windows platforms, you can start MATLAB from the Windows Explorer tool 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 file type associations for the Windows operating system. When you installed MATLAB for Windows platforms, you specified the file types to associate with MATLAB. For example, if you accepted the default options, double-clicking an M-file in the Windows Explorer tool opens the file in the MATLAB Editor.

The default option also associates MEX-files and P-files with MATLAB in the Windows Explorer tool, which assigns the file types an icon for MATLAB. However, double-clicking a file with a `.mex` (`.mexw32` or `.mexw64`), or `.p` extension does not run or open the file in MATLAB.

File Extension and Resulting Action

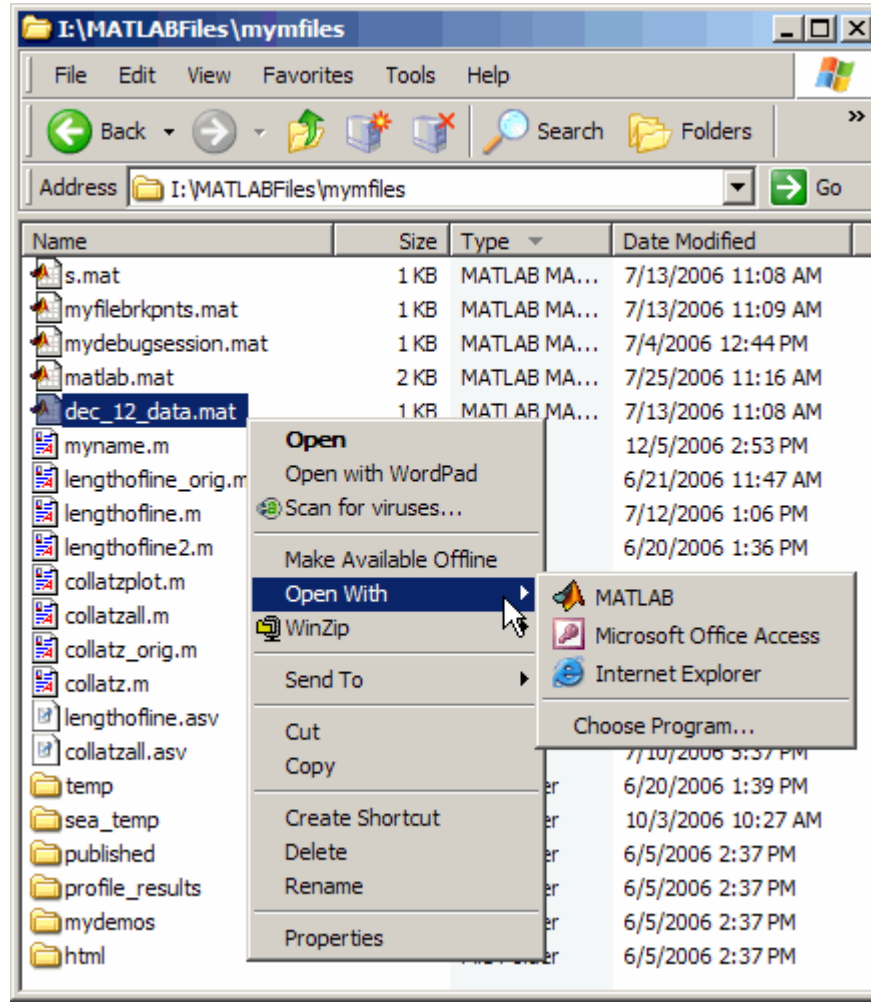
File Extension	Result
`.fig`	Opens file in figure window
`.m`	Opens file in Editor
`.mat`	Opens Import Wizard to load the data into the MATLAB workspace
`.mdl`	Opens file in a Simulink® model window
`.mex`	Displays icon for MATLAB in Windows Explorer tool
`.p`	Displays icon for MATLAB in Windows Explorer tool

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

If you double-click a file with one of these file extensions and it does not open in MATLAB, try this:

1. In the Windows Explorer tool, right-click a file that has one of the extensions for MATLAB, 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 File Associations for Windows Platforms" on page 1-6
   - "Changing File Associations for the MATLAB Program from the Windows Environment" on page 1-6

   After associating a file type with MATLAB, you can open other applications that have the same extension via the context menu. For example, if you want to open a MAT-file with the Access application, right-click `myfile.mat`, and from the context menu, select **Open With**. The Access application should be one of the options.



File associations for the Windows Explorer tool 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 the Access application, when you open a MAT-file from within MATLAB, it opens the Import Wizard to load the data.

Utility to Change File Associations for Windows Platforms

If you are viewing this topic in the MATLAB Help browser, you can run one of the utilities provided here to create associations in the Windows environment for common file types used by MATLAB. 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 files with .fig extension with MATLAB
- Run utility to associate files with .m extension with MATLAB
- Run utility to associate files with .mat extension with MATLAB
- Run utility to associate files with .mdl extension with MATLAB
- 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 the Windows Explorer tool might not reflect the change immediately.

Changing File Associations for the MATLAB Program from the Windows Environment

You can associate file types with MATLAB from the Windows Explorer tool. This is useful if you want associate file types other than those you can change with the “Utility to Change File Associations for Windows Platforms” on page 1-6. For example, you can associate the `.xml` extension with MATLAB so that when you double-click an XML file, it opens in the MATLAB Editor.

The following examples show one way to change file associations in the Windows Explorer tool. Note that these instructions might not exactly apply to your version of the Windows operating system. 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 the Windows Explorer tool, it opens in the Microsoft Access application, but you want the file to open in MATLAB.

1. In the Windows Explorer tool, 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 the list does not include `MATLAB`, 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\R2009b\bin`.
4. In the Open With dialog box, click **OK**. In the Folder Options dialog box, click **Close**.

Starting the MATLAB Program on UNIX Platforms

To start the MATLAB program on UNIX[2] platforms, 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 the folder in which you installed MATLAB. (For more information, see the `matlabroot` reference page). If you have trouble starting MATLAB, see troubleshooting information in the *Installation Guide for UNIX*.

2. UNIX is a registered trademark of The Open Group in the United States and other countries.

You can specify the current folder upon startup as well as other options—for more information, see “Startup Folder for the MATLAB Program” on page 1-11 and “Startup Options” on page 1-17.

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 the MATLAB Program on Macintosh Platforms

In this section...

Starting the MATLAB Program from the Macintosh Desktop

To start the MATLAB program on Apple Macintosh platforms, double-click the `MATLAB_R2009b` icon in the `Applications` folder.

You can specify the current folder upon startup as well as other options—for more information, see "Startup Folder for the MATLAB Program" on page 1-11 and "Startup Options" on page 1-17.

If MATLAB fails to start due to a problem with required system components such as X11 or Sun Microsystems™ Java™ software, diagnostics run automatically and advise you of the problem, along with suggestions to correct it.

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.

Limitation

On Macintosh platforms, if you run MATLAB remotely, for example using `rlogin`, you must run with `nodisplay`, `noawt`, and `nojvm` startup options—for more information, see "Startup Options" on page 1-17.

Starting the MATLAB Program from a Shell on Macintosh Platforms

You can start MATLAB on Macintosh platforms the way you would start it on other UNIX[3] platforms. For example, you would run

```
/Applications/MATLAB_R2009b.app/bin/matlab
```

For more information, see "Starting the MATLAB Program on UNIX Platforms" on page 1-7.

3. UNIX is a registered trademark of The Open Group in the United States and other countries.

Startup Folder for the MATLAB Program

In this section...

What Is the Startup Folder?

The *startup folder* is the current folder in the MATLAB application when it starts. It is convenient if you make the current folder upon startup be a folder that you frequently use. On Windows and Apple Macintosh platforms, a folder called `userpath` is added automatically to the search path upon startup, and is the default startup folder. On UNIX[4] platforms, you can set the `userpath` as the startup folder. The default value for `userpath` is, for example, `Documents/MATLAB` on Microsoft Windows Vista™ platforms. You can specify a different default value for `userpath`, or specify a different startup folder.

Accepting the default value for `userpath` and using it as the startup folder offers these benefits:

- You can store the MATLAB files you work with in one, appropriately-named location, such as `Documents/MATLAB`.
- Your MATLAB files are readily available upon startup, because the current folder is always the same, for example, `Documents/MATLAB`.
- You can always run your files because MATLAB automatically adds the `userpath` folder to the top of the search path upon startup.
- The first time you run a new version of MATLAB, MATLAB automatically creates the `userpath` folder if it does not exist.

4. UNIX is a registered trademark of The Open Group in the United States and other countries.

- When you upgrade to a newer version of MATLAB, MATLAB automatically continues to use the same `MATLAB` folder and your existing files, with all of its other benefits.
- The default `userpath` also utilizes the benefits provided by the standard location in the Windows and Macintosh environments for storing personal files. Files in the `Documents/MATLAB` folder (or `My Documents/MATLAB` on Windows platforms other than Windows Vista) are available to you when you use other machines. Because each user has their own `Documents/MATLAB` folder, other users, even those using your machine, cannot access files in your `Documents/MATLAB` folder.

To view the `userpath` value, run the `userpath` function. To specify a location other than the default for `userpath`, or if you do not want to take advantage of `userpath`, make changes with the `userpath` function.

There are other ways to change the startup folder as well as the folders on your search path. For more information, see "Changing the Startup Folder" on page 1-14 and "Ways to View and Change the Search Path" on page 6-47.

Startup Folder on Windows Platforms

The startup folder on Windows platforms depends on any startup options you specified and the way you started MATLAB:

Startup Folder When Starting the MATLAB Program from a Windows Shortcut

When you start the MATLAB program from a Windows shortcut, by default MATLAB sets the startup folder to the `userpath` value, whose default value is `My Documents\MATLAB`, or `Documents\MATLAB` on Windows Vista platforms. The `userpath` folder is automatically added to the search path.

If there is a value specified in the **Start in** field of the Properties dialog box for the MATLAB program, that value is the startup folder, although the `userpath` is added to the search path. If MATLAB does not find a valid `userpath` value upon startup, and the **Start in** field is empty, the startup folder is the Windows desktop.

Startup Folder When Starting the MATLAB Program from an Associated File

When you start MATLAB by double-clicking a file type associated with MATLAB, that file's folder is the startup folder. The `userpath` folder is automatically added to the search path.

Startup Folder When Starting the MATLAB Program from a DOS Window

When you start MATLAB from a DOS window, the startup folder is the folder in which you ran the `matlab` function. The `userpath` folder is automatically added to the search path.

Startup Folder on UNIX Platforms

On UNIX platforms, the default startup folder is the folder from which you started MATLAB. You can specify that the `userpath` be the startup folder by setting the value of the environment variable `MATLAB_USE_USERPATH` to 1 prior to startup. By default, `userpath` is *`userhome`*`/Documents/MATLAB`, and MATLAB automatically adds the `userpath` folder to the top of the search path upon startup. To specify a different folder for `userpath`, and for other options, use the `userpath` function.

Startup Folder on Macintosh Platforms

When you start MATLAB on Apple Macintosh platforms by double-clicking the MATLAB application, the startup folder is the value returned when you enter `userpath`, which by default is *`userhome`*`/Documents/MATLAB`. MATLAB automatically adds the `userpath` folder to the top of its search path upon startup. To specify a different folder for `userpath`, and for other options, use the `userpath` function.

When you start MATLAB in a shell, the startup folder is the same as for other UNIX platforms—see “Startup Folder on UNIX Platforms” on page 1-13.

Changing the Startup Folder

You can start MATLAB in a folder other than the default in one of these ways:

Changing the Startup Folder Via the userpath Function

Use the `userpath` function to change the startup folder as well as to add the startup folder to the search path upon startup. For more information, see the `userpath` reference page and “Startup Folder for the MATLAB Program” on page 1-11.

Changing the Startup Folder Using the Shortcut — Windows Platforms Only

To change the startup folder on Windows platforms using the shortcut,

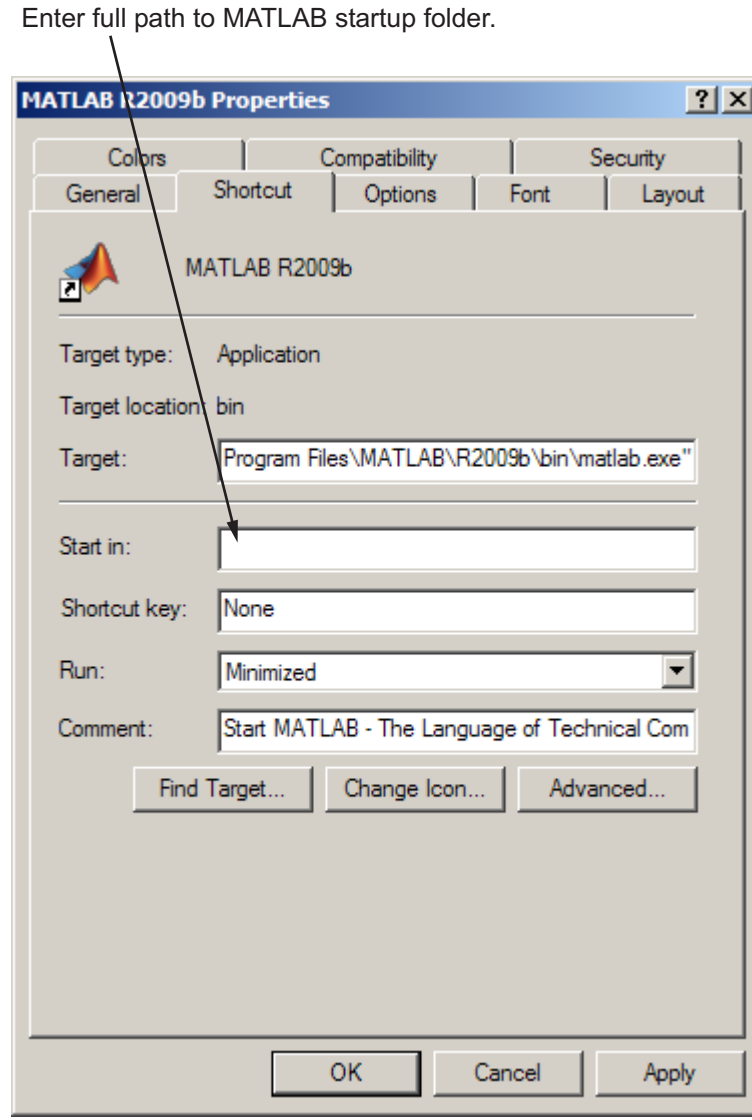
1. Right-click the shortcut icon for MATLAB and select **Properties** from the context menu.

   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 folder is `My Documents\MATLAB` or `Documents\MATLAB` on Windows Vista platforms; for more information, see “Startup Folder on Windows Platforms” on page 1-12.

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



The next time you start MATLAB using that shortcut icon, the current folder will be the one you specified in step 2.

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

Changing the Startup Folder Using the startup.m File

Use the `startup.m` file to specify the startup folder as well as other startup options—for details, see "Specifying Startup Options Using the Startup File for the MATLAB Program, startup.m" on page 1-19.

Startup Options

In this section...

About Startup Options

You can define startup options that instruct the MATLAB program to perform certain operations when you start it. On MicrosoftWindows platforms, you can use a GUI to specify the options. On all platforms, you can specify these options using a startup file (`startup.m`), or in conjunction with the `matlab` startup function.

Specifying 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 shortcut in the Windows environment for MATLAB. Or you can add them to the command line when you start MATLAB in a DOS window. For more information about the options, see "Commonly Used Startup Options" on page 1-20.

On Windows platforms, 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 users on Windows platforms.

Startup Options for a Shortcut in Windows Environment

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

1. Right-click the shortcut icon for MATLAB 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 folder or on the search path for MATLAB. The statement in the **Target** field might appear as

```
C:\Program Files\MATLAB\R2009b\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 a 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 folder, after starting MATLAB in a DOS window:

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

Specifying Startup Options for UNIX Platforms

Include startup options (also called command flags or command line switches) after the `matlab` command when you start MATLAB on UNIX[5] platforms. For more information about the options, see "Commonly Used Startup Options" on page 1-20. On UNIX platforms, a startup option is preceded by a hyphen (-). For example, to start MATLAB without the splash screen, type

```
matlab -nosplash
```

See also the `userpath` function.

Specifying Startup Options for Macintosh Platforms

On Apple Macintosh platforms, specify startup options for Macintosh platforms with the `matlab` command, as described at "Specifying Startup Options for UNIX Platforms" on page 1-19.

Specifying Startup Options Using the Startup File for the MATLAB Program, startup.m

At startup, MATLAB automatically executes the file `matlabrc.m` and, if it exists, `startup.m`. The file `matlabrc.m`, which is in the *`matlabroot`*`/toolbox/local` folder, 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 defaults for Handle Graphics® objects. Use the following statements in a `startup.m` file to add the specified folder, `/home/`*`username`*`/mytools`, to the search path, and to change the current folder to `mytools` upon startup.

```
addpath /home/username/mytools
cd /home/username/mytools
```

5. UNIX is a registered trademark of The Open Group in the United States and other countries.

Location of startup.m

Place the `startup.m` file in the default or current startup folder, which is where MATLAB first looks for it. For more information, see "Startup Folder for the MATLAB Program" on page 1-11.

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.
Windows platforms	`-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 platforms	`-nojvm`	Start MATLAB without loading the Sun Microsystems JVM™ software. This minimizes memory usage and improves initial startup speed, but restricts functionality. With `nojvm`, you cannot use the desktop, figures, or any tools that require Java software. 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.
All	`-nosplash`	Start MATLAB without displaying its splash screen.

Platform	Option	Description
All	`-r "statement"`	Automatically run the specified statement immediately after MATLAB starts. This is sometimes referred to as calling MATLAB in batch mode. Files you run must be in the startup folder for MATLAB or on the search path. Do not include pathnames or file extensions. Enclose the statement in double quotation marks (`"statement"`). Use semicolons or commas to separate multiple statements
All	`-singleCompThread`	Limit MATLAB to a single computational thread. By default, MATLAB makes use of the multithreading capabilities of the computer. For more information about multithreading, see “MATLAB Multiprocessing”.

Toolbox Path Caching in the MATLAB Program

In this section...

About Toolbox Path Caching in the MATLAB Program

For performance reasons, the MATLAB program caches toolbox folder 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 folder cache. Because of the cache file, startup is faster, especially if you run MATLAB from a network server or if you have many toolbox folders. 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 foldersories, 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` folders. These folders are for files provided with MathWorks products that should not change except for product installations and updates. Caching those folders provides better performance during a session because MATLAB does not actively monitor those folders.

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

When to Update the Cache

When you add files to *matlabroot*/toolbox folders, 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 installer for MATLAB. MATLAB also updates the cache and cache file automatically when you use MATLAB tools, such as when you save files from the MATLAB Editor to *matlabroot*/toolbox folders.

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

- Save new files in *matlabroot*/toolbox folders using an external editor
- Use operating system features and commands to add or remove files in *matlabroot*/toolbox folders

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 folders.

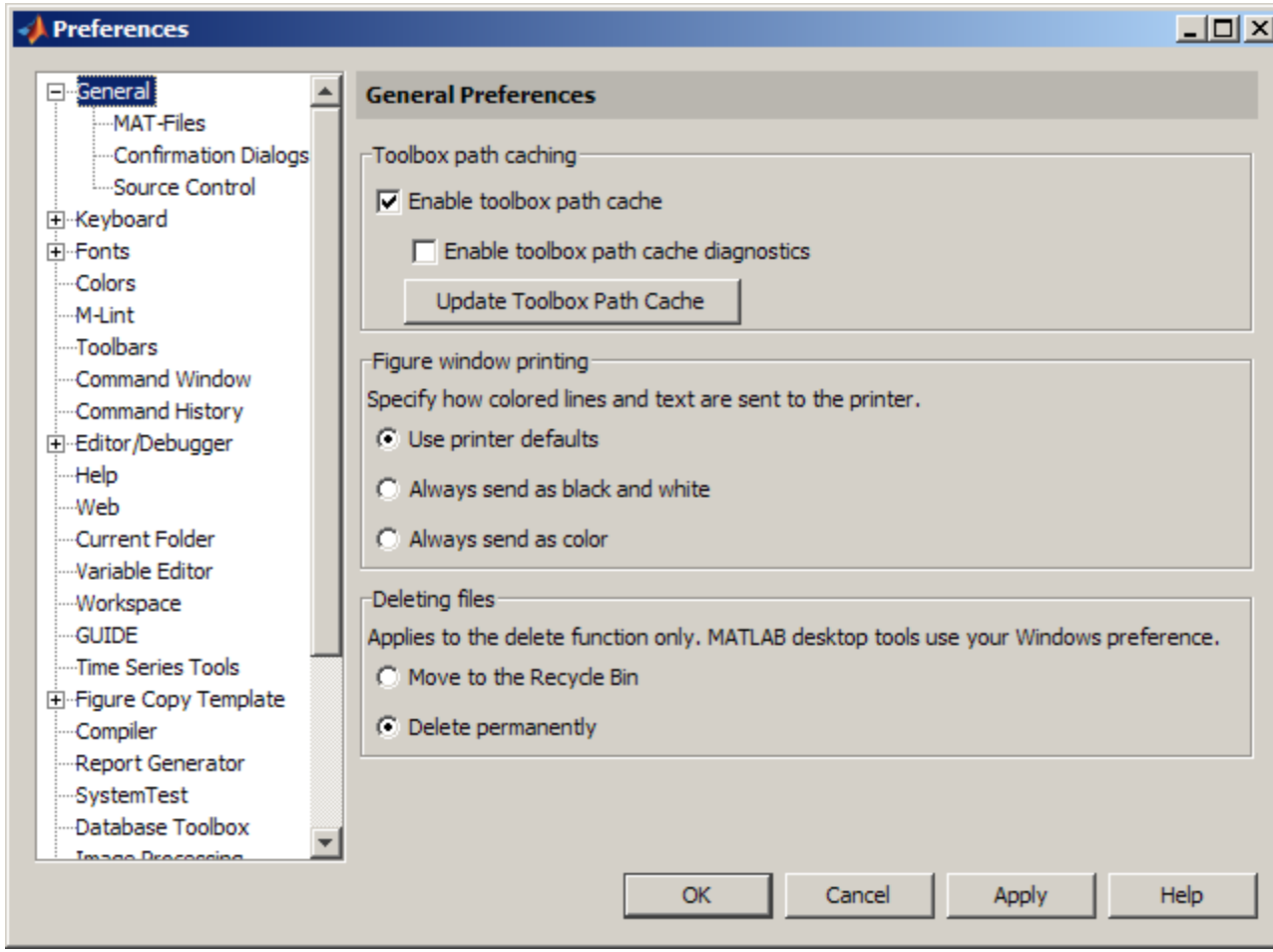
Steps to Update the Cache

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

Upon startup, if the MATLAB program 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 1-29.

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 Software from the MATLAB Program

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

Quitting the MATLAB Program

In this section...

Ways to Quit the MATLAB Program

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

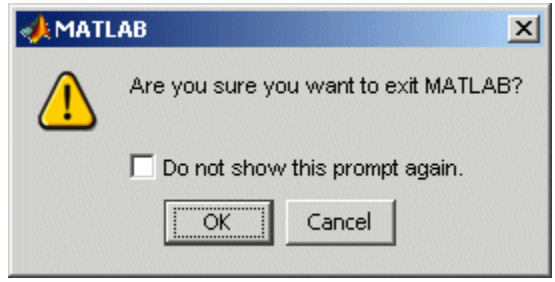
- Click the Close box in the MATLAB desktop.
- Select **Exit MATLAB** from the desktop **File** menu.
- Type `quit` at the Command Window prompt.

MATLAB closes after

- Prompting you to confirm quitting, if that preference is specified (see "Confirm Quitting the MATLAB Program" on page 1-28)
- Prompting you to save any unsaved files
- Running the `finish.m` script, if it exists in the current folder or on the search path (see "Running a Script When Quitting the MATLAB Program" on page 1-29)

Confirm Quitting the MATLAB Program

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.



For more information, see “Confirmation Dialogs Preferences” on page 2-133.

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 the MATLAB Program

When MATLAB quits, it runs the script `finish.m`, if `finish.m` exists in the current folder or anywhere on the 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

- “When the MATLAB Program Terminates Unexpectedly” on page 1-30
- “Error Log Reporting” on page 1-31
- “Recovering Data After an Abnormal Termination” on page 1-31

When the MATLAB Program Terminates Unexpectedly

In the event MATLAB experiences a segmentation violation (segv) or other serious problem, the MATLAB System Error dialog box opens to notify you about the problem. When this occurs, the internal state of MATLAB is unreliable and not suitable for further use. You should exit as soon as possible and then restart. However, you might want to first try to save your work in progress.

To exit and restart without trying to save your work, follow these steps:

1. If you want to view the stack trace for the problem, click **Details**.
2. Click **Close** to terminate MATLAB.
3. Restart MATLAB. If the Error Log Reporter dialog box opens, send a report to The MathWorks.

To try to save your work in progress before exiting and restarting MATLAB, follow these steps:

1. If you want to view the stack trace for the problem, click **Details**.
2. Click **Attempt to Continue**. MATLAB tries to return to the Command Window or tool you were using.

   The Command Window displays the message `Please exit and restart MATLAB` to the left of the prompt, which reminds you to discontinue use.
3. From the Command Window or tool, try to save the workspace and unsaved files.

   Caution Because the internal state of MATLAB might be corrupted, do not save existing files to the same filename. Instead, specify a new filename. The information in the new file might be corrupted or incomplete.
4. Exit MATLAB immediately after saving because any further usage would be unreliable.

5 Restart MATLAB. If the Error Log Reporter dialog box opens, send a report to The MathWorks.

Error Log Reporting

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. The error log contains the stack trace and information about the MATLAB software configuration. If the problem occurs repeatedly, make note of what seems to cause it, look for information about it in the MathWorks Bug Reports database, and if the problem is reproducible, please submit a Service Request via http://www.mathworks.com/support/contact_us/ts/help_request_1.html.

E-Mailing Error Log Reports. 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. It also will not open if you selected the option to never send error reports the last time the Error Log Reporter opened. If you experience abnormal termination but do not see the Error Log Reporter on subsequent startups, you can instead e-mail the reports.

Send e-mail to segv@mathworks.com with this file attached: `C:\Temp\matlab_crash_dump.####`. After you send the log file, delete it or move it to another location. If you do not delete the log file, the Error Log Reporter can detect it on the next startup and prompt you to send it, even though you already e-mailed it.

Recovering Data After an Abnormal Termination

If MATLAB terminates unexpectedly, you might lose information. After you start MATLAB again, you can try these suggestions to recover some of the information.

- Use the Command History or the file on which it is based, `history.m`, to run statements from the previous session. You might be able to approximately recreate data as it was prior to the termination. For more information, see “Overview of the Command History Window” on page 3-61.
- If you used the `diary` function or `-logfile` startup option for the session in which MATLAB terminated unexpectedly, you might be able to recover output.

- If you saved the workspace to a MAT-file during the session, you can recover it by loading the MAT-file. For more information, see “Viewing and Loading a Saved Workspace and Importing Data” on page 5-7, and “Saving the Current Workspace” on page 5-5.
- If you were editing a file in the Editor when MATLAB terminated unexpectedly, and you had the autosave preference enabled, you should be able to recover changes you made to files you had not saved.
- If you were in a Simulink session when a segmentation violation occurred, and you have the Simulink **Autosave Options** preference selected, note that the last autosave file for the model reflects the state of the autosave data prior to the segmentation violation. Because Simulink models might be corrupted by a segmentation violation, a model is not autosaved after a segmentation violation occurs.

Some of the above suggestions refer to actions you might have needed to take during the session when MATLAB terminated. If you did not take those actions, consider regularly performing them to help you recover from any future abnormal terminations you might experience.

2

Desktop

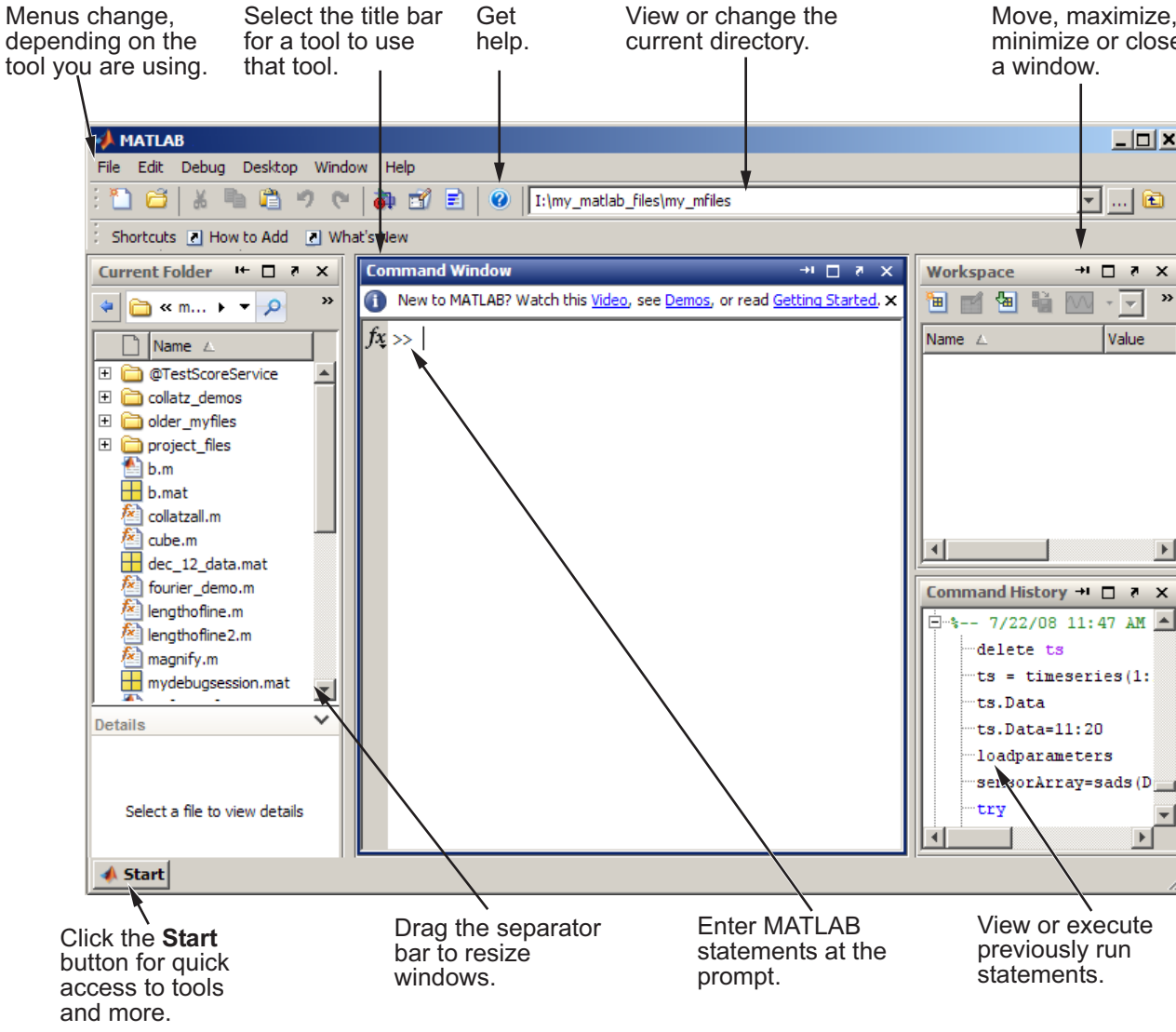
Desktop Overview

In this section...

About the Desktop

In general, when you start the MATLAB program, 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. The desktop manages tools differently from documents. The Command History and Editor are examples of tools, and an M-file is an example of a document, which appears in the Editor tool. For

details, see "Opening and Arranging Desktop Tools" on page 2-6 and "Opening and Arranging Desktop Documents" on page 2-21.

Summary of Desktop Tools

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

Desktop Tool	Description
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 language statements.
"Using the Current Folder Browser to Manage Files" on page 6-8	View files, perform file operations such as open, find files and file content, and manage and tune your files.
Editor	Create, edit, debug, and analyze M-files (files containing MATLAB language statements).
File Exchange	Access a repository of files, created by users for sharing with other users, at the MathWorks Web site.
Figures	Create, modify, view, and print figures generated with MATLAB.
File and Folder 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.

Desktop Tool	Description
Start Button	Run tools and access documentation for all your MathWorks products, and create and use toolbar shortcuts for MATLAB.
Variable Editor	View array contents in a table format and edit the values.
Web Browser	View HTML and related files produced by MATLAB.
Workspace Browser	View and change the contents of the workspace.

Opening and Arranging Desktop Tools

In this section...

See also "Examples of Desktop Arrangements" on page 2-42.

Opening Desktop Tools

To open a tool, select it from the **Desktop** menu. A check mark in front of the tool name on the menu indicates that the tool is open. The tool opens in the location it occupied the last time you used it. The dimensions of other open tools adjust to accommodate the newly opened tool. If you close and then reopen multiple tools sequentially, the location and size of the tools when you reopen them can be different from when you last closed them.

Tools and the documents associated with them can be part of the desktop. You can open a document and its associated tool at the same time, as follows:

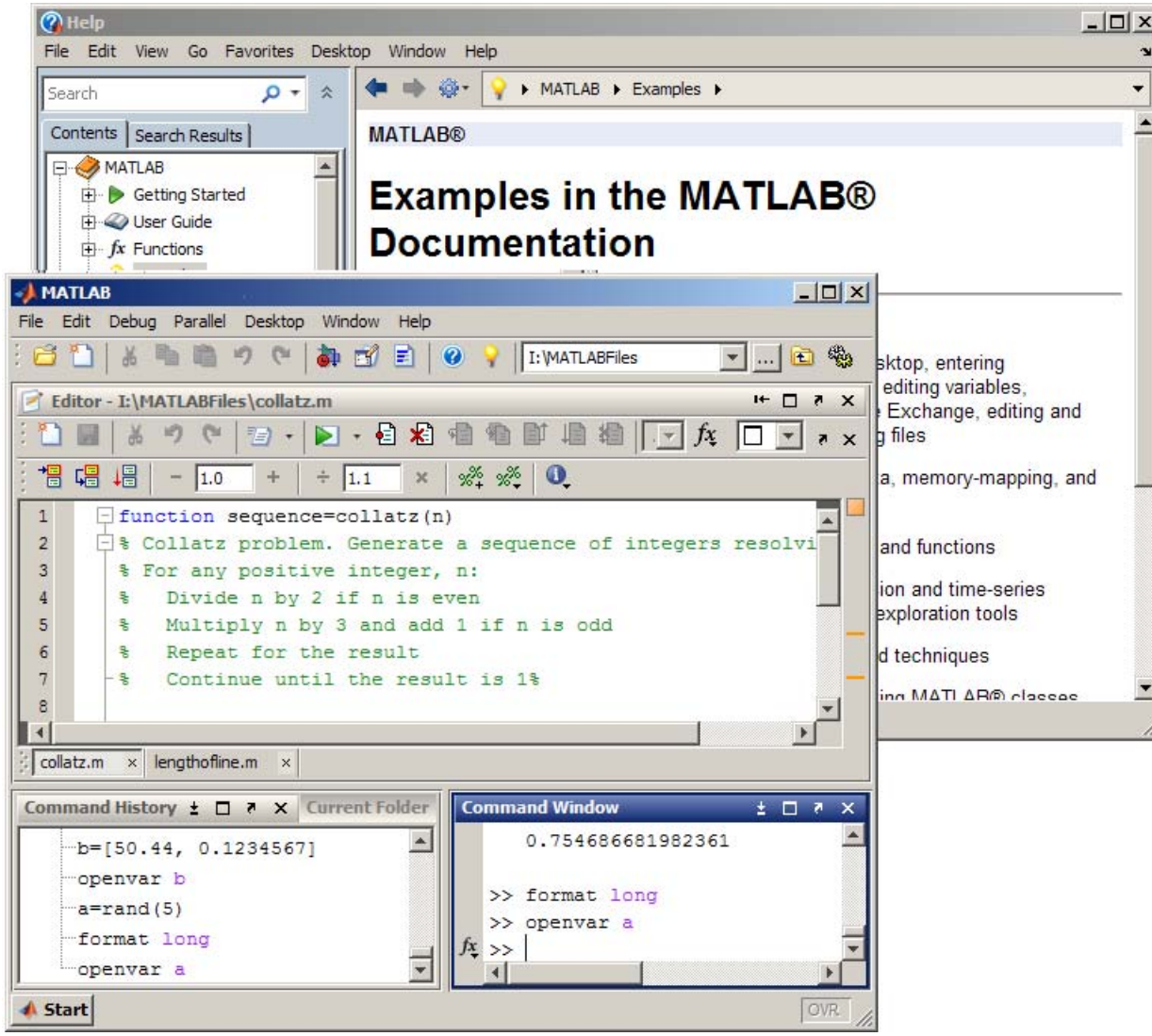
- Variable Editor — Open it by double-clicking a variable in the Workspace browser.
- Editor — Open it by creating a text file, such as an M-file, or opening an existing file. For instructions, see "Starting, Creating Files, and Closing the Editor" on page 8-7.

- Figures — Create figures using `plot` and other graphics functions.

You also can open most desktop tools by:

- Clicking the desktop **Start** button, selecting **Desktop Tools**, and then clicking the tool you want to open.
- Using a function. For example, type `helpbrowser` to open the Help browser. For information on how to open a given tool using a function, see the documentation for that tool.

The following example shows how the MATLAB desktop can look with the Command Window, Command History window, Help browser, and the files `collatz.m` and `lengthofline.m` open in the Editor. Because the Command Window is the active window, its title bar is dark blue.



Navigating Among Desktop Tools and Documents

- "Navigating Among Desktop Tools and Documents" on page 2-9

• "Making a Tool or Document the Active Window" on page 2-9

Navigating Among Desktop Tools and Documents

You can navigate among desktop tools and documents by:

- Choosing an entry in the **Window** menu
- Using a function that opens the tool or document
- Clicking the entry for an undocked tool or document on the Microsoft Windows task bar (or the equivalent for your platform)

Making a Tool or Document the Active Window

To make a tool or document the active window, do one of the following:

- Select the tool from the **Window** menu.

  The **Window** menu displays all open desktop tools and documents, as well as opened tools for other MathWorks products.

- Use the shortcut or mnemonic indicated on the **Window** menu for that tool or document.

  See "Keyboard Key Combinations" on page 2-68

- Run the function that opens the tool.

  - If the tool is already open, the command selects the tool.

  - If the tool is not already open, the command opens and selects the tool. For example, type `helpbrowser` to open or select the Help browser. The documentation for each tool includes information on the function for opening and selecting that tool.

Closing Desktop Tools

To close a desktop tool, do one of the following:

- Select the item on the **Desktop** menu.

  The check mark preceding the tool name on the menu clears and the tool closes.

- Click the Close box on the title bar for the tool.

- Select **File > Close *Toolname***.
- Right-click the Microsoft Windows task bar entry for an undocked tool and select **Close**.

When you close a tool, other tools in the desktop adjust their sizes accordingly.

For tools that contain documents, all documents in that tool close, as well. For the Editor, a dialog box appears asking you to save any documents that have unsaved changes. If you do not want to see that dialog box or save any unsaved changes, hold **Ctrl** and click the Close box.

Resizing Desktop Tools

To resize tools on the MATLAB desktop, you can use the mouse or the keyboard, as described in the following sections:

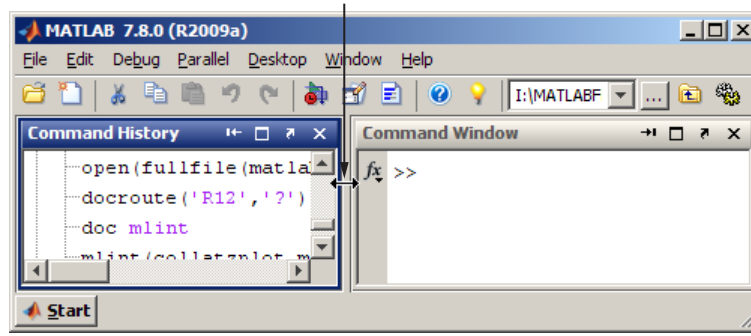
- "Resizing Desktop Tools Using the Mouse" on page 2-10
- "Resizing Desktop Tools Using the Keyboard" on page 2-11

Resizing Desktop Tools Using the Mouse

To expand or reduce the size of adjacent tool windows, use the pointer to drag the bar that appears between them. This bar is the separator bar. When you move the pointer onto the separator bar, the pointer assumes a different shape, as follows:

- On Windows platforms, when the pointer is between two tools or documents, it is a double-headed arrow ↔.
- On UNIX platforms, when the pointer is between two tools or documents, it is an arrow with a bar. →|
- When the pointer is between three or four documents, it is a four-headed arrow ✥.

Drag the separator bar to resize tools in the desktop.



Resizing Desktop Tools Using the Keyboard

You can use menu item mnemonics to resize desktop tools using the keyboard.

For example, suppose the Command Window is open on the desktop along with other tools. To make the Command Window the active tool:

1. Click in the Command Window.
2. Press **Alt+D, Z**. This action is the mnemonic equivalent of selecting **Desktop > Resize Command Window**.

   The pointer shape becomes an arrow.
3. Use the keyboard arrow keys to change the size of the Command Window.
4. Press **Enter** to accept the new size, or press **Esc** to return the Command Window to its original size.

Moving Tools Within the Desktop

To move the location of tools on the MATLAB desktop, use the mouse or the keyboard, as described in the following sections:

- "Moving Tools Using the Mouse" on page 2-12
- "Moving Tools Using the Keyboard" on page 2-14

Moving Tools Using the Mouse

To move a tool to another location on the MATLAB desktop using the mouse, follow these steps:

1. Drag the title bar of the tool to where you want the tool to be.

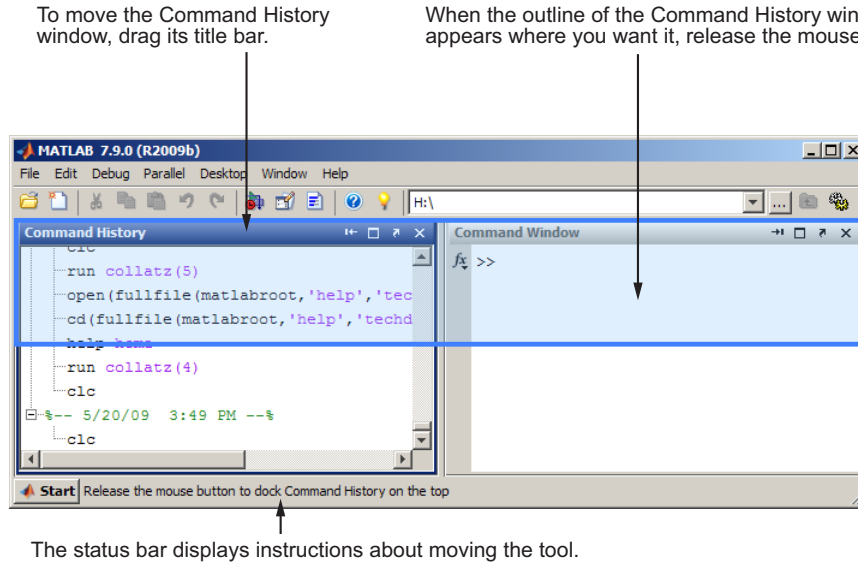
   As you drag the tool, an outline of it appears. The status bar indicates where the tool moves if you release the mouse. For instance, it can display:

   - Release the mouse to dock the Editor on the top.
   - Release the mouse to tab-dock the Current Folder.
   - Release the mouse to leave the Editor in the current location.

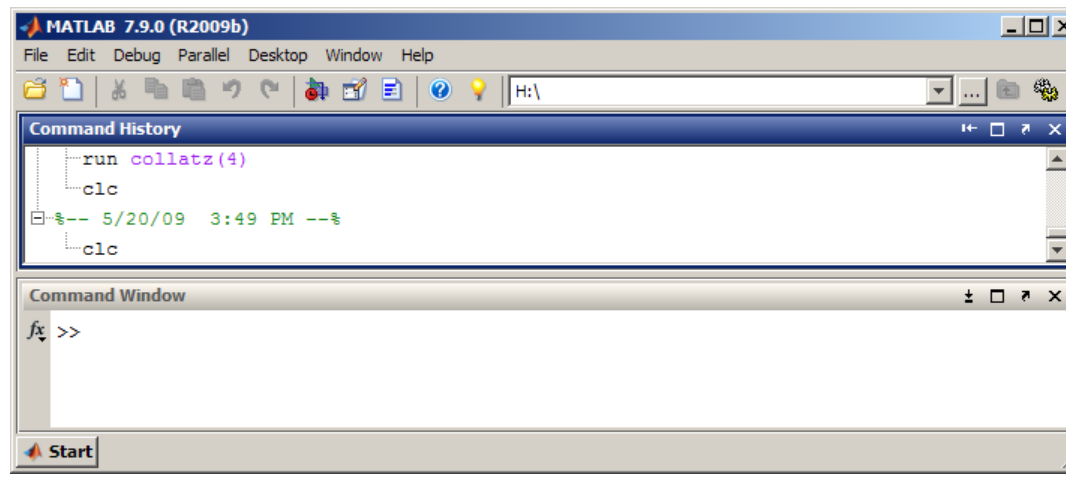
2. When the outlined position is where you want the tool to be, release the mouse button.

   The tool stays at the new location.

The following illustration shows how it looks as you drag the Command History tool above the Command Window. When you begin dragging the Command History tool, the outline appears around the tool. When you drag it across the boundary separating the two tools, the outline indicates the top-bottom arrangement. If you release the mouse button, you change the arrangement from side-by-side to top-bottom.



Other tools on the desktop automatically resize to accommodate the new configuration. The following example shows how the desktop looks after you move the Command History tool above the Command Window.



Moving Tools Using the Keyboard

To move desktop tools using the keyboard, follow the menu item mnemonics. For example, suppose the Command Window and other tools are currently open on the desktop. To move the Command Window to a new location, follow these steps:

1. Make the Command Window the active tool by pressing **Ctrl+0**.
2. Press **Alt+D, V**, which is the mnemonic equivalent for selecting **Desktop > Move Command Window**.

   The pointer shape becomes an arrow.
3. Use the arrow keys to move the outline of the Command Window to a new location.
4. Press **Enter** to keep the tool at the new location, or press **Esc** to return the Command Window to its original position.

Undocking Tools to Move Them Outside the Desktop

You can move a tool outside the MATLAB desktop (called undocking) to make it larger or easier to use. For example, when referring to the online documentation, you can move the Help browser off the desktop and enlarge it.

To move a tool outside the desktop:

1. Select the tool to make it active.
2. Perform one of the following:
   - Click the Undock button in the title bar of the tool you want to move outside the desktop.
   - Select **Undock** for that tool from the **Desktop** menu; the tool must be the currently active one.
   - Drag the title bar of the tool outside the desktop. As you drag, an outline of the tool appears. Release the mouse.

The tool displays 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.

Moving Undocked Tools Back onto the Desktop

To move a tool that is outside the MATLAB desktop back onto the desktop, do one of the following:

- Click the Dock button in the menu bar for that tool.
- Select **Dock** from the **Desktop** menu for that tool.

The tool moves onto the desktop and other tools within the desktop automatically resize to accommodate the new tool.

Grouping Desktop Tools Together

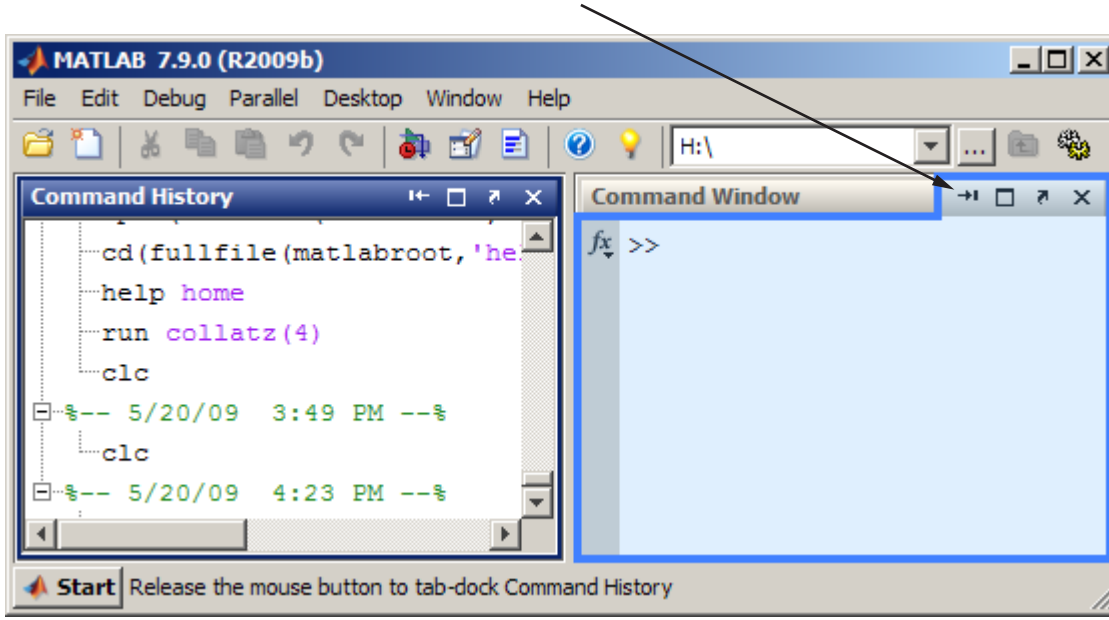
You can group tools so that they occupy the same location on the MATLAB desktop. Basically, you are stacking one tool on top of another. Then, you can access the individual tools using the tool name on the title bar:

To group tools:

1. Drag the title bar of one tool on the desktop on top of another tool on the desktop.

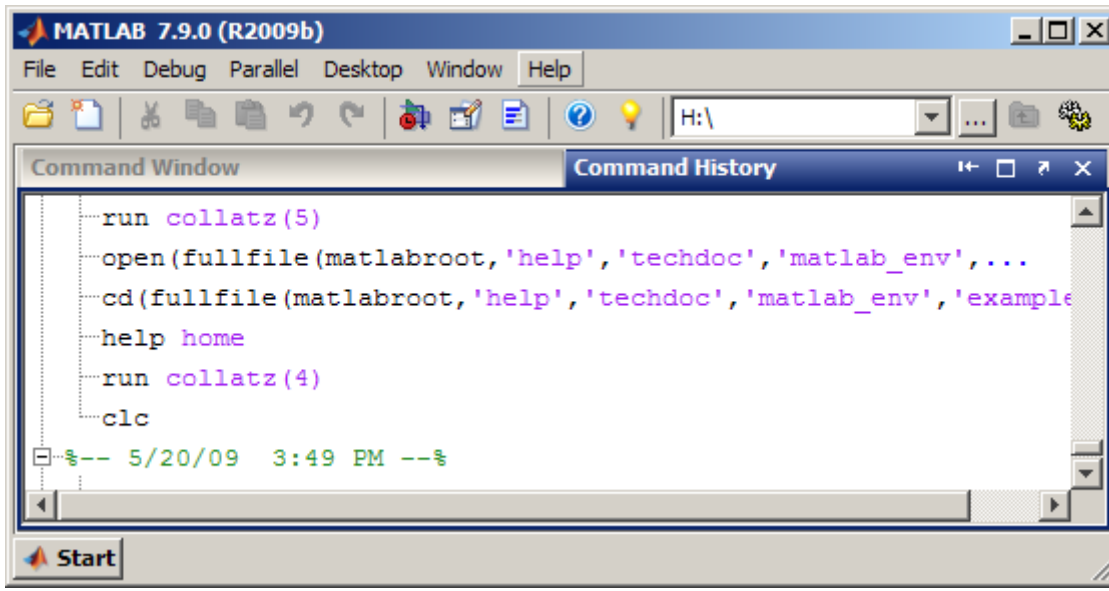
   An outline of the tool you are dragging overlies the target tool.

Outline of the Command History window being dragged on top of the Command Window to group both tools together



2 Release the mouse.

Both tools occupy the same space. Labeled tabs appear at the top of that space.



To view a grouped tool, click the title bar for the tool. The selected tool moves to the foreground and becomes the currently active window.

When you click the Close box for a tool grouped with other tools, that tool closes. You cannot close all the grouped tools at once. Instead, close each tool individually.

Right-click the title bar for a tool and use the context menu to close, undock, maximize, or minimize the tool.

Maximizing Available Space on the Desktop

To hide the title bars for desktop tools so they use less space, select **Desktop > Titles**. This action clears the check mark next to the **Titles** menu item. Identify a desktop tool with a hidden title by hovering over the area where the title bar used to be. A ToolTip displays the name of the tool.

Maximizing Tools Within the Desktop

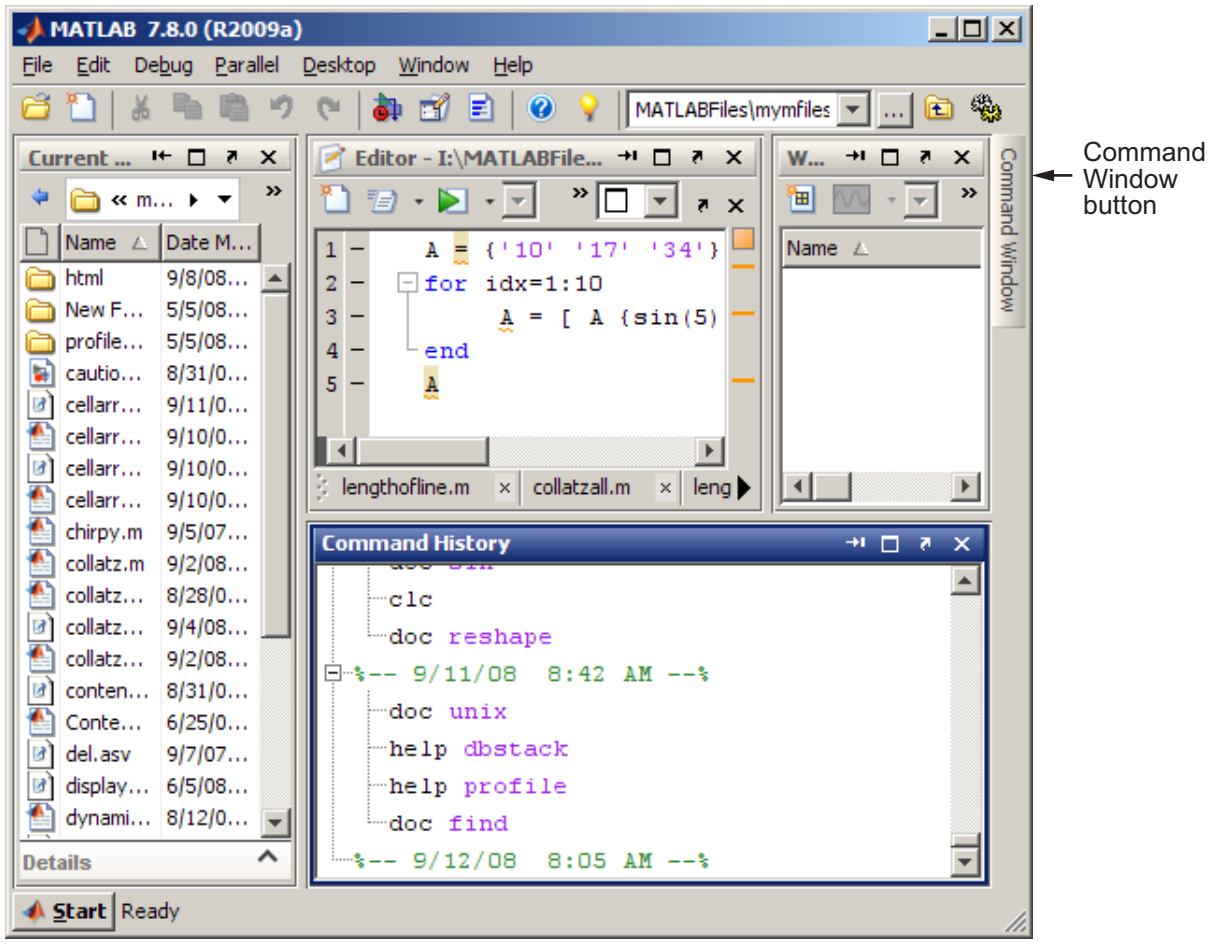
MATLAB software provides multiple ways to maximize tools on the desktop. It also has multiple ways of restoring the desktop to the layout in place before you resized it. For example:

- To resize the active tool so it occupies the entire MATLAB desktop do one of the following:
  - Double-click the title bar in that tool.
  - Select **Desktop > Maximize Toolname**.
  - Click the Maximize button on the tool title bar.

- To return to the layout as it appeared before you maximized it, do one of the following:
  - Double-click the maximized title bar for that tool.
  - Select **Desktop > Restore Toolname**.
  - Click the Restore button on the title bar in that tool.

Minimizing Tools Within the Desktop

You can minimize any tool on the desktop, which creates a button representing the tool along an edge of the desktop. For example, suppose you minimize the Command Window. The desktop looks like the following, although the layout of the desktop and the location of the button can be different for your desktop.



MATLAB software provides multiple ways to minimize tools in the desktop, restore the previous desktop layout, and manipulate the location of the tool button. For example:

- To minimize a tool, do one of the following:
  - Select **Desktop > Minimize Toolname**.
  - Use the Minimize button on the title bar for the tool.

The button for the tool appears along the edge indicated by the minimize arrow in the **Desktop** menu item or by the arrow on the button.

- To view or use a minimized tool, hover over or click the button for the tool. This action temporarily opens the tool on the desktop. When you finish using the tool, click the Minimize button or another tool. The tool appears again as a button along the edge of the desktop.
- To return the tool to the position it occupied before you minimized it, do one of the following:
  - Double-click the button for the tool.
  - Right-click the button for the tool, and then select **Restore > Toolname**.
  - Hover over or click the button for the tool, and then click the Restore button  on the title bar of the tool.
- To move the button for the tool, drag it to a different edge.

  If you drag the button to a nonedge location on the desktop or outside of the desktop, it moves the tool and opens restores it.

Opening and Arranging Desktop Documents

In this section...

Opening Documents

Use the document bar to go to a document that is open, but not in view. The names of all open documents appear on the document bar. Click the document name to open the document. If the document bar is not open, select **Desktop > Document Bar > Bar Position** and select the position for it, for example, **Right**. For more information, see "Navigating Among Open Documents Using the Document Bar" on page 2-24.

Entries for undocked documents appear on the Windows task bar, or the equivalent for your platform. Click the task bar entry for a document to make that document active.

When you open MATLAB documents, they open in the associated tool and appear in the position they occupied when last used. Figures open undocked, regardless of the last position occupied. If the tool is not already open, it opens when you open the document.

How you open a document depends on the document type, as described in the following table.

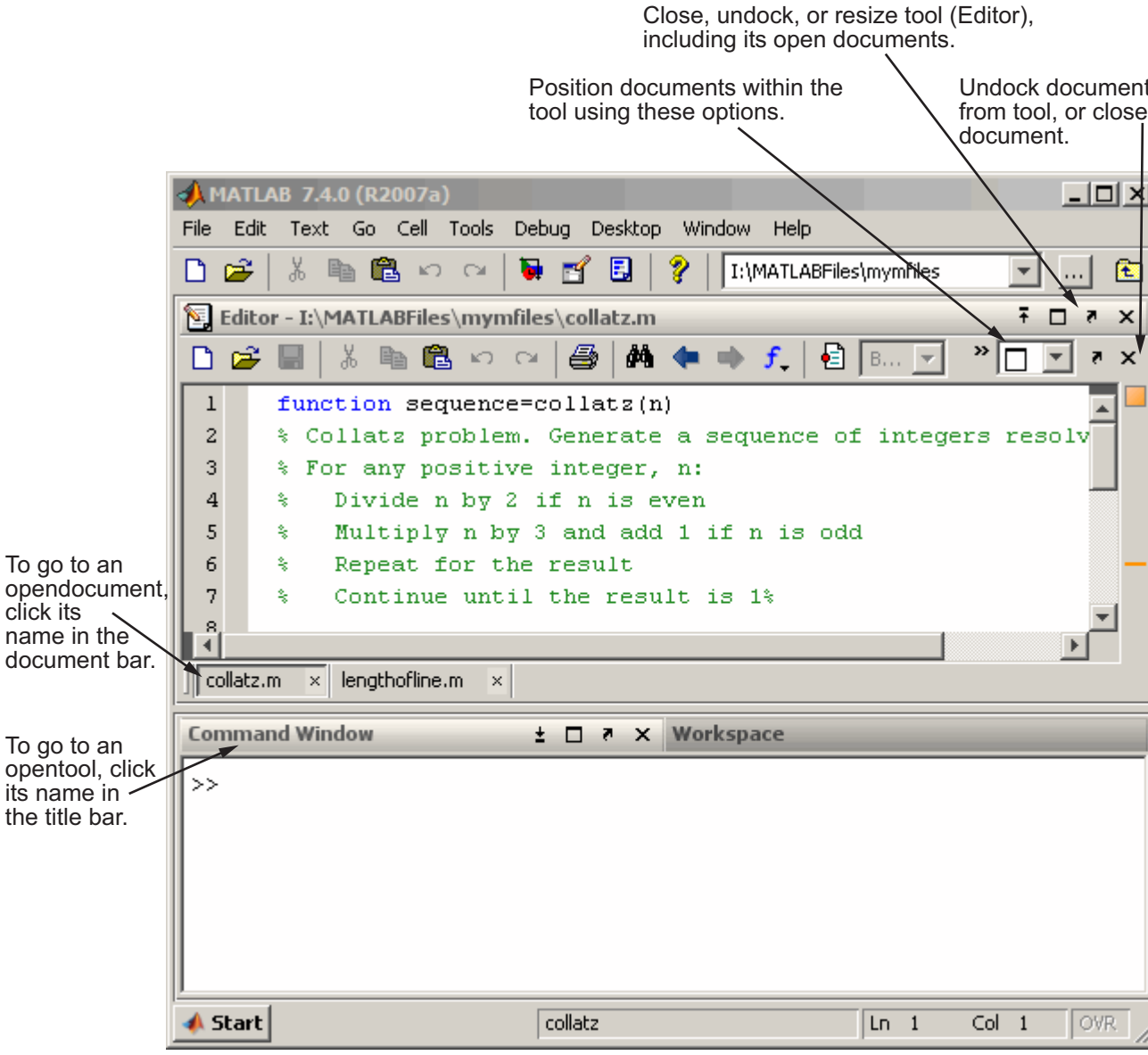
Document Type and Tool	How to Open Document	Where Document Appears by Default	Other Techniques to Open Document
M-file (or other text file) in the Editor	Click the **Open file** button on the desktop toolbar and select the file.	In the last location of the Editor. The default location for the Editor is outside the desktop.	"Opening Existing Files in the Editor" on page 8-9
Variable in the Variable Editor	Double-click a variable in the Workspace browser.	In the last location of the Variable Editor. The default location of the Variable Editor is docked on the desktop.	"Opening the Variable Editor" on page 5-24
HTML or similar page in the Web browser	Double-click the file name in the Current Folder browser.	In the last location of the Web browser, replacing the existing Web browser document.	"Using Web Browsers from MATLAB" on page 2-103
Figure	Use the `plot` function.	In a figure window, outside the desktop.	Any other function or tool that creates a figure window.

Example of Working with Documents on the Desktop

Some common actions for working with documents on the desktop are:

- Select a document from the document bar, making it the active open document.
- 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 2-42.


Close, undock, or resize tool (Editor), including its open documents.
Position documents within the tool using these options.
Undock document from tool, or close document.
To go to an opendocument, click its name in the document bar.
To go to an opentool, click its name in the title bar.
MATLAB 7.4.0 (R2007a)
File Edit Text Go Cell Tools Debug Desktop Window Help
I:\MATLABFiles\mymfiles
Editor - I:\MATLABFiles\mymfiles\collatz.m
function sequence=collatz(n)
% Collatz problem. Generate a sequence of integers resolv
% 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%
collatz.m
lengthofline.m
Command Window
Workspace
>>
Start
collatz
Ln 1 Col 1
OVR

Navigating Among Open Documents Using the Document Bar

When you have more than one document open in a tool, each document appears either maximized (the default), tiled, or floating (cascading). Tiled and floating arrangements make multiple documents visible simultaneously. The document bar shows the names for all open documents docked together in a tool.

Making a Document Active

To make a document that is open and in view active, click it.

To make an open document that is not in view active, do one of the following:

- Select the document from the document bar.

  If all the document names do not fit on the document bar, use the **More Documents** button ▸ on the document bar. This button enables you to see the names of additional open documents. Hover over the arrow to scroll automatically through all the names, or click the arrow to move quickly through the names.
- From the **Window** menu, select the document name.
- From the **Window** menu, select **Next Tab** to make the next document on the document bar active (relative to the currently active document).
- From the **Window** menu, select **Previous Tab** to make the previous document on the document bar active (relative to the currently active document).

See also "Performing Desktop Actions Using the Keyboard" on page 2-68.

Adjusting the Document Bar

You can show, hide, move, alphabetize, and adjust the size of the tabs in the document bar as described in the following table.

To Accomplish This:	Do This:
Show the document bar.	Select **Desktop > Document Bar > Bar Position**, and then select a location, for example, **Right**.
Hide the document bar.	Select **Document Bar > Bar Position > Hide** from the **Window** menu or the document bar context menu.
Move the document bar.	Do one of the following: • Drag it to another location • Select a new location from the **Desktop > Document Bar > Bar Position** submenu.
Alphabetize the names of the documents on the document bar. Alphabetizing is useful if you have many documents open at once.	Do one of the following: • Right-click on the document bar and select **Alphabetize**. • Select **Desktop > Document Bar > Alphabetize**.
Reorder document names on the document bar.	Do one of the following: • Drag a document name to a different position in the document bar. • Select **Move documentname On Bar** and select a direction. For example, select **to Beginning** from either the **Desktop > Document Bar** menu or from the document bar context menu.
Widen or narrow document names on the document bar. If document names are long, or if you have many documents open, the entire document name does not display. Instead, you see the first	Do one of the following: • When the document bar is on the top or bottom drag the separator bar between two names on the bar. (Do this, for example, to see an entire document name.)

To Accomplish This:	Do This:
few characters followed by ellipsis (. . .).	• When the document bar is on the left or right, change the width of the bar by dragging its left or right edge.

Positioning Documents

You can position open documents so that one document or multiple documents are in view from within a tool. Select the arrangement from the **Window** menu or use the Arrange Documents drop-down menu  , as described in the sections that follow. When you *tile* documents, they are all visible within the tool, arranged in a grid pattern.

- "Viewing One Document (Default)" on page 2-26
- "Viewing All Open Documents, Layered on Top of One Another" on page 2-26
- "Viewing Documents, Side-By-Side" on page 2-27
- "Viewing Open Documents, One Above the Other" on page 2-27
- "Viewing Open Documents, Tiled Within the Tool" on page 2-27
- "Viewing a Subset of Open Documents, Tiled Within the Tool" on page 2-31

Viewing One Document (Default)

To have one document in view that occupies the entire tool (the default), do one of the following:

- Select **Window > Maximize**.
- From the **Arrange Documents** drop-down menu, choose the Maximize option ❐.

The illustration in "Example of Working with Documents on the Desktop" on page 2-22 shows this arrangement.

Viewing All Open Documents, Layered on Top of One Another

You can use the **Float** or **Cascade** options to layer open documents one on top of another. To do this, use one of the following methods:

- Select **Window > Float**.
- In the **Arrange Documents** drop-down menu, choose the **Float** option 🗗.

  Optionally, select **Window > Cascade** to make the document arrangement neater.

Viewing Documents, Side-By-Side

You can use the **Tile** option to view two documents side-by-side. To do this, use one of the following methods:

- Select **Window > Left/Right Tile**.
- In the Arrange Documents drop-down menu, choose the Left/Right Tile option ◫.

See also the Editor "Split Screen Display" on page 8-63 that enables you to view two different parts of the same file simultaneously.

Viewing Open Documents, One Above the Other

You can use the **Top/Bottom Tile** option to view two documents stacked one above the other by using one of the following methods:

- Select **Window > Top/Bottom Tile**.
- In the Arrange Documents drop-down menu, choose the Top/Bottom Tile option ⊟.

See also the Editor "Split Screen Display" on page 8-63 that enables you to view two different parts of the same file simultaneously.

Viewing Open Documents, Tiled Within the Tool

To have all open documents in view, tiled within the tool, follow these steps:

1. Select the tiling option using one of these methods:
   - Select **Window > Tile**.

     On the Apple Macintosh platform, this option might be unavailable, so use the drop-down menu instead.

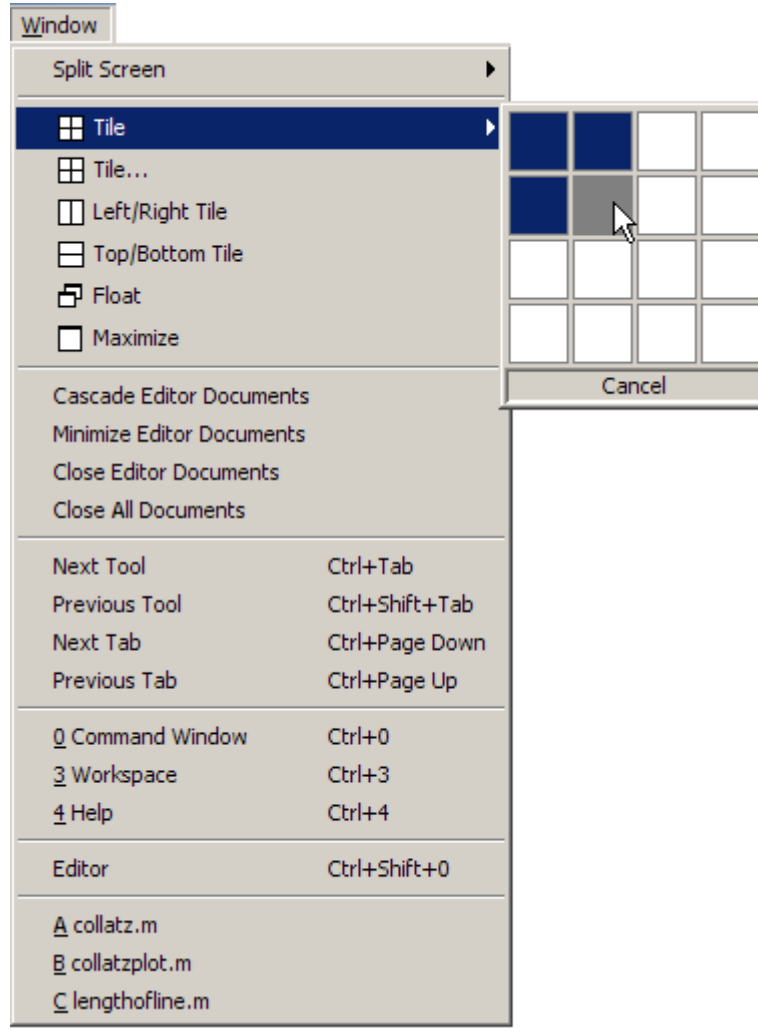
- In the **Arrange Documents** drop-down menu, choose the Tile option ⊞.

A four-by-four grid displays.

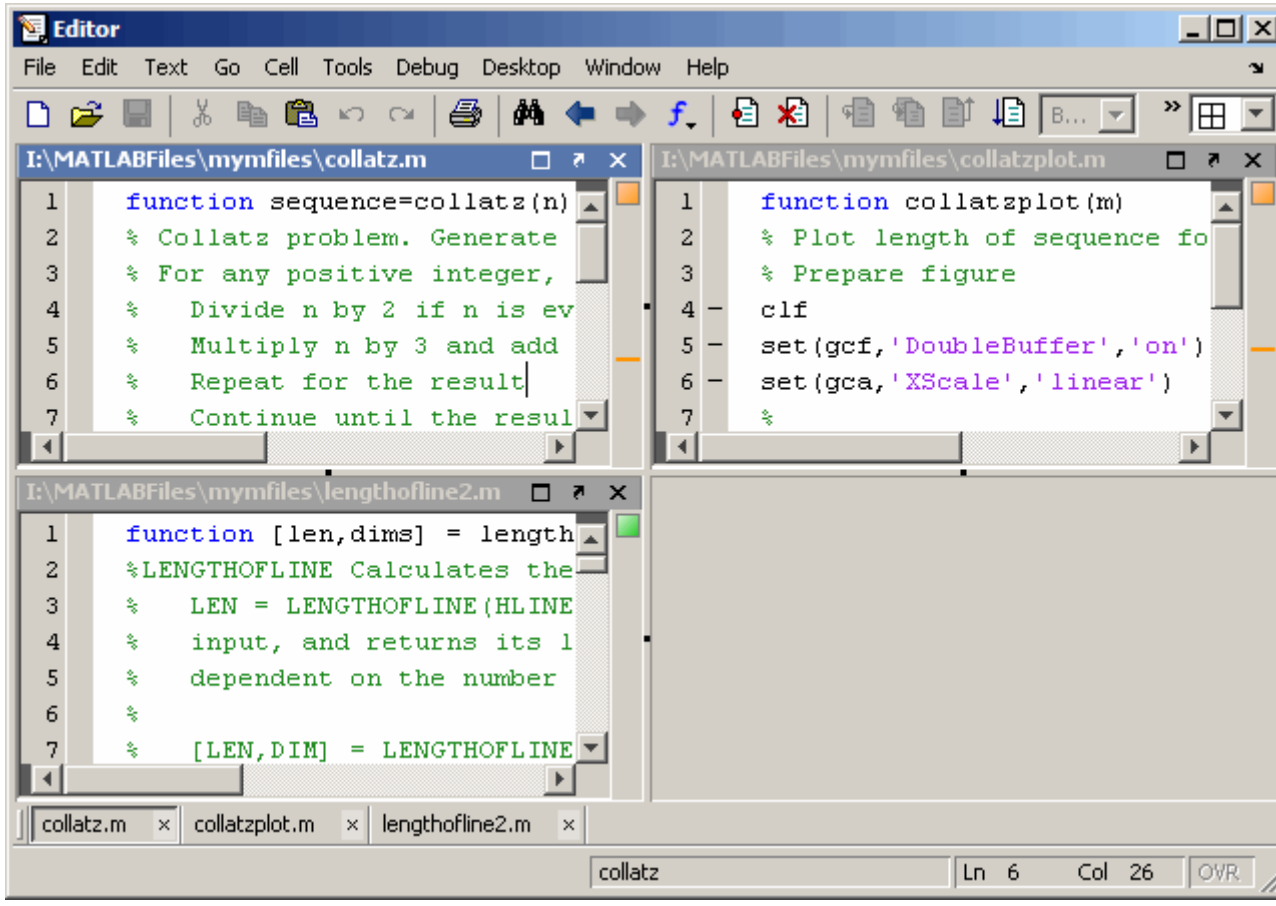
2 Move the pointer across the grid to define the number and position of the tiles, as shown in the following illustration.

You can select more or fewer tiles than there are open documents. In the example, there are three open documents, but you must select four tiles to make a square grid shape. The tiles that will contain documents appear blue, whereas the tiles that will be empty appear gray.

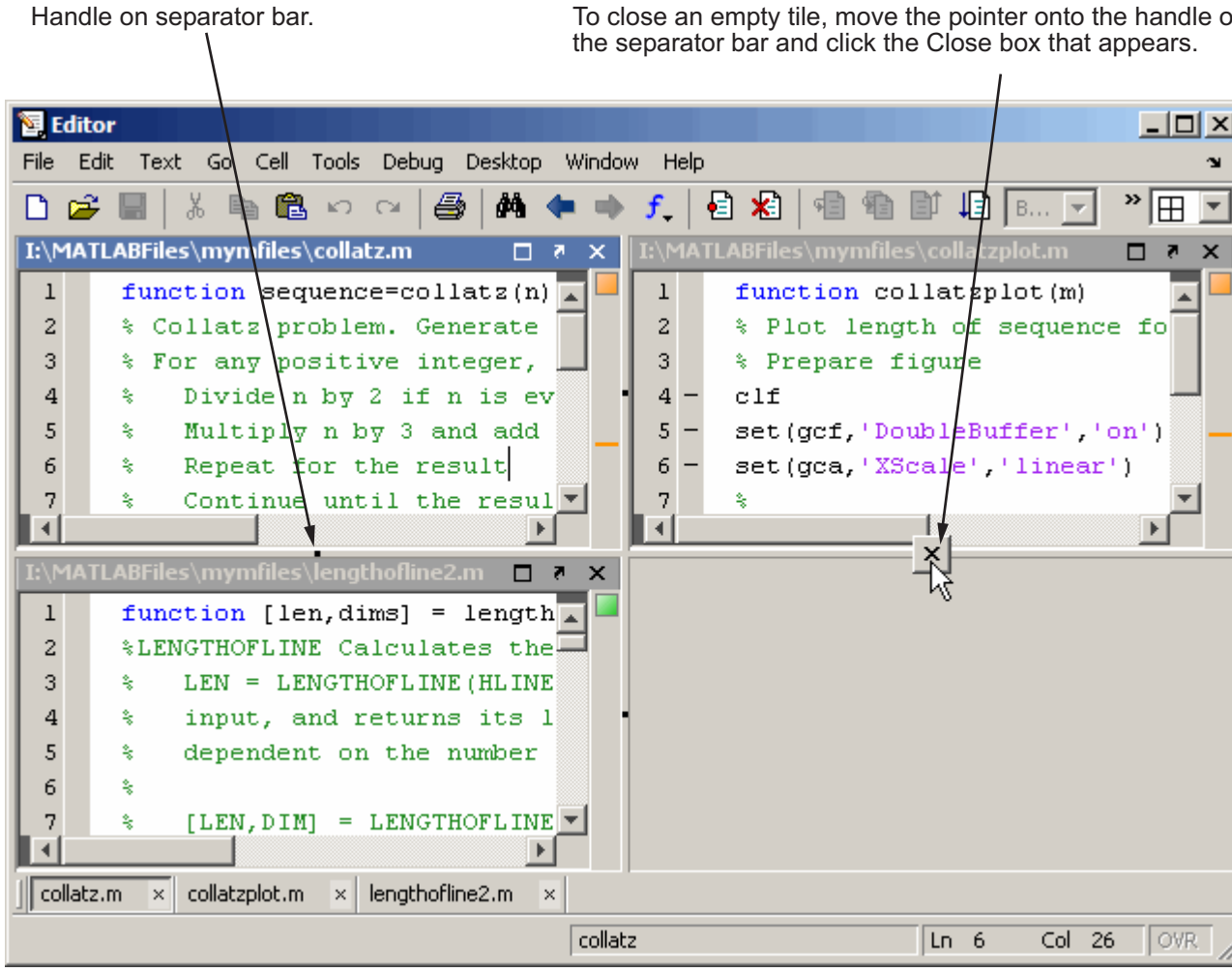
This example shows how to select an arrangement so that all three documents will be in view. The resulting arrangement has two documents above, one below, and one empty tile.



The following arrangement shows three documents tiled in the Editor. The Editor is undocked from the desktop.



To close an empty tile, move the pointer over the handle ▪ on the separator bar, and then click the Close box that appears. If you click the handle between two open documents, both documents stay open, but one moves on top of the other.



Viewing a Subset of Open Documents, Tiled Within the Tool

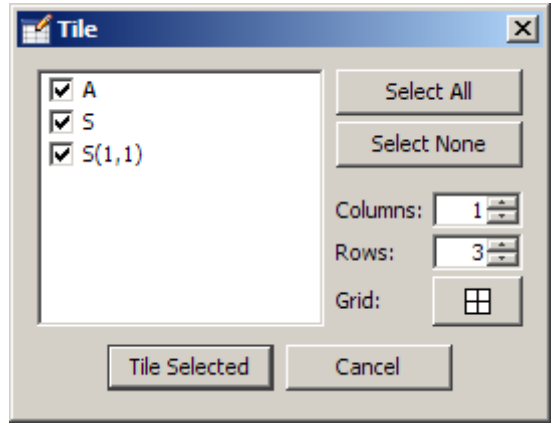
To view only a subset of all your open documents displayed in tile format, follow these steps:

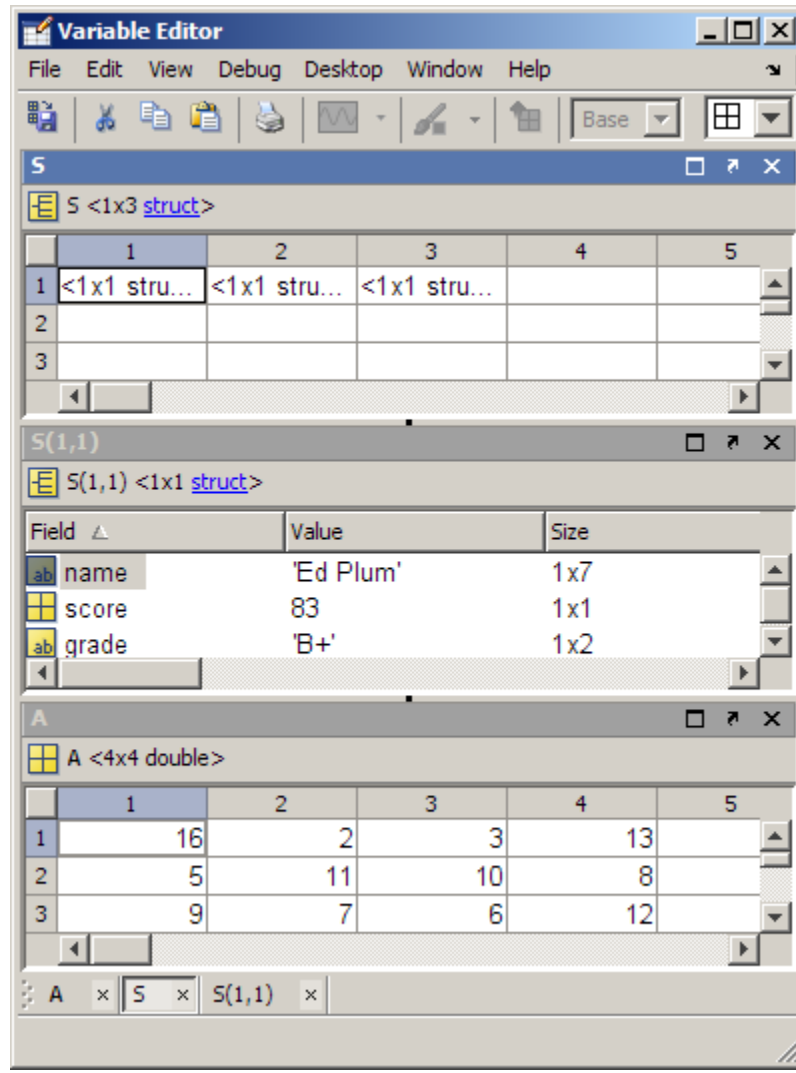
1 Select **Window > Tile ...**.

The Tile dialog box opens.

2 Indicate the documents you want to view and the grid pattern to use for the arrangement of their display.

The following illustrations show how to specify the arrangement for three variables in three rows in the Variable Editor, and the resulting configuration.





Moving and Resizing Documents

You can move and resize documents to organize them as you want, as described in the following table.

To Accomplish This:	Do This:
Minimize all open documents in a tool	Make that tool active, and then select **Window > Minimize *Toolname* Documents**
Float documents	Select **Windows > Float**.
Minimize (hide) a floating document	Click the minimize button – in the document title bar.
Access a minimized document	Select its name from the document bar or the **Window** menu.
Move or resize a maximized document	Move or resize the tool that contains it.
Make a document larger when it is next to an empty tile	Hover over the handle on the separator bar, and then click the Close box that appears.
Resize tiled documents	Drag the separator bar that is between the documents.
Move tiled documents	Drag the title bar of the document to another tile. If you drag it to a tile that already contains a document, the document you are dragging covers up the other document.

Closing Documents

There are many ways to close a document. Use any of the following methods:

- Click the Close box ⊠ in the title bar for the tool.
- Right-click the title bar for the document, and then select the **Close** option.
- Right-click the name of the document in the document bar, and then select one of the **Close** options.
- Click the Close box ⊠ next to the name of the document in the document bar.

- Place the mouse pointer over the name of the document in the document bar, and then click the middle mouse button. (Works on Microsoft Windows and Linux® only.)
- Select **Window > Close *`Toolname`* Documents**.
- Select **File > Close *`Current_Document_Name`***.
- For an undocked document or tool, right-click the Windows task bar entry (or equivalent for your platform) and select **Close**.
- When there are open documents, undocked from within their tools, close all open documents and the tool by selecting **Window > Close All Documents** from the desktop.

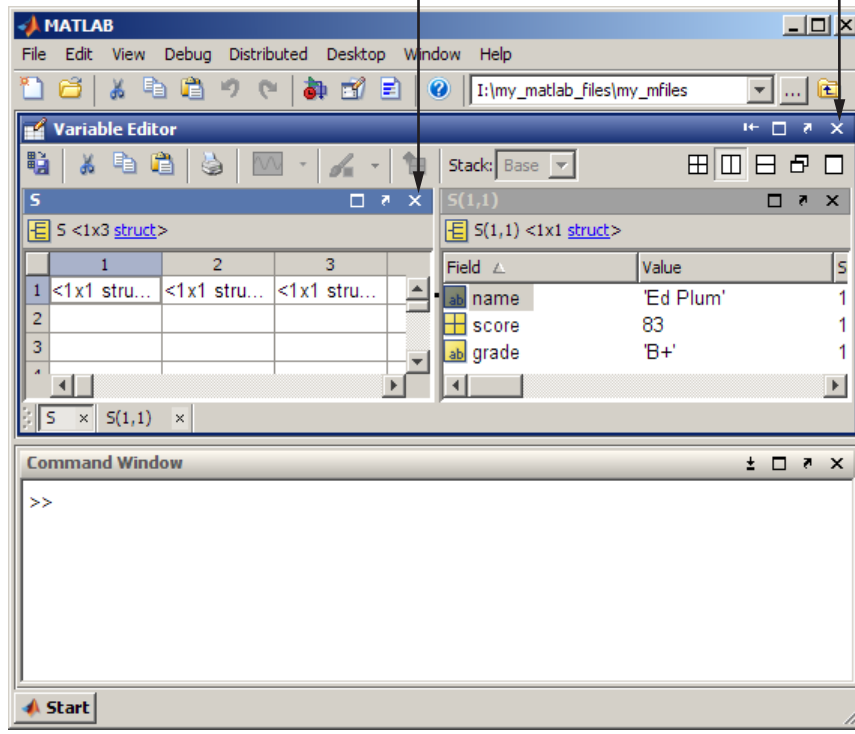
For example, in the undocked Editor, select **Window > Close Documents** to close all documents in the Editor. The Editor remains open with no documents in it, and any undocked Editor documents remain open.

To close a document, click the close box in the document's toolbar.

To suppress the save confirmation dialog box, which closes documents without saving changes, hold **Ctrl** while clicking the close box.

To close a tool and all the documents within it, click the close box in the tool's title bar.

Any undocked documents remain open.



In the Editor, 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 Close box for the document.

Moving Documents Outside of the Desktop (Undocking)

You can move a tool outside of the MATLAB desktop (called undocking) to make it larger or easier to work with. For example, you can move the Help browser outside of the desktop when referring to the online documentation.

To move a tool outside the desktop, do one of the following:

- Click the Undock arrow on the title bar of the tool you want to move outside the desktop.
- Make the tool you want to move outside the desktop active. Then for that tool, select **Desktop > Undock**.
- Drag the title bar of the tool outside the desktop. As you drag the title bar, an outline of the tool appears outside of the desktop. When the outline appears where you want the tool to be, release the mouse.

The tool displays outside the MATLAB desktop and an entry for it appears in the Windows task bar. Tools within the desktop automatically resize accordingly.

Docking Documents and Tools

To dock documents and their associated tool, click the Dock button on the menu bar for the tool.

If you dock a tool that includes documents, the tool and the documents within the tool move onto the desktop. For example, when you dock the Editor and it has open files, both the Editor and the documents move onto the desktop.

When you dock a document, it moves to the position in the tool that it occupied before you undocked the document.

Grouping Documents in a Tool Outside the Desktop

To group all the documents for a tool outside of the desktop, undock the tool from the desktop, not just the individual documents.

If you have already undocked all the documents and closed the empty tool that had contained them, follow these steps:

1. Select **Desktop > Dock All in Editor**, for example.

   This selection moves all the documents into the tool in the desktop.

2. Undock the tool.

Managing Desktop Layouts

When you end a session, MATLAB saves the current desktop arrangement. The next time you start MATLAB, the desktop appears like the way you left it. However, some tools, such as the Help browser, Web browser, and Array Editor do not reopen automatically, even if they were open when you ended the last session. You can use startup options to specify tools that you want to open on startup. For example, to have the Help browser open each time you start MATLAB, add `helpbrowser` to a `startup.m` file. For more information, see "Startup Options" on page 1-17.

You can also use predefined layouts, and you can save your own layouts for later reuse.

In this section...

Saving a Desktop Layout

To save your current desktop arrangement:

1. Select **Desktop > Save > Layout**.
2. Assign a name to the layout in the resulting dialog box, and then click **OK**.

MATLAB stores the arrangements you save as XML files in the preferences folder for MATLAB. Type `prefdir` in the Command Window to display the location of these XML files. The layout last used in a session is `MATLABDesktop.xml`. The `MATLABDesktop.xml` file loads when you startMATLAB and is overwritten when you close MATLAB.

Reusing a Saved or Predefined Desktop Layout

Select **Desktop > Desktop Layout**, and then select the name of the layout you want to use.

MATLAB includes the following predefined layouts:

- **Default** — Contains the Current Folder, Command Window, Workspace Browser, and Command History windows.
- **Command Window Only** — Contains the Command Window only.
- **History and Command Window** — Contains the Command History window and Command Window.
- **All Tabbed** — Contains all desktop tools, opened, maximized, and tabbed together.
- **All but Command Window Minimized** — Contains all tools, opened and minimized in the desktop, except for the Command Window and sometimes the Editor. The Command Window and the Editor (if it contains a document) remain maximized.

When you select a saved or predefined layout, document tools already open in the desktop remain open.

Renaming a Saved Desktop Layout

Rename a desktop layout that you have previously created and saved as follows:

1. Select **Desktop > Organize Layouts**.
2. In the resulting dialog box, select a layout, click the **Rename** button.
3. Type the new name over the existing name.
4. Click **Close**.

You can rename desktop layouts that you created only.

Deleting a Saved Desktop Layout

Delete a desktop layout that you have previously created and saved as follows:

1. Select **Desktop > Organize Layouts**.
2. In the resulting dialog box, select a layout, click the **Delete** button, and then click **Close**.

You can delete desktop layouts that you created only.

Restoring the Default Desktop Layout

If you are dissatisfied with your current desktop arrangement, you can restore it to the default arrangement as follows:

Select **Desktop > Desktop Layout > Default**

The default arrangement is that which appeared when you first installed MATLAB.

Examples of Desktop Arrangements

In this section...

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. For details, see "Opening and Arranging Desktop Tools" on page 2-6 and "Opening and Arranging Desktop Documents" on page 2-21.

Tool Outside of Desktop and Other Tools Grouped Inside Desktop Example

This example shows two ways you can increase the size of a tool:

- Move a tool outside of the desktop to increase its size.

  In the illustration that follows, the Help browser is outside of the desktop and made larger. To move a tool outside of the desktop, click the Undock button in the title bar of the tool when the tool is in the desktop.

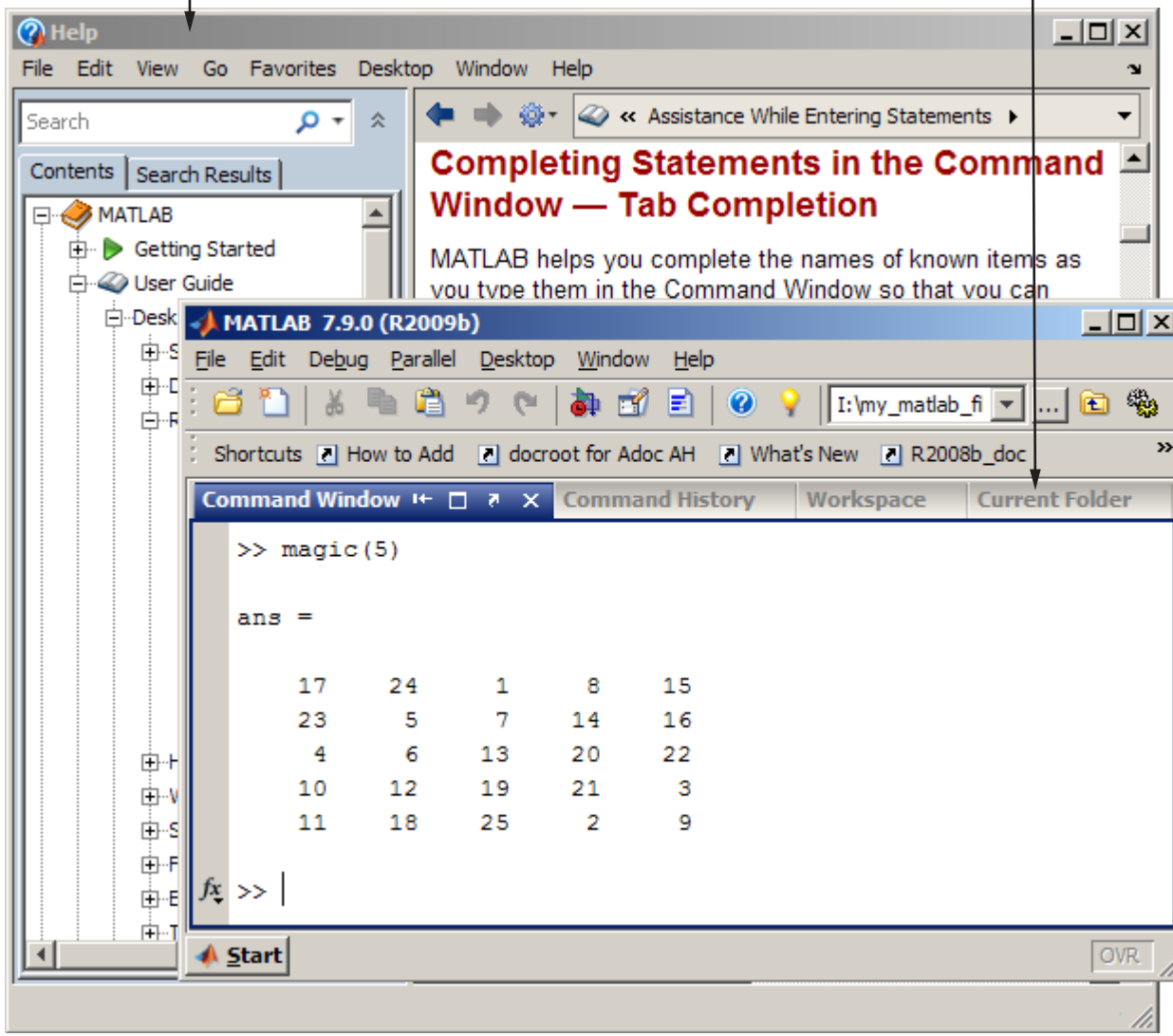
- Group tools inside the desktop, and then access a particular tool by clicking the name of that tool in the title bar.

  In the illustration that follows, the Command Window, Command History, Workspace browser, and Current Folder browser appear together as a group. To achieve this arrangement, drag the title bar of one tool on top of the title bar of the tool (or tools) with which you want to group it.

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

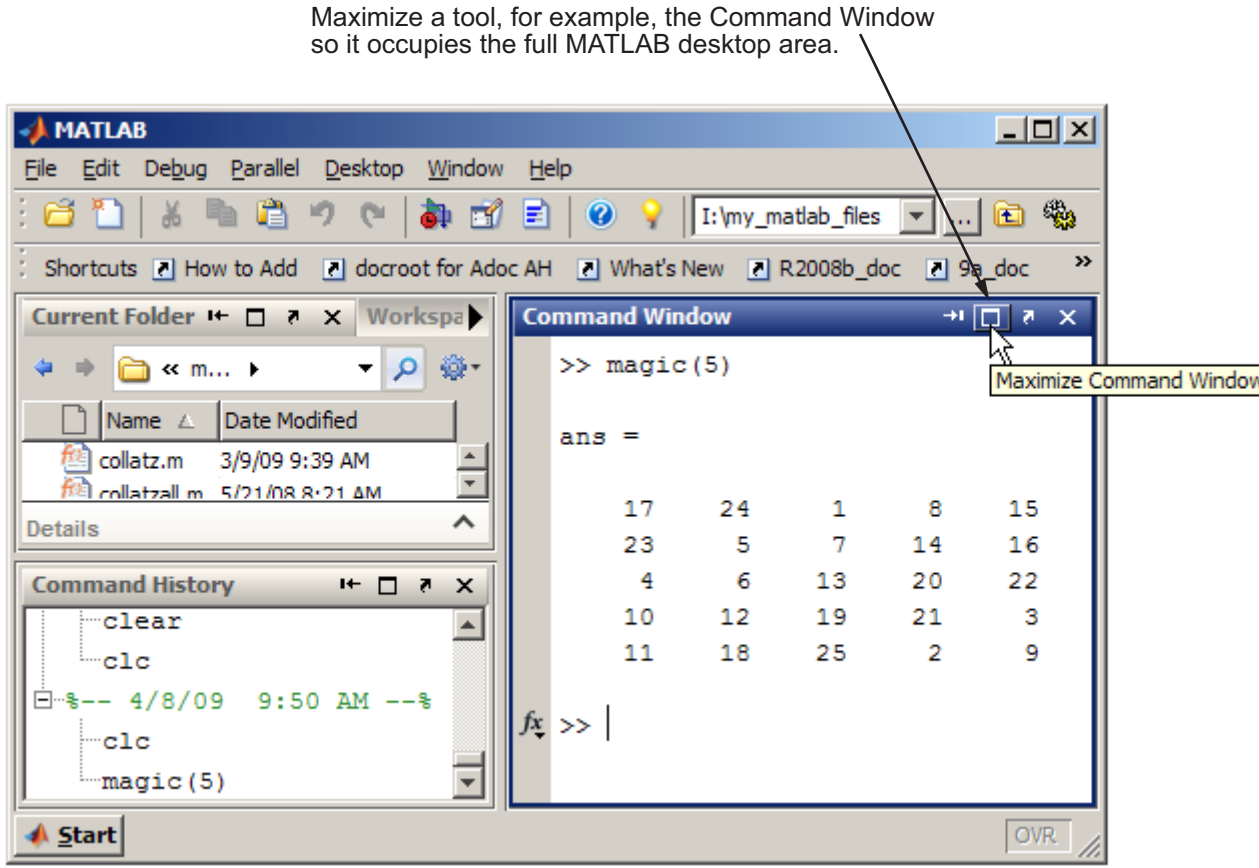
Four desktop tools grouped together.

Click a tool name to make that tool active.



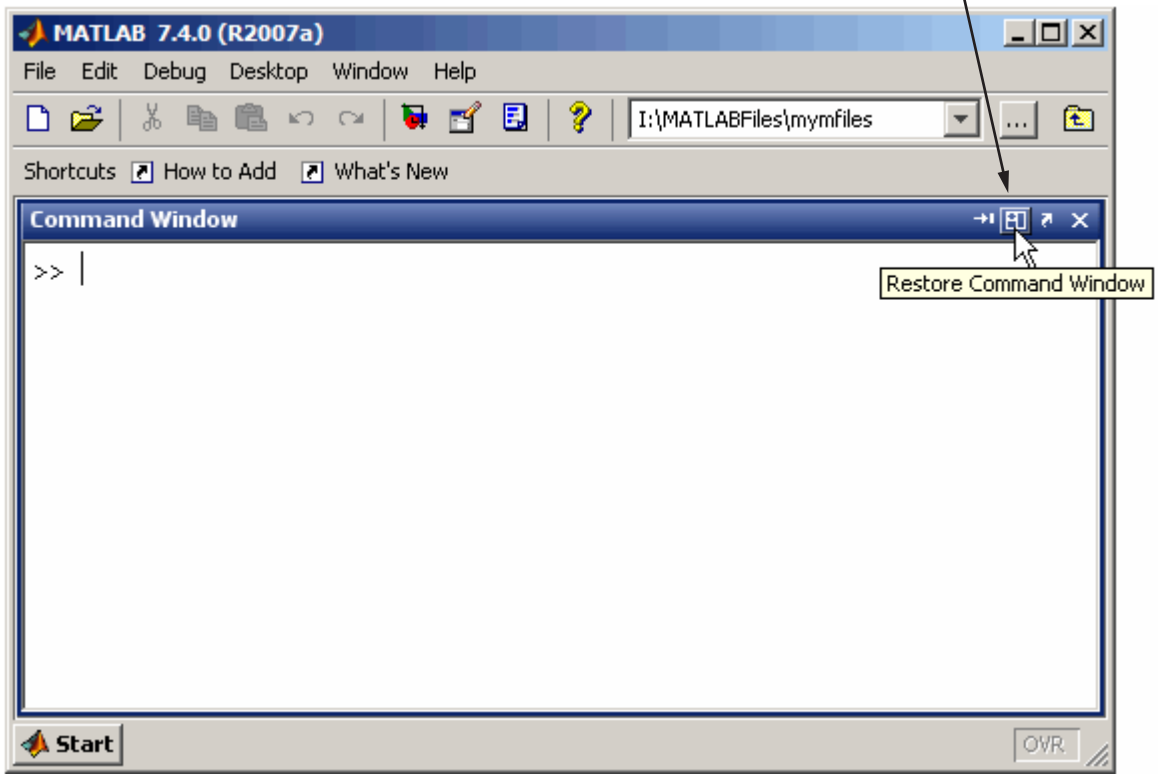
Maximized Tool in Desktop Example

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 is temporarily maximized by clicking the Maximize button in the Command Window title bar.



In this example, you return the maximized Command Window to its size and position in the desktop by clicking the Restore button in the title bar.

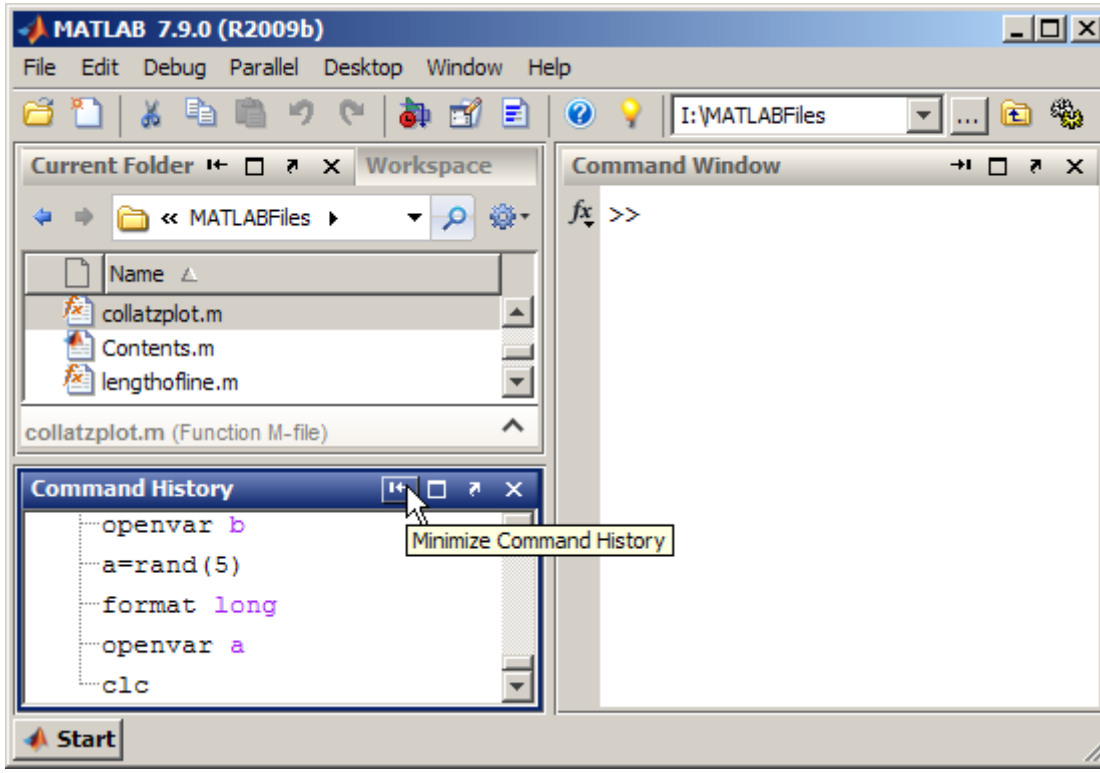
Maximized, the Command Window now occupies the full desktop area. Restoring the Command Window returns it to its original size and location in the desktop.



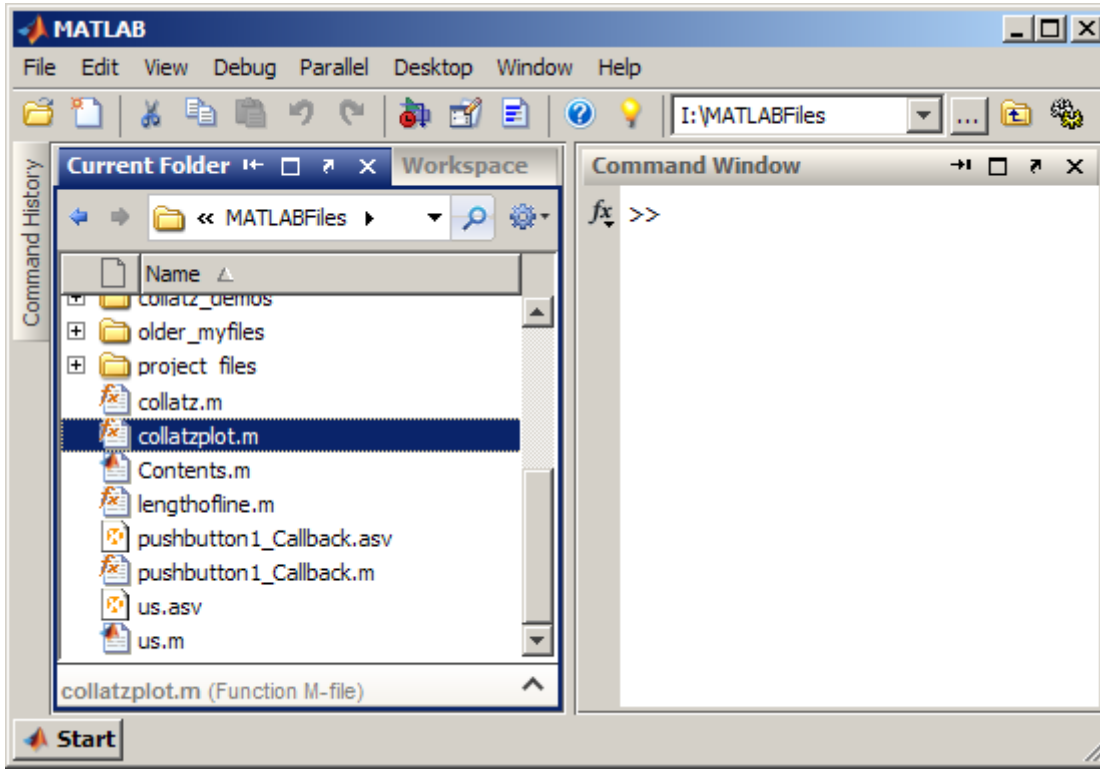
Minimized Tools in Desktop Example

Minimize a tool in the desktop to give the remaining desktop tools more space in the desktop. Minimizing is available on Microsoft Windows and UNIX[6] platforms. This illustration shows the button and associated tooltip for minimizing the Command History window to the left edge of the desktop.

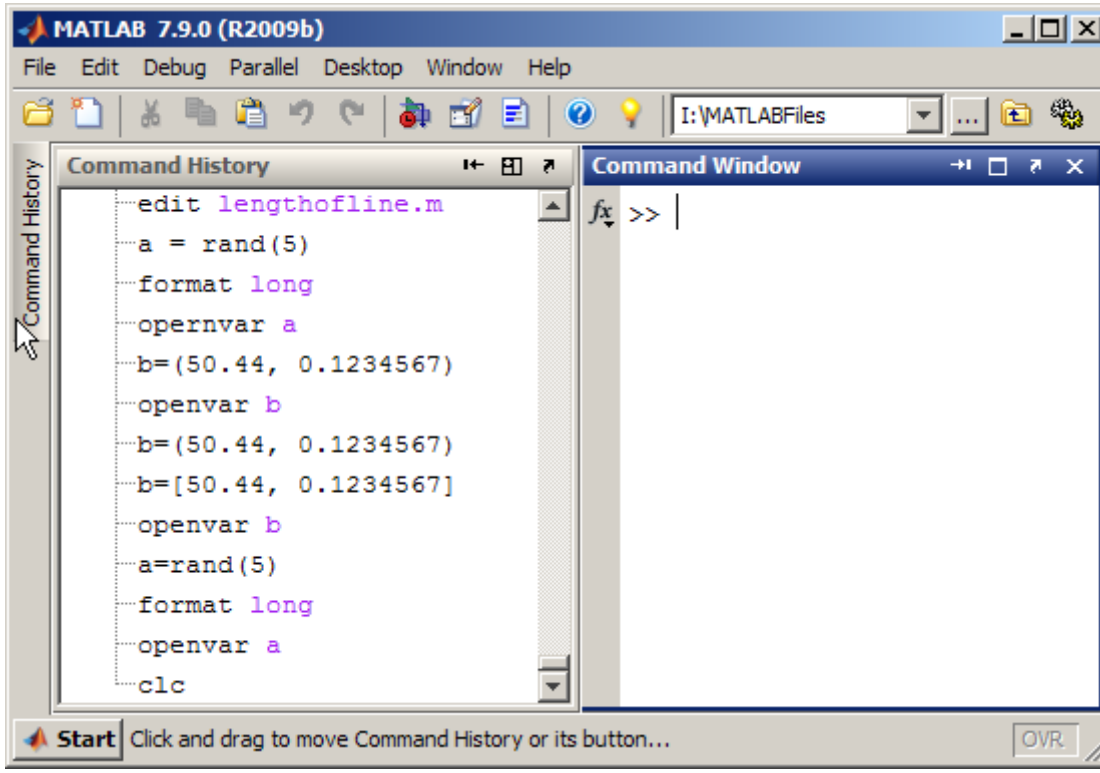
6. UNIX is a registered trademark of The Open Group in the United States and other countries.



This illustration shows the Command History minimized. It appears as a button along the left edge.



To view or use a minimized tool temporarily, hover over or click the button representing the minimized tool. MATLAB temporarily displays the tool. This illustration shows the minimized Command History temporarily open, as a result of hovering over the button.



When you select another tool, the tool on temporary display becomes minimized again.

To return the Command History to the position and size it occupied in the desktop before minimizing, do one of the following:

- Click the button representing the minimized tool, and then click the **Restore** button.
- Right-click the button representing the minimized tool, and then choose the **Restore** option.

Tiled Documents in Desktop Example

When you open a document (for example, an M-file), it also opens the tool (for example, the Editor) if the tool is not already open. Subsequent documents of the same type open in the tool and you can then arrange the documents within the tool, as follows:

- Accept the default to have documents appear one on top of another, such that the one on top hides the one or ones beneath it.

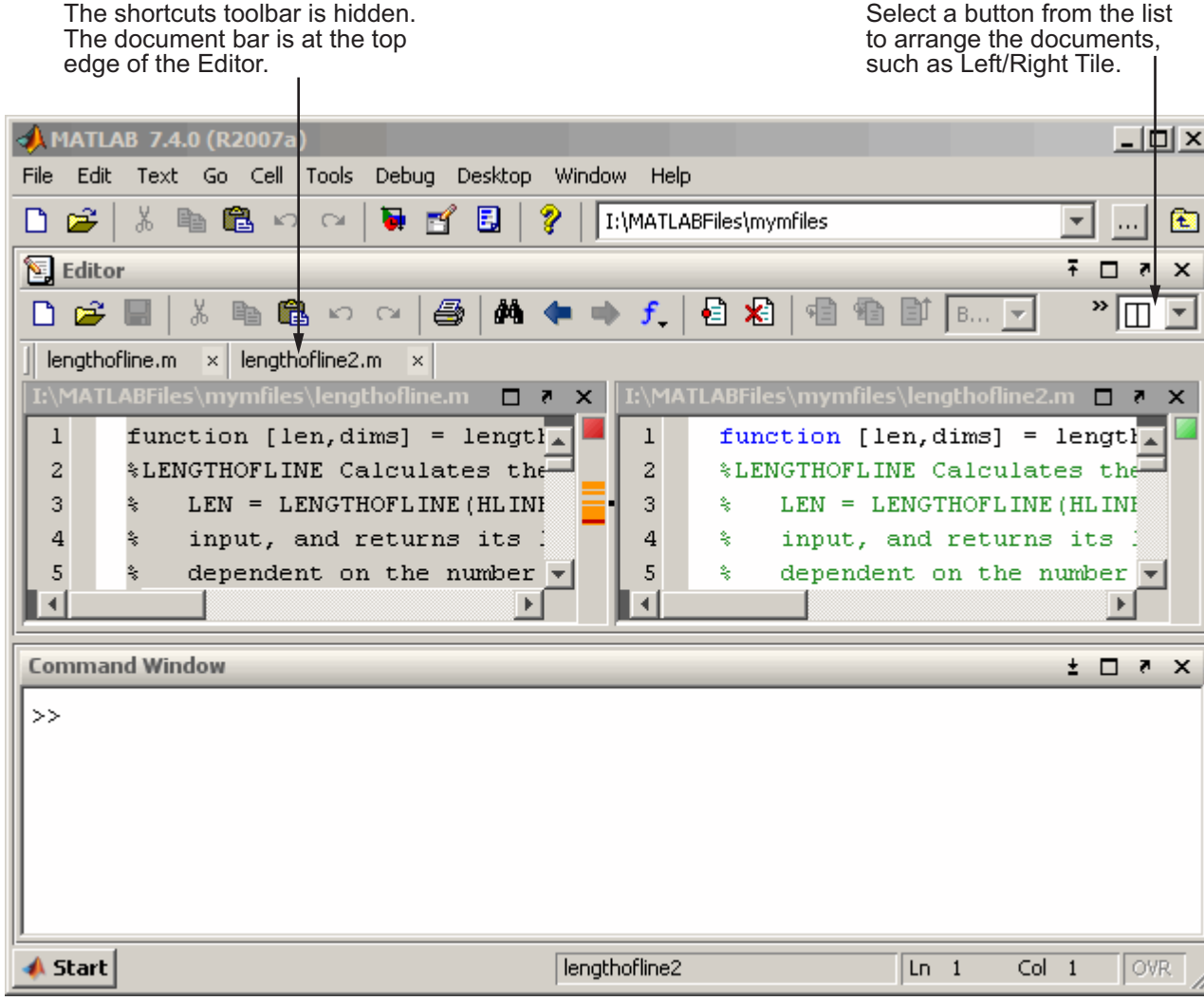
  After changing the arrangement from the default, you can restore it by selecting **Window > Maximize** (or the ❐ toolbar button).

- To arrange documents so that two documents display simultaneously, side-by-side, select **Window > Left/Right Tile** (or the ◫ toolbar button).

When tools and documents are docked, you can save space by hiding toolbars and document bars:

- To hide (or show) a toolbar, select **Desktop > Toolbar name**.
- To see or move the document bar, select **Desktop > Document Bar > Bar Position**, and choose its location, for example, **Top**.

The following example shows two M-files, side-by-side, with the desktop shortcuts toolbar hidden.


The shortcuts toolbar is hidden. The document bar is at the top edge of the Editor.
Select a button from the list to arrange the documents, such as Left/Right Tile.
MATLAB 7.4.0 (R2007a)
File Edit Text Go Cell Tools Debug Desktop Window Help
I:\MATLABFiles\mymfiles
Editor
lengthofline.m
lengthofline2.m
I:\MATLABFiles\mymfiles\lengthofline.m
I:\MATLABFiles\mymfiles\lengthofline2.m
function [len,dims] = lengt
%LENGTHOFLINE Calculates the
% LEN = LENGTHOFLINE(HLINE
% input, and returns its
% dependent on the number
Command Window
>>
Start
lengthofline2
Ln 1
Col 1
OVR

No Empty Document Tiles Example

The following example illustrates many of the options described in this list for creating and manipulating tiled documents.

- To see more than two documents at once:

  1 Click the Tile button.

  2 In the grid that appears, move the pointer 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.)

- To move a document to any empty tile, drag its title bar to the new location.
- To close an empty tile:

  1 Position the pointer over the handle on the separator bar.

  The handle becomes a Close box, as shown in the example.

  2 Click the Close box.

  The empty tile closes, and the neighboring document expands as shown in the following "After" illustration.

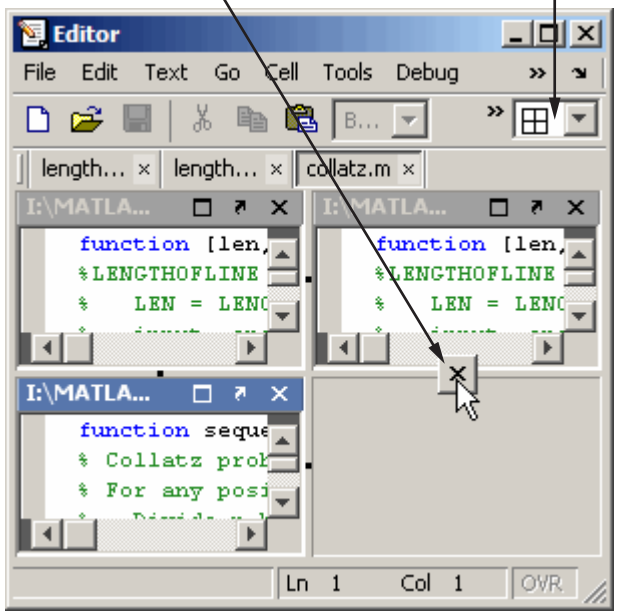
- To hide one document behind another, click the Close box between two tiles containing documents.

  One document becomes hidden.

Before

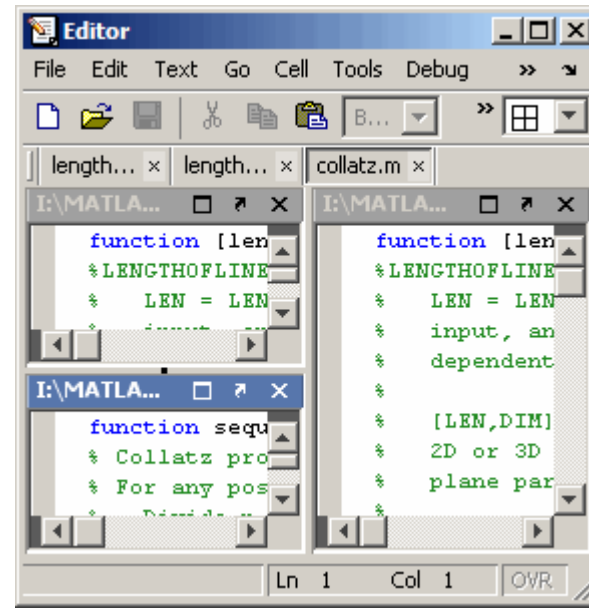
Close the empty tile using the handle on the separator bar.

Tile more than two documents with the Tile button.



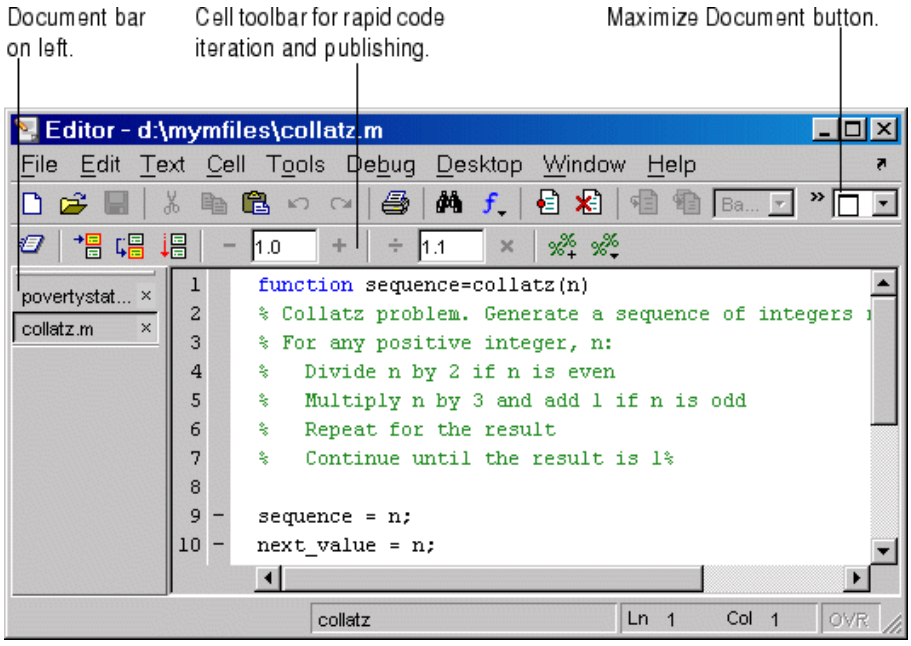
After

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.



Some common actions for working with documents outside of the desktop are

- Group all Editor documents — select **Desktop > Dock All in Editor** from any Editor document.
- Move all Editor documents outside of the desktop — select **Desktop > Undock Editor** when the Editor is the active window.
- Make a document occupy the full area in the Editor — click the Maximize button in the Editor 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.

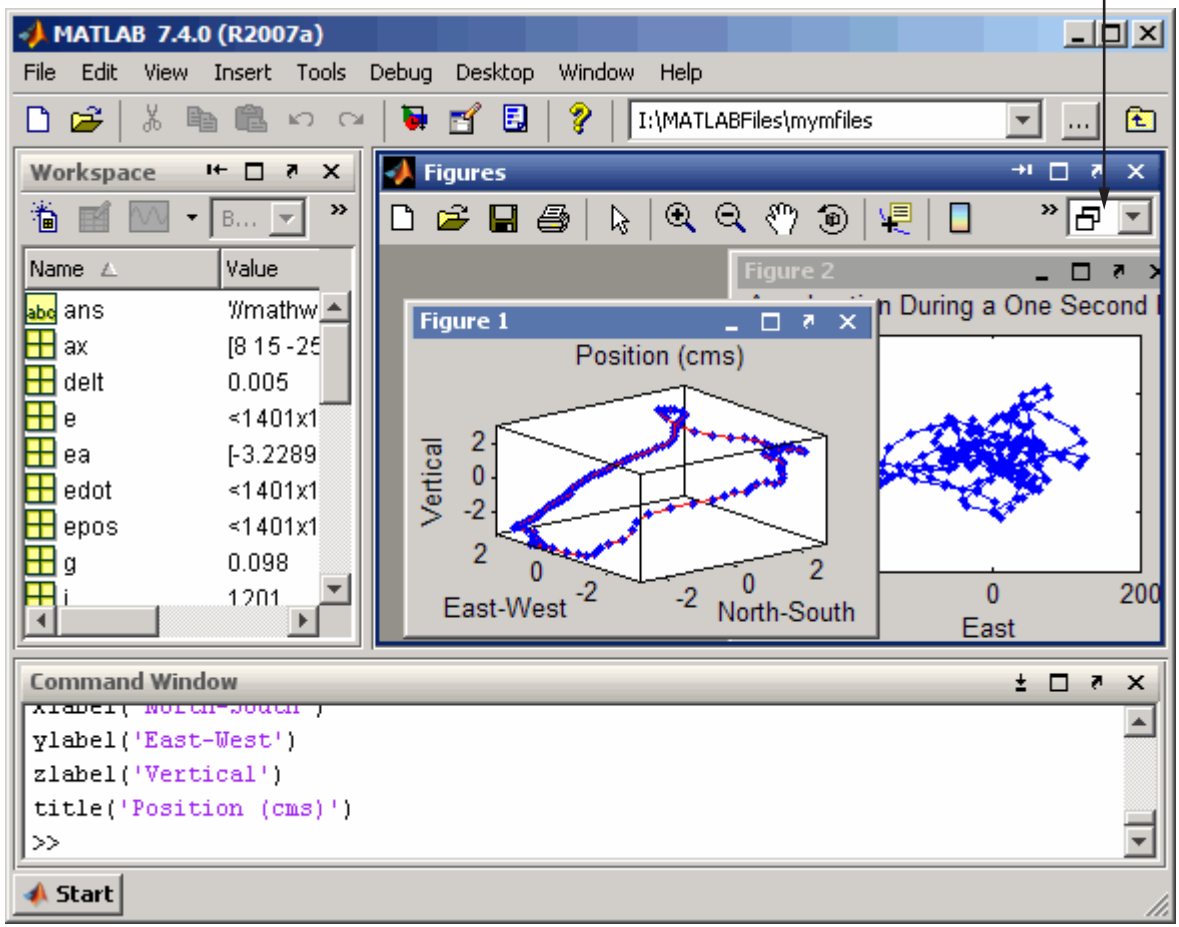
- Access any document in the Editor using the document bar. To show the document bar on the left side of the Editor, select **Desktop > Bar Position > Document Bar > Left** from the Editor.

Floating (Cascaded) Figures in Desktop Example

This example illustrates multiple figures in the desktop. By default, figures open outside the desktop. You can arrange and adjust the figures, as follows:

- To move the figures into the desktop, click the Dock button in the menu bar of each figure.
- To float (also called cascade) the figures, select **Window > Float**, or click the Float button 🗗.
- To get 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 as illustrated in the example that follows.

To undock a tool and its documents:

1. Select **Desktop > Undock Toolname**.
2. Select **Desktop > Undock Documentname** from the tool.

If you undock all documents from a tool, an empty tool window remains.

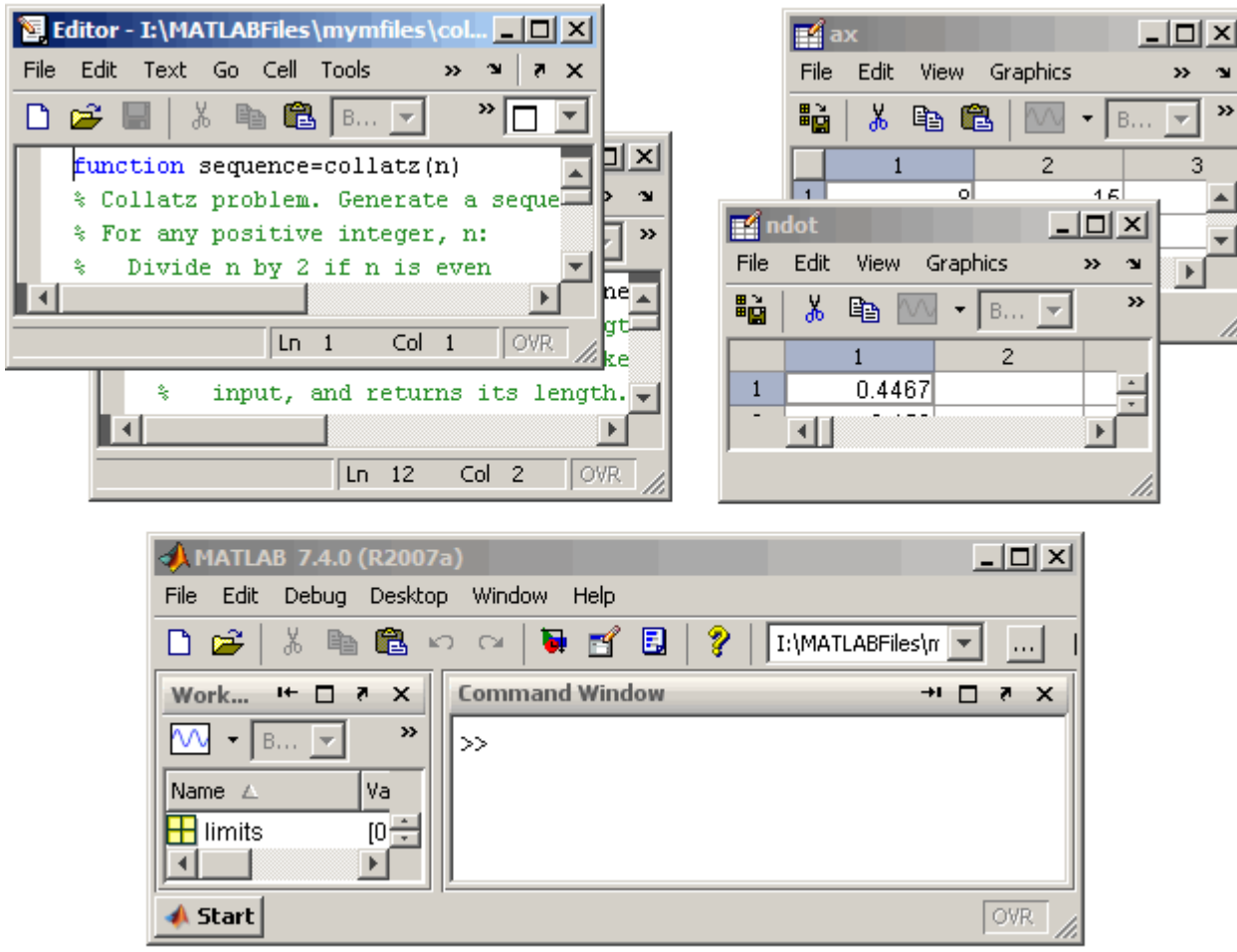
To close all undocked documents and their tools at once, select **Window > Close All Documents** from an undocked document window.

Notice the following in this example:

- One of the Editor documents, `collatz.m`, includes the name of the tool with it.
- The other Editor document, `lengthofline.m`, does not include the name of the tool with it.

  If you close the Editor, the `lengthofline.m` document remains open, but `collatz.m` closes.
- Neither of the Variable Editor documents includes the name of the tool.

  This is because the Variable Editor is undocked from the desktop, the variables are undocked from the Variable Editor, and the “empty” Variable Editor window is closed. The undocked documents in the tool remain open.


Editor - I:\MATLABFiles\mymfiles\col...
File Edit Text Go Cell Tools
function sequence=collatz(n)
% Collatz problem. Generate a seque
% For any positive integer, n:
% Divide n by 2 if n is even
Ln 1 Col 1 OVR
% input, and returns its length.
Ln 12 Col 2 OVR
ax
File Edit View Graphics
ndot
File Edit View Graphics
1 2
1 0.4467
MATLAB 7.4.0 (R2007a)
File Edit Debug Desktop Window Help
I:\MATLABFiles\m
Work...
Name Va
limits [0
Command Window
>>
Start
OVR

Running Frequently Used Statement Groups with MATLAB Shortcuts

In this section...

What Is a MATLAB Shortcut?

A MATLAB shortcut is an easy way to run a group of MATLAB language statements that you use regularly. Although like an M-file script, a shortcut is not an M-file. It need not be on the search path nor in the current folder when you run it, and MATLAB does not store it as an M-file.

You can create shortcuts that you run from the **Start** button or the shortcut toolbar, depending on your preferences.

MATLAB maintains shortcut information in the `shortcuts.xml` file. This file is located in the folder displayed when you type `prefdir` in the Command Window. It is unlikely that you will need to access this file, because MATLAB updates it automatically.

When to Use MATLAB Shortcuts

Create MATLAB shortcuts to:

- Run a group of functions you use frequently.

  For example, use a shortcut to set up your environment when you start working. This practice is useful when you do not use a `startup` file, or if there are statements you do not want to include in the `startup` file.

- Set the same properties for figures you create, such as adding a legend and setting the background color.
- Run a long statement, such as changing the current folder (`cd`) when the path names are long.
- Run a single function that you use frequently, such as `clc` to clear the Command Window.
- Run a statement that you use frequently, but have trouble remembering.

Creating MATLAB Shortcuts — Tutorials

You can create a MATLAB shortcut to run from the desktop **Start** button, or from the shortcuts toolbar, as described in the tutorials that follow. Both tutorials create a shortcut for a project called the Sea Temperature project. For these tutorials, you set up your environment in a certain way by running a series of MATLAB language statements. You create a shortcut called `sea_temp_env`, which contains these statements. Then, you run the shortcut to execute all the statements with a single click. The statements are:

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

Creating MATLAB Start Button Shortcuts

To create a start button shortcut, follow these steps:

1. From the **Start** button, select **Shortcuts > New Shortcut**.
2. Complete the Shortcut Editor dialog box.
   a. Provide a shortcut name in the **Label** field, for example, `sea_temp_environment`.
   b. Put the statements in the **Callback** field as shown in the following illustration. You can:

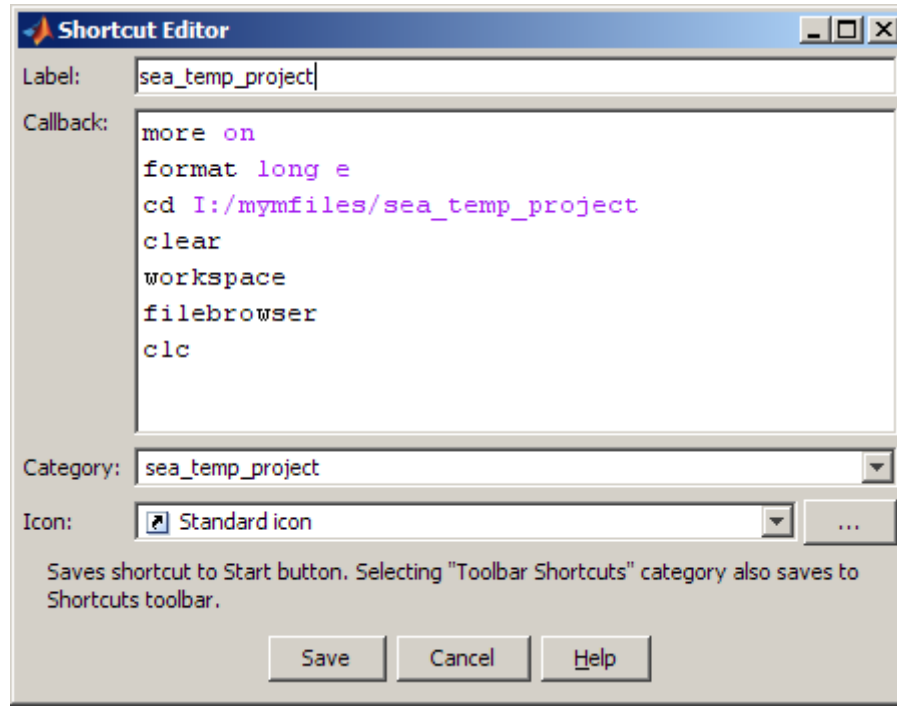
- Enter them by typing.
- Copy and paste them from a desktop tool—if prompts (>>) from the Command Window appear, MATLAB automatically removes them when you save the shortcut.
- Drag them from a desktop tool.
- Edit them, if necessary.

The **Callback** field uses the Editor preferences for keyboard shortcuts, colors, and fonts.

c Assign a **Category**, which is like a folder for organizing shortcuts. For this example, specify `sea_temp_project`.

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

e Click **Save**.



MATLAB adds the shortcut to the **Shortcuts** entry in the **Start** button.

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

Creating MATLAB Toolbar Shortcuts

This example assumes that you have the following statements in the Command Window:

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

Follow these steps:

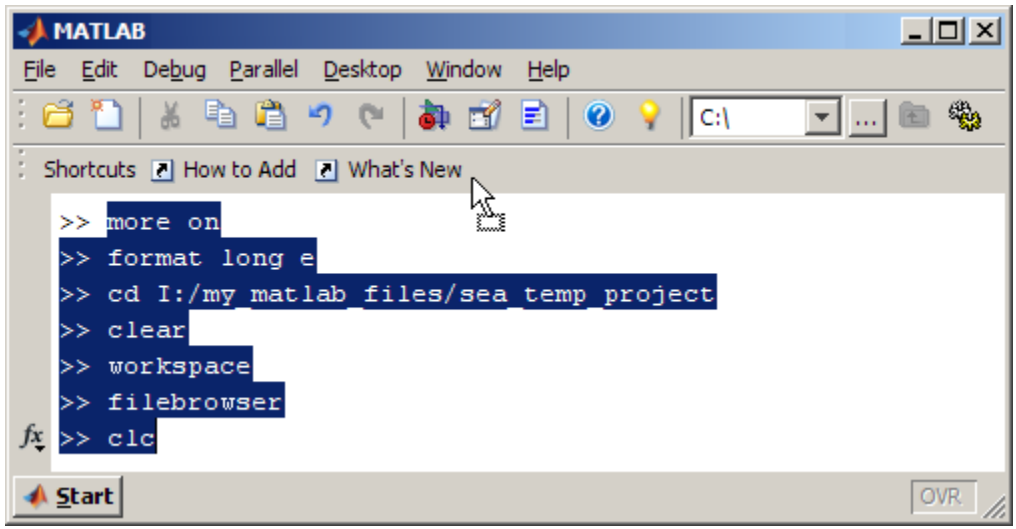
1. If the shortcuts toolbar is not currently on the desktop, choose **Desktop > Toolbars > Shortcuts**.

2. Select statements from the Command Window.

   You can also select statements from the Command History Window or an M-file.

3. Drag the selection to the desktop **Shortcuts** toolbar.

   This illustration shows highlighted statements being dragged from the Command Window to the toolbar.



4 Create the shortcut by completing the Shortcut Editor dialog box:

   a In the **Label** field, enter a name for the shortcut.

   b In the **Callback** field, edit the selected statements, if necessary.

      If prompts (>>) from the Command Window appear, MATLAB automatically removes them when you save the shortcut.

   c Do not change the **Category** field value, `Toolbar Shortcuts`.

      For the shortcut to appear on the toolbar, this value must remain as-is.

   d In the **Icon** field, select an icon, or keep the default.

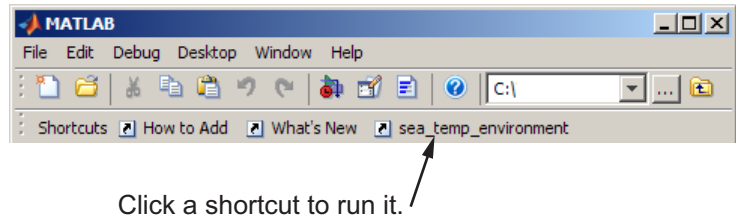
   e Click **Save**.

   The shortcut icon and label appear on the toolbar. If you have more shortcuts on the toolbar than the desktop can display concurrently, use the drop-down list to access them all.

Running MATLAB Shortcuts

To run a shortcut, do one of the following:

- To run a Start menu shortcut, select **Start > Shortcuts**. Then select the shortcut name, or a category submenu, followed by the shortcut name.
- To run a toolbar shortcut, click its icon on the shortcuts toolbar.



All the statements in the shortcut **Callback** field execute. It is as if you ran those statements from the Command Window, although they do not appear in the Command History window.

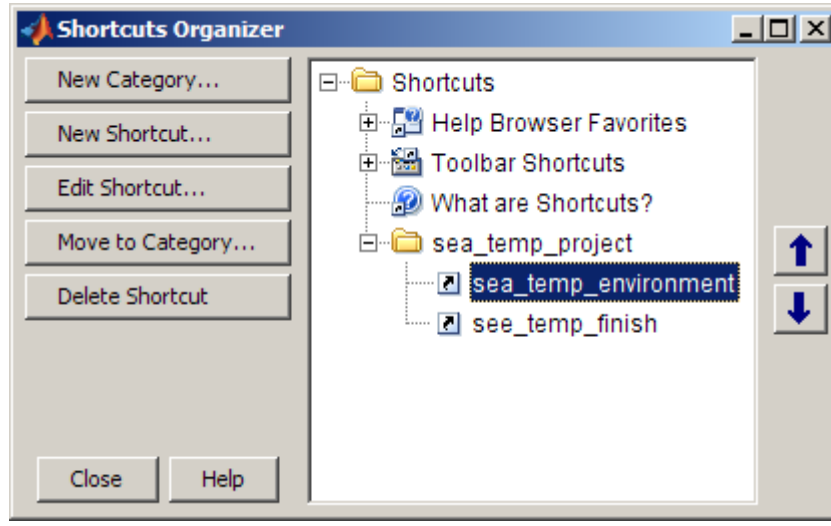
Editing and Organizing MATLAB Shortcuts

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

1 Open the Shortcuts Organizer dialog box, by doing one of the following:

- Click the **Start** button and select **Shortcuts > Organize Shortcuts**.
- From the shortcuts toolbar context menu, choose **Organize Shortcuts**.

The Shortcuts Organizer dialog box appears. When you select a shortcut category in the dialog box, the **Edit Shortcut** button replaces the **Rename Category** button.



2 To edit and organize shortcuts and categories, do one of the following:

- Click buttons in the dialog box.
- 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.

Customizing MATLAB Toolbar Shortcuts

For step-by-step instructions on how to add a shortcut to the toolbar, see “Creating MATLAB Toolbar Shortcuts” on page 2-62.

Default Toolbar Shortcuts

By default, the **Shortcuts** toolbar includes these two shortcuts:

- **How to Add**—Provides help about shortcuts and adding them to the **Shortcuts** toolbar.

- **What's New** —Displays the Release Notes documentation.

If you want to remove one or both of these shortcuts, see "Deleting MATLAB Shortcuts from the Toolbar" on page 2-66.

Hiding MATLAB Shortcut Labels on the Toolbar

To hide the shortcut labels on the toolbar, and leave just the shortcut icon:

1. Right-click in the **Shortcuts** toolbar.
2. From the context menu, select **Show Labels**, to clear the check mark next to the item.

   Now, when you move the mouse over a shortcut icon, its label appears as a tooltip.

Displaying Hidden MATLAB Shortcut Labels on the Toolbar

To redisplay a shortcut label that you previously hid:

1. Right-click on the **Shortcuts** toolbar.
2. From the context menu, check the **Show Labels** item by selecting it. The label reappears on the toolbar.

Deleting MATLAB Shortcuts from the Toolbar

To remove a shortcut:

1. Right-click the toolbar shortcut button.
2. From the context menu. select **Delete**.
3. Click **OK** in the confirmation dialog box.

Alternative Ways to Create MATLAB Shortcuts

In addition to the previously described ways to create shortcuts, you can do any of the following:

- From the Command History window, select MATLAB language statements, right-click, and select **Create Shortcut** from the context menu. By default, MATLAB assigns shortcuts created from the Command History window to the `Toolbar Shortcuts` category. Shortcuts in the `Toolbar Shortcuts` category appear on the **Shortcuts** toolbar.
- Drag statements from a desktop tool, such as the Command History, onto the **Start** button.
- Right-click the **Shortcuts** toolbar, and select **New Shortcut**. Complete the resulting Shortcut Editor dialog box. If you maintain the `Toolbar Shortcuts` category, the shortcut appears on the shortcuts toolbar.

Performing Desktop Actions Using the Keyboard

Keyboard Key Combinations

As an alternative to using the mouse, you can press a combination of keyboard keys to perform some desktop actions. MATLAB supports the use of both mnemonics and keyboard shortcuts. The following topics explain the differences between these two methods and how to use them:

What Is a Mnemonic?

A *mnemonic* is a means of using keystrokes to perform a desktop action, such as clicking a button or opening a menu, and then choosing an option. It is called a mnemonic because it frequently uses the first letter of the menu or menu option name to help you remember the keystrokes required to use it.

Mnemonics appear as underlined letters on menus or buttons. For instance, the F in the MATLAB **File** menu appears as shown in the following image.

Using Mnemonics

To open a menu or activate a button using mnemonics, press the **Alt** key and the letter key that corresponds to the underlined letter in the menu name, menu option name, or button name. The action occurs when you press the letter. You also can use mnemonics to perform an action that would require multiple mouse clicks. For example, opening the print dialog box: **Alt**+**F**, **P**. For more information see "Examples of Mnemonics and a Keyboard Shortcut" on page 2-70.

Customized keyboard shortcuts can override mnemonics. For example, if you specify **Alt+F**, **P** as the keyboard shortcut for the Delete action across the desktop, then pressing **Alt+F**, **P** no longer opens the Print dialog box. You cannot customize mnemonics.

Platform Differences.

- MATLAB running on the Apple Macintosh platform does not support mnemonics.
- The Windows operating system has a setting to hide the display of mnemonics. To display hidden mnemonics, make the MATLAB desktop the active window, and then press the **Alt** key. For details, see the Windows documentation.

What Is a Keyboard Shortcut?

A *keyboard shortcut* is a means of using keyboard key strokes to perform a desktop action, without opening a desktop menu. If a keyboard shortcut is assigned to an action on a menu option, it appears to the right of the option name on the menu. For example, the default keyboard shortcut for opening the Open dialog box is **Ctrl+O**.

It is possible to have multiple keyboard shortcuts for an action. All defined shortcuts work, but only one appears next to the option name on the desktop menu. When you provide a keyboard shortcut for a menu option that previously had none, the desktop menu automatically updates to display the new keyboard shortcut.

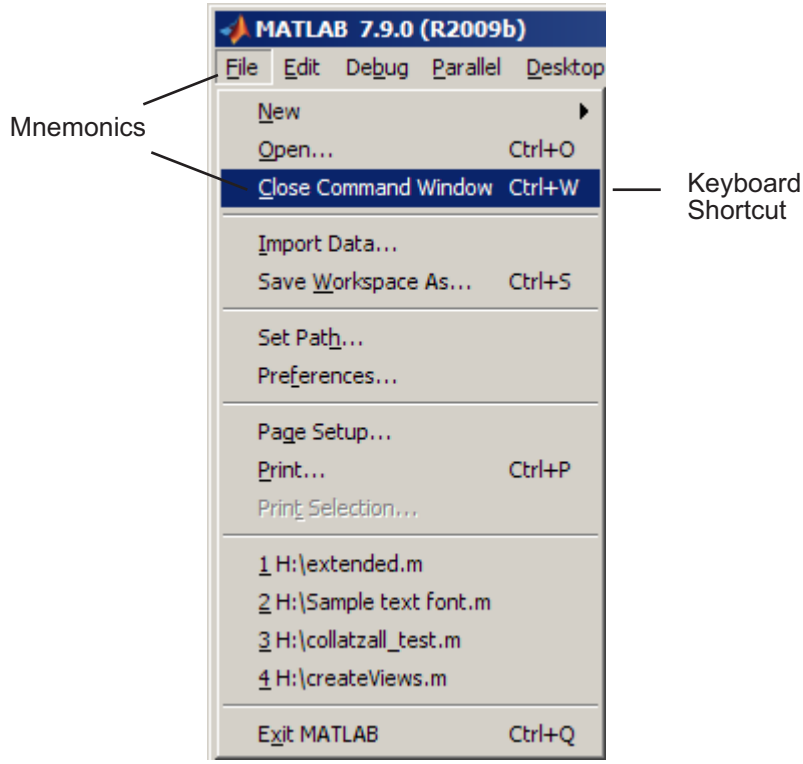
If you define a keyboard shortcut that uses the same keystrokes as a mnemonic but performs a different action, then the mnemonic no longer works.

You can choose from a set of shortcuts that install with MATLAB, create customized sets, or use a set copied from another system. For more

information see “Performing Desktop Actions Using the Keyboard” on page 2-68

Examples of Mnemonics and a Keyboard Shortcut

The image that follows shows the desktop File menu. Notice the mnemonic and keyboard shortcut for closing the Command Window.



- **Mnemonics**

  In the illustration, the underlined F in **File** on the menu bar and the underlined C in the **Close Command Window** option name indicate the mnemonics. You can close the Command Window without using the mouse by pressing **Alt+F**, **C**.

- **Keyboard Shortcut**

In the illustration, **Ctrl+W**, to the right of the**Close Command Window** menu option indicates the keyboard shortcut. You can close the Command Window without opening the **File** menu by pressing **Ctrl+W**.

Performing Desktop Actions Using Keyboard Shortcuts

In this section...

Overview

A *keyboard shortcut* is a means of using keyboard key strokes to perform a desktop action, without opening a desktop menu. If a keyboard shortcut is available for a menu option, it appears to the right of the option name on the menu. For example, the default keyboard shortcut for opening the Open dialog box is **Ctrl+O**.

It is possible to have multiple keyboard shortcuts for an action. All defined shortcuts work, but only one appears next to the option name on the desktop menu. When you provide a keyboard shortcut for a menu option that previously had none, the desktop menu automatically updates to display the keyboard shortcut.

If you define a keyboard shortcut that uses the same keystrokes as a mnemonic but performs a different action, then the mnemonic no longer works.

You can choose from a set of shortcuts that install with MATLAB, create custom shortcut sets, or use shortcut sets copied from other users or systems. For more information see "Performing Desktop Actions Using the Keyboard" on page 2-68

Choosing a Set of Keyboard Shortcuts

You choose a set of keyboard shortcuts to use from the **Active settings** drop-down list in the Keyboard Shortcuts preferences dialog box. By default, MATLAB uses a settings file that corresponds to the platform on which you are running. Use the default keyboard shortcuts set or use a different set to:

- Modify keyboard shortcuts. See "Steps for Customizing Keyboard Shortcuts" on page 2-79.
- Create a new settings file. See "Saving Keyboard Shortcuts to a Settings File" on page 2-81.
- Use a settings file that does not install with MATLAB. See "Using Keyboard Shortcuts Settings Files Created on Other Systems" on page 2-89.

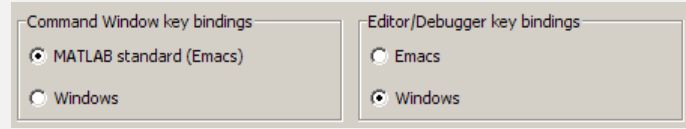
To choose a keyboard shortcuts settings file, follow these steps:

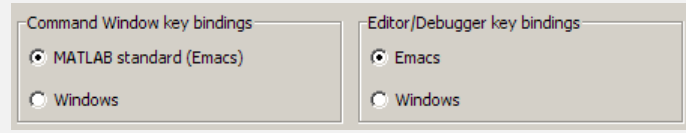
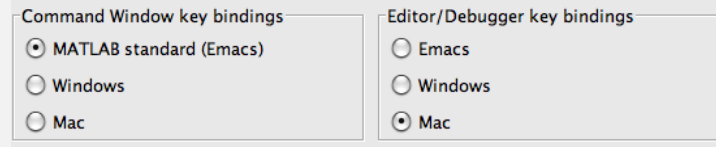
1. Open the Keyboard Shortcuts Preferences dialog box by choosing **File > Preferences > Keyboard > Shortcuts**.
2. Click the down arrow in the **Active settings** field, and then choose a settings file that you want to use.

   If a file that you want to use is on your system, but does not appear in the drop-down list, click **Browse** to find it.
3. Click **Apply**.

Keyboard Shortcuts Settings Files

The following table lists the keyboard shortcuts settings files installed with MATLAB.

Operating System	Keyboard shortcut settings files installed with MATLAB
Windows	• `Windows Default Set` (Default) Specifies MATLAB keyboard shortcuts for the Windows operating system across the desktop. • `Emacs Default Set` Specifies Emacs keyboard shortcuts across the MATLAB desktop. • `R2009a Windows Default Set` Restores the MATLAB default keyboard shortcuts that were in place for the Windows platform in Version 7.9 (R2009a) and earlier releases. In those releases, this set of keyboard shortcuts appeared on the **Keyboard Preferences** pane as a combination of Command Window key bindings and Editor/Debugger key bindings, as shown in the following image. 
UNIX	• `Emacs Default Set` (Default) Specifies Emacs keyboard shortcuts across the MATLAB desktop. • `Windows Default Set` Specifies MATLAB keyboard shortcuts for the Windows operating system across the desktop. • `R2009a UNIX Default Set` Restores the MATLAB default keyboard shortcuts that were in place for the UNIX platforms in Version 7.9 (R2009a) and earlier releases (not including the Macintosh platform).

Operating System	Keyboard shortcut settings files installed with MATLAB
	In those releases, this set of keyboard shortcuts appeared on the **Keyboard Preferences** pane as a combination of Command Window key bindings and Editor/Debugger key bindings, as shown in the following image. 
Macintosh	• `Macintosh Default Set` (Default) Specifies MATLAB keyboard shortcuts for the Macintosh operating system across the desktop. • `R2009a Macintosh Default Set` Restores the MATLAB default keyboard shortcuts that were in place for the Macintosh platform in Version 7.9 (R2009a) and earlier releases. In those releases, this set of keyboard shortcuts appeared on the **Keyboard Preferences** pane as a combination of Command Window key bindings and Editor/Debugger key bindings, as shown in the following image. 

Identifying Keyboard Shortcuts

To determine the current keyboard shortcut for an action, view one of the following:

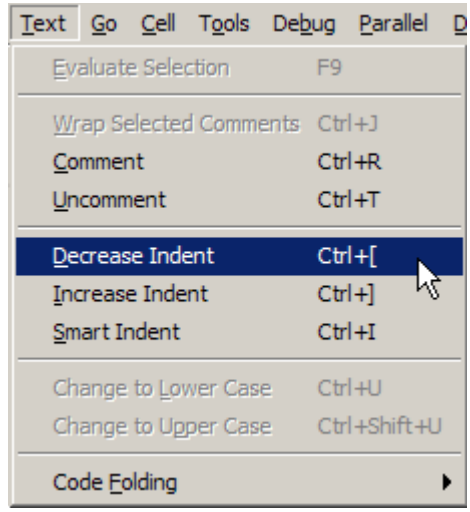
- “Keyboard Shortcuts on Menus” on page 2-76

- "Keyboard Shortcuts on the Keyboard Shortcuts Preferences Pane" on page 2-76

Keyboard Shortcuts on Menus

Open the menu to see if the keyboard shortcut appears next to the menu option.

For example, suppose you want to determine the keyboard shortcut for decreasing the indent in the Editor. Open the **Text** menu, and then view the keyboard shortcut for **Decrease Indent**. The keyboard shortcut **Ctrl+[** appears to the right of the option.



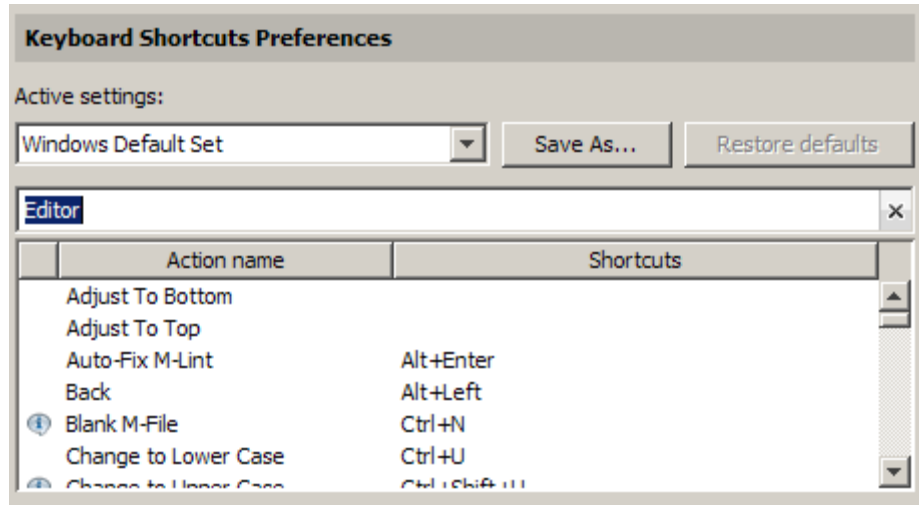
If no keyboard shortcut appears on the menu, one does not currently exist for that action. To create a keyboard shortcut for an action, follow the steps in "Customizing Keyboard Shortcuts" on page 2-78.

Keyboard Shortcuts on the Keyboard Shortcuts Preferences Pane

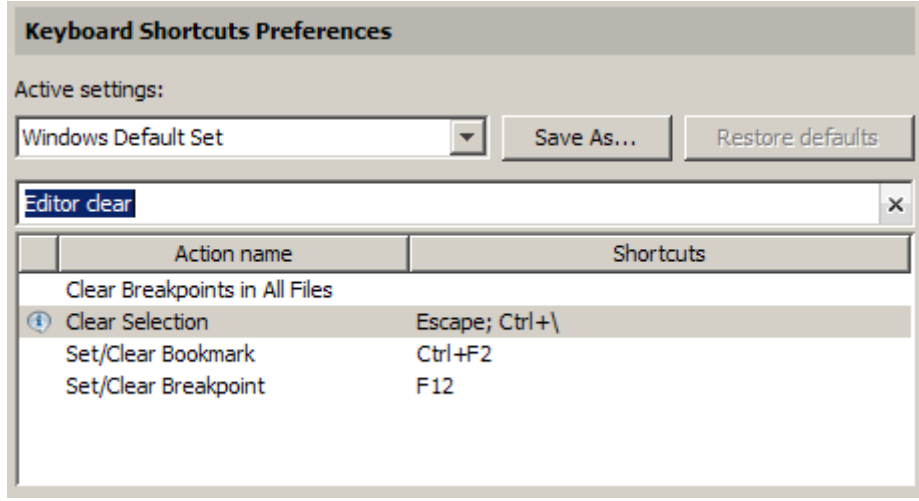
To identify a keyboard shortcut when there is no menu option for an action, use the **Keyboard Shortcuts Preferences** pane:

1 Choose **File > Preferences > Keyboard > Shortcuts**.

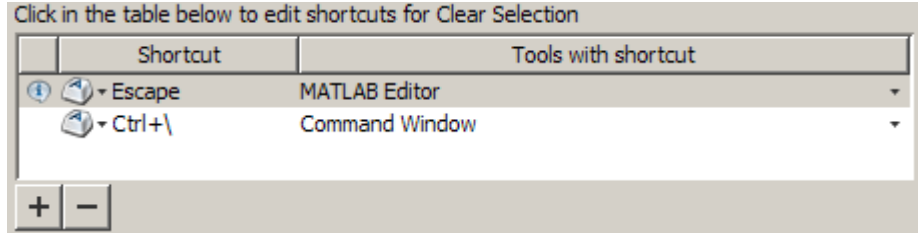
2 In the filter field, type the name of the tool for which you want to list the keyboard shortcuts. For example, type `Editor` to see the keyboard shortcuts currently defined for actions you can perform in the Editor.



3 Narrow the list of **Action names** that the preferences pane displays by typing a string describing the action. For example, type `clear`, if you want to find the keyboard shortcut for clearing selected text in the Editor. Type a short string to increase the likelihood of the filter returning the action you seek.



4 View the table that appears in the middle of the **Keyboard Shortcuts Preferences** pane. For example, this table indicates that the **Escape** key is the current keyboard shortcut for the **Clear Selection** action in the Editor.



Customizing Keyboard Shortcuts

To customize or view keyboard shortcuts for MATLAB desktop tools, choose **File > Preferences > Keyboard > Shortcuts**. If you have an active Internet connection, you can watch the Customizable Keyboard Shortcuts video for an overview.

The following sections provide details:

- “Steps for Customizing Keyboard Shortcuts” on page 2-79

- "Filtering Keyboard Shortcut Actions" on page 2-82
- "Specifying Keystrokes for a Keyboard Shortcut" on page 2-83
- "Evaluating and Resolving Keyboard Shortcut Conflicts" on page 2-84
- "Examples of Creating, Modifying, and Deleting Keyboard Shortcuts" on page 2-86
- "Keyboard Key Combinations" on page 2-68
- "Identifying Keyboard Shortcuts" on page 2-75

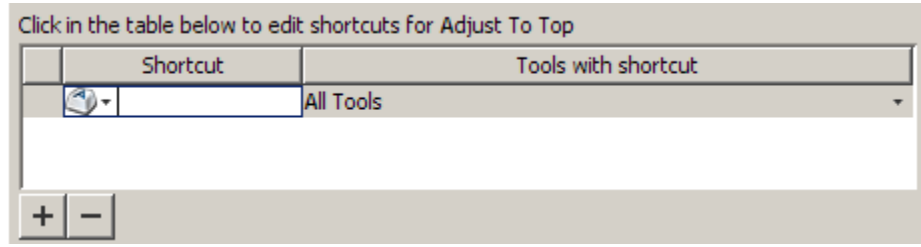
Steps for Customizing Keyboard Shortcuts

1. Choose **File > Preferences > Keyboard > Shortcuts**.
2. In the **Active settings** field, choose the file that contains the set of keyboard shortcuts that you want to customize.

   Typically, the first time you modify keyboard shortcuts, you begin with the default settings for your platform. For details, see "Choosing a Set of Keyboard Shortcuts" on page 2-73.
3. Under **Action name**, select the action for which you want to define or modify a keyboard shortcut. An action is the operation for which you want to customize the shortcut, such as `Clear Command History`.

   For tips on finding the action you want, see "Filtering Keyboard Shortcut Actions" on page 2-82.
4. Click the Add button +.

   An editable field opens under the **Shortcut** column.



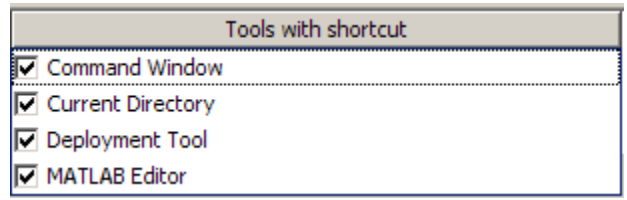
5 Type the shortcut that you want to use for the action you selected in Step 3. Alternatively, you can choose a shortcut from the drop-down menu.

   For details, see "Specifying Keystrokes for a Keyboard Shortcut" on page 2-83.

6 Assign the shortcut to the tool or tools with which you want to use it. For example, in the **Tools with shortcut** column:

   a Click the down arrow for the list of desktop tools to which you can assign a shortcut. Not all actions are available with all desktop tools.

   b Select a check box to assign the shortcut to a tool. Clear a check box to remove it.



7 Evaluate and resolve conflicts, indicated by the informational and error icons, if necessary.

   For more information, see "Evaluating and Resolving Keyboard Shortcut Conflicts" on page 2-84.

8 Click **Apply**.

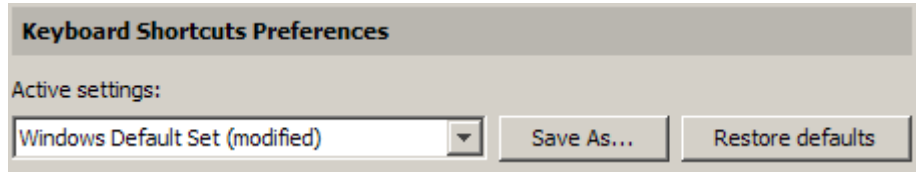
   - The keyboard shortcut becomes available immediately.
   - If you changed a shortcut that corresponds to a menu option that previously did not display a keyboard shortcut, MATLAB reflects the new keyboard shortcut in the menu.

Restoring Default Sets

If you modify keyboard shortcuts, and then decide you do not want to keep the changes, click **Restore defaults**. Be aware that clicking **Restore defaults** reverts all keyboard shortcuts changes you have made since you last saved the set. If you modify keyboard shortcuts in a set that installs with MATLAB,

such as `Windows Default`, you lose all previous modifications. You cannot revert changes on a shortcut-by-shortcut basis.

The following image shows the **Active settings** field displaying a modified Windows Default set of keyboard shortcuts and the **Restore defaults** button.



Saving Keyboard Shortcuts to a Settings File

Save a settings file to:

- Save changes you make to a default settings file, such as the `Windows default` set, to a new set.

  Although changes you make to the default sets are preserved across MATLAB sessions, you lose all changes if you click the **Restore defaults** button and you have not previously saved the to a new set.
- Copy it to another system running MATLABand use it there.
- Overwrite a file that you previously saved.

  You cannot overwrite the default settings files that install with MATLAB. MATLAB saves modifications that you make to a default set using the name of the default set appended with the text `(modified)`. For instance, `Windows default (modified)`.
- Share it with others.

  For information on sharing your active settings file with the MATLAB community, see “Submitting Your Files to the Repository” on page 7-41.

To save a keyboard shortcuts settings file, follow these steps:

1. Open the Keyboard Shortcuts Preferences dialog box by choosing **File > Preferences > Keyboard > Shortcuts**.
2. Click the **Save As** button.

3 Navigate to the folder in which you want to save the file, specify the file name, and then click **Save**.

MATLAB saves the file as an `.xml` file in the folder that the Command Window returns when you type `prefdir`.

Filtering Keyboard Shortcut Actions

Use the filter field to see the list of actions for which you can customize or define a keyboard shortcut as follows:

1 Type all or part of any one of the following:

- An action name, for example, `Delete`.

  MATLAB displays only the action names or desktop menus that contain the text you specify.

- The name of a desktop tool or menu, for example, `File` or `Command Window`.

  MATLAB displays the action names associated with the tool or menu you specify. In addition, the list of action names includes any action names that contain the name of the tool or menu. For example, if you specify `Command History`, the list of action names includes `Next History Command`, which is a Command Window action.

- A keyboard shortcut, for example, `Ctrl+R`

  MATLAB displays only the action names that have the shortcut you specify. Be aware of the following:

  - You can enter most keyboard shortcuts by either pressing the keystrokes or typing the key names.

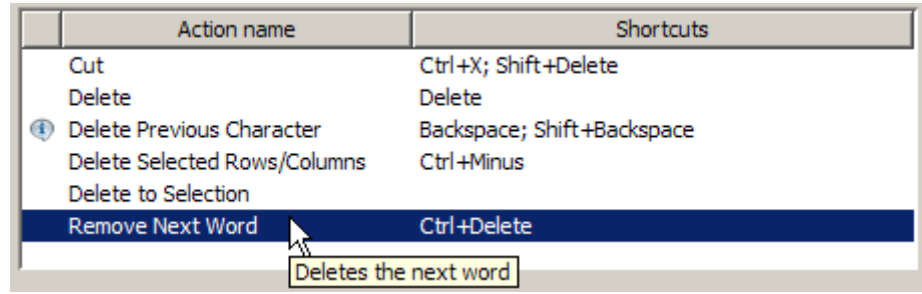
    To specify **Ctrl+S**, for example, you can type `Ctrl+S` character-by-character, or you can press the keyboard shortcut, **Ctrl+S**.

  - If pressing the keystrokes for a keyboard shortcut does not work, try typing it instead. Some examples of keyboard shortcuts you *must* type character-by-character are **Tab**, **Backspace**, and **Delete**.

  - Use `NumPad` to refer to the number pad that is on the far right of some keyboards.

- Use `Up` and `Down` to refer to the **Up arrow** and **Down arrow** keypad keys.

2. Verify that an **Action name** performs the action you expect:
   a. Hover the mouse pointer over the **Action name**. For example, **Remove Next Word**.
   b. View the tooltip that appears.



Specifying Keystrokes for a Keyboard Shortcut

Specify the keystrokes for a keyboard shortcut as follows:

1. Click the **Add** button 
2. Specify the number of keystrokes you want to use for the shortcut.
   - To use the default, which is one keystroke, skip to step 3.
   - To specify multiple keystrokes, or to specify explicitly one keystroke follow these steps:
     1. Click the down arrow next to the key icon  in the **Shortcuts** field.
     2. Choose `Limit to 1 keystroke`, `Limit to 2 keystrokes`, or `Limit to 3 keystrokes`.

        For instance, **Ctrl+F** is one keystroke, **Ctrl+Y**, **Shift+Z** is two keystrokes, and **Ctrl+Y**, **Shift+Z**, **F9** is three keystrokes.

3. Specify the keystrokes, by doing one of the following:

- Type the keystrokes, by pressing the keys, not spelling the key names.

  For example, press the **Ctrl** key and the **Y** key. Do not type **C-t-r-l-+-Y**.

- Choose a keystroke such as the **Tab** key, by clicking the down arrow next to the key icon in the **Shortcuts** field. Then, choose the key name.

  The listed keys are those keys that already have a defined action within a dialog box. For example the **Tab** key navigates from one field to the next in a dialog box.

Evaluating and Resolving Keyboard Shortcut Conflicts

Conflicts arise when the same shortcut is assigned to two or more different actions. There is no requirement that you resolve keyboard shortcut conflicts. However, if the same shortcut specifies two different actions, the shortcuts can be confusing to use.

View keyboard shortcut conflicts by choosing **File > Preferences > Keyboard > Shortcuts**.

The Keyboard Shortcuts preferences pane indicates conflicts using informational and error icons.

- —An informational icon indicates that two different actions in two different tools have the same shortcut. For information on resolving these conflicts, see “Actions in Different Tools Have the Same Shortcut — Evaluating Conflicts” on page 2-84.
- —An error icon indicates that two different actions within the same tool have the same shortcut. For information on resolving these conflicts, see “Actions in the Same Tool Have the Same Shortcut — Evaluating Conflicts” on page 2-85.

Actions in Different Tools Have the Same Shortcut — Evaluating Conflicts

Typically, you want to resolve conflicts indicated by the informational icon when all the following are true:

- You use both tools frequently.
- You perform both actions frequently.

- You have difficulty remembering the action that the shortcut performs in each tool.

For instance on Microsoft Windows platforms, by default, **Ctrl+Shift+U** undocks a tool from the MATLAB desktop. However if you select text in the Editor, and then press **Ctrl+Shift+U**, it changes the selected text to uppercase. If you frequently use both of these actions, you can specify a different keyboard shortcut for one or both actions.

Actions in the Same Tool Have the Same Shortcut — Evaluating Conflicts

Typically, you want to resolve conflicts indicated by the error icon  .

It can be *unnecessary* to resolve these conflicts if one or more of the following are true:

- The situation is temporary

  For instance, you are performing a two-step procedure. In the first step, you assign the keyboard shortcut to an action that results in a conflict. Then, in the second step, you remove the shortcut from the original action.

- The two actions are associated with different modes of the same tool.

  By default, when the MATLAB Editor is in cell mode, **Ctrl+Up** and **Ctrl+Down** move the cursor to the Next and Previous cell, respectively. When the Editor is not in cell mode, those keyboard shortcuts scroll up and scroll down, respectively. The shortcuts are in conflict, but the behavior probably is expected, for the given MATLAB Editor mode.

  Similarly, although not evident from the preferences pane, on Windows systems, the keyboard shortcut for interrupting MATLAB execution is **Ctrl+C**. However, the default keyboard shortcut for the copy action is also **Ctrl+C**. Under these conditions, if you:

  - Select an item, and then press **Ctrl+C**, it copies the selected item to the clipboard, — regardless of whether MATLAB is busy
  - Do not select an item and press **Ctrl+C**, it interrupts MATLAB execution

  If you change the default keyboard shortcut for the copy action from **Ctrl+C** to another keystroke, then **Ctrl+C** interrupts MATLAB execution, regardless of whether you have selected an item.

Resolving Keyboard Shortcut Conflicts

To resolve a conflict, change or delete shortcuts such that there is a one-to-one correspondence between a shortcut and a frequently used action. For examples, see "Changing a Keyboard Shortcut" on page 2-87 and "Deleting a Keyboard Shortcut" on page 2-88.

Examples of Creating, Modifying, and Deleting Keyboard Shortcuts

Creating a New Keyboard Shortcut

By default, no keyboard shortcut is available for adding a Help topic to the list of favorites. If you frequently mark topics as favorites, you can define a keyboard shortcut for this action, as follows:

1. Choose **File > Preferences > Keyboard > Shortcuts**.
2. In the filter field, type **Help**.
3. Scroll through the **Action name** list, and select **Add to Favorites**.
4. Click the plus button 

   MATLAB adds a row to the table above the plus button.
5. In the **Shortcut** field, click the down arrow, and then change `Limit to 1 keystroke` to `Limit to 2 keystrokes`.
6. In the **Shortcut** field, press **Ctrl+S**, and then **Alt+V**.

   Notice that the All Possible Conflicts table is empty, which indicates that no other desktop action is currently using this combination of keystrokes.
7. Click **Apply.**

Notice that:

- The Add to Favorites dialog box opens when you press **Ctrl+S**, **Alt+V** in the Help browser.
- **Ctrl+S**, **Alt+V** appears next to **Add to Favorites** when you click the **Favorites** menu in the Help browser.

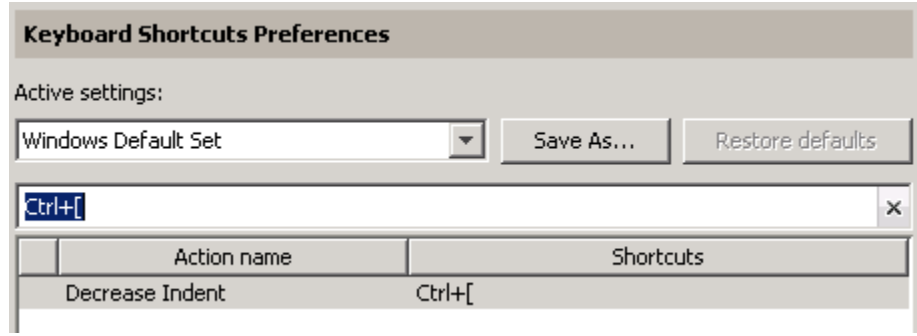
Changing a Keyboard Shortcut

Suppose you frequently adjust indenting in the MATLAB Editor. However, you have difficulty remembering the default keyboard shortcut of **Ctrl+[** for decreasing the indent. So, you decide to change it to something that is easier to remember.

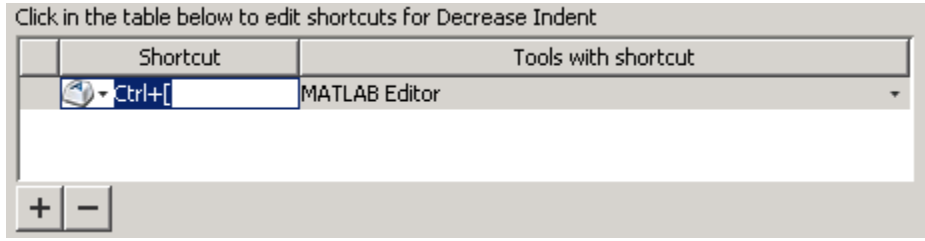
This example changes the keyboard shortcut for **Decrease Indent** in the MATLAB Editor from **Ctrl+[** to **Ctrl+Backspace:**.

1. Choose **File > Preferences > Keyboard > Shortcuts**.
2. Under **Active settings**, choose `Windows Default Set`.
3. In the filter field, press **Ctrl+[**.

   Under **Action name**, **Decrease Indent** displays in shading.



4. In the table labelled **Click in the table below to edit shortcuts for Decrease Indent**, under **Shortcut**, click **Ctrl+[**. MATLAB makes the field editable.



5 In the **Shortcut** field, press **Ctrl+Backspace**.

6 Click **Apply**.

MATLAB saves your changes to the `Windows Default Set (modified)` settings.

Deleting a Keyboard Shortcut

Suppose you find yourself frequently pressing the wrong keyboard shortcut. For example, you press **Alt+Enter** (apply M-Lint autofix) instead of **Ctrl+Enter** (evaluate the current cell in the MATLAB Editor). To avoid accidentally applying an M-Lint autofix, you decide to delete the **Alt+Enter** shortcut. Follow these steps:

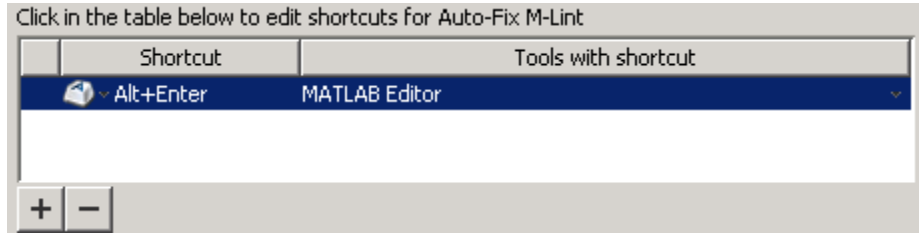
1 Choose **File > Preferences > Keyboard > Shortcuts**.

2 Under **Active settings**, choose `Windows Default Set` or `Windows Default Set (modified)`.

3 In the filter field, press **Alt+Enter**.

4 Under **Action name**, select the row containing **Autofix M-Lint**.

5 In the next table, under **Shortcut**, select the row containing **Alt+Enter**.



6 Click the remove button .

7 Click **Apply**.

If it does not exist, MATLAB creates a `Windows Default Set (modified)` keyboard shortcut set. This set consists of the `Windows Default Set` of keyboard shortcuts, less the shortcut for **Alt+Enter**. If the `Windows Default Set (modified)` settings file exists, then MATLAB deletes the **Alt+Enter** keyboard shortcut from that set of keyboard shortcuts.

Using Keyboard Shortcuts Settings Files Created on Other Systems

If you find a keyboard shortcuts settings file that is useful to you, or if you want to use one you created on a different system, make it the active keyboard shortcuts settings file as follows:

1 Copy the settings file to a folder on your system, such as:

```
I:\my_matlab_files\active_settings_files\new_settings.xml
```

2 Choose **File > Preferences > Keyboard > Shortcuts**.

3 In the **Active settings** field, click the down arrow, and then click **Browse**.

4 In the Open dialog box, navigate to the folder where you copied the settings file.

5 Select the settings file, and then click **Open**.

6 In the Keyboard Shortcuts preferences pane, click **Apply**. The settings file you specified is now the active settings file for MATLAB.

Consider using File Exchange to share your active settings file with others, or to find a file that is useful to you. For more information on File Exchange, see Chapter 7, "File Exchange — Finding and Getting Files Created by Other Users".

Keyboard Shortcut Restrictions

These sections describe the tools, portions of tools, and actions for which you cannot change keyboard shortcuts:

Tools for which You Cannot Customize Keyboard Shortcuts

You cannot change the keyboard shortcuts associated with the following tools or portions of tools:

- Figure windows—For example, you cannot modify the keyboard shortcut, **Ctrl+S**, for saving a MATLAB `.fig` file.
- Toolboxes—For example, you cannot modify keyboard shortcuts in the SimBiology® desktop.
- Incremental search—You can modify the keyboard shortcuts for initiating a forward or backward incremental search. However, you cannot change the keyboard shortcuts that you use within incremental search mode, such as **Ctrl+Shift+S** to search forward.
- Dialog boxes—For example, you cannot create a keyboard shortcut for the **OK** button.

Actions for Which You Cannot Customize Keyboard Shortcuts

The following table describes some frequently used actions for which you cannot customize keyboard shortcuts.

Action	Keyboard Shortcut
Cancel the current action.	**Esc** (escape) For example, if you select the **Edit** menu, the menu items display. Pressing **Esc** retracts the menu items. In the Function Browser, pressing **Esc** up to three times has the following effects: **1** Dismisses the search history **2** Clears the search field **3** Closes the Function Browser
Interrupt MATLAB execution on all supported platforms.	**Ctrl+C**
Interrupt MATLAB execution on Windows and UNIX systems.	**Ctrl+Cancel**
Interrupt MATLAB execution on Macintosh systems.	**Cmd+.** (period)
Open context menu on Windows and UNIX systems.	**Ctrl+Shift+F10**
Close the desktop and consequently shut down the MATLAB program. Outside the desktop, close the active window (except on Macintosh platforms).	**Alt+F4**

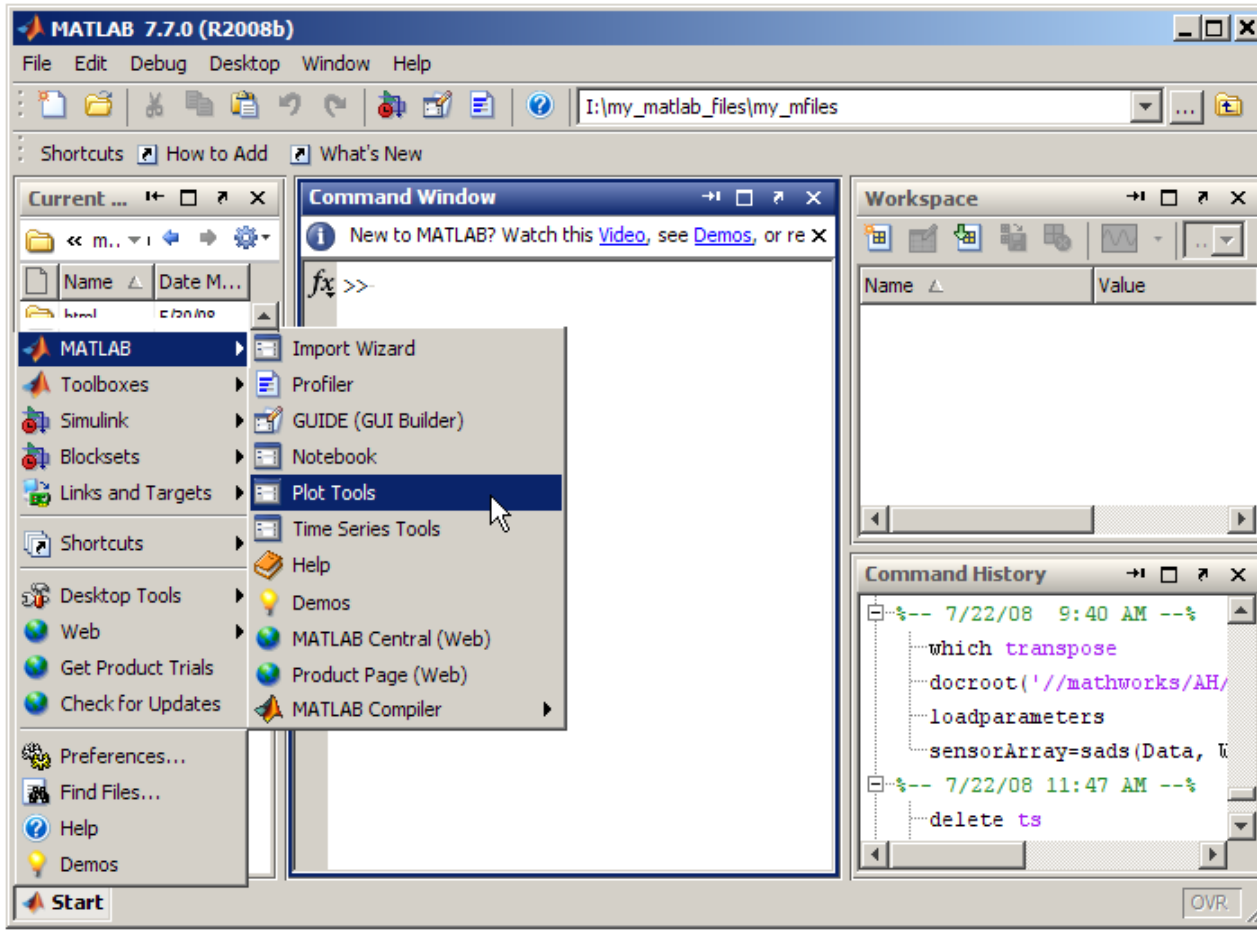
Action	Keyboard Shortcut
Accessibility affordances	**Tab** for navigating through fields in dialog boxes, for example.
Make an open tool the active tool	These shortcuts appear on the desktop **Windows** menu. They are: • Command History: **Ctrl+1** • Command Window: **Ctrl+0** • Current Folder: **Ctrl+2** • Editor: **Ctrl+Shift+0** • Figures: **Ctrl+Shift+1** • Figure Palette: **Ctrl+7** • File and Folder Comparisons: **Ctrl+Shift+4** • File Exchange: **Ctrl+6** • Help: **Ctrl+4** • Plot Browser: **Ctrl+8** • Profiler: **Ctrl+5** • Variable Editor: **Ctrl+Shift+3** • Web Browser: **Ctrl+Shift+2** • Workspace: **Ctrl+3**

Accessing Tools with the Start Button

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

Viewing Products and Tools with 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** (except on Apple Macintosh platforms). In the following illustration, the **Start** button shows MATLAB selected.



2 From the menu and submenu items, select an item to open it. The icons help you quickly locate a type of product or tool—see the icon descriptions in the following table.

For example, to open plot tools, select **Start > MATLAB > Plot Tools**.

Identifying Icons in the Start Button

Icons in the Start button menus 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.
	Block library opens.
	Document opens in your system Web browser.

Adding Your Own Toolboxes to the Start Button

About Adding Your Own Toolbox to the Start Button

When you provide a collection of files for use with MathWorks products to other users, it is called a *toolbox*. You can provide access to your toolbox from the **Start** button. (For background information about the **Start** button, see "Accessing Tools with the Start Button" on page 2-93.)

MATLAB determines the information to display on the **Start** button using `info.xml` files that are in folders on the search path:

- Create an `info.xml` file for your toolbox that contains tags for all the subentries you want on the **Start** button.

- Provide the `info.xml` file to your toolbox users, along with your toolbox files.
- With the `info.xml` file in a folder on their search path, users can access your toolbox from the **Start** button.

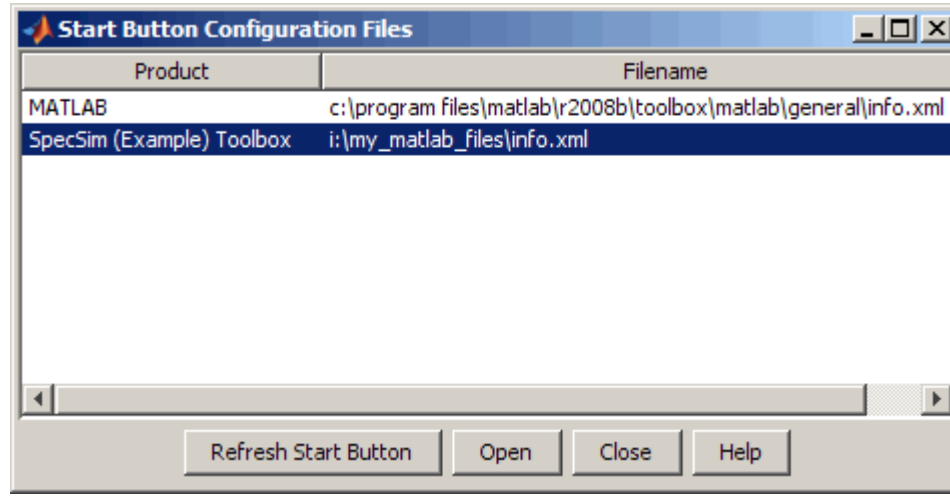
If you also want to include help files for your toolbox in the Help browser, put the Help browser and **Start** button information into a single `info.xml` file.

Procedure for Adding Your Own Toolbox to the Start Button

The example file, `startinfo.xml`, accompanies the following instructions. It adds the SpecSim (Example) Toolbox to the **Start** button. Use the example as a basis for creating an `info.xml` file for your own toolbox.

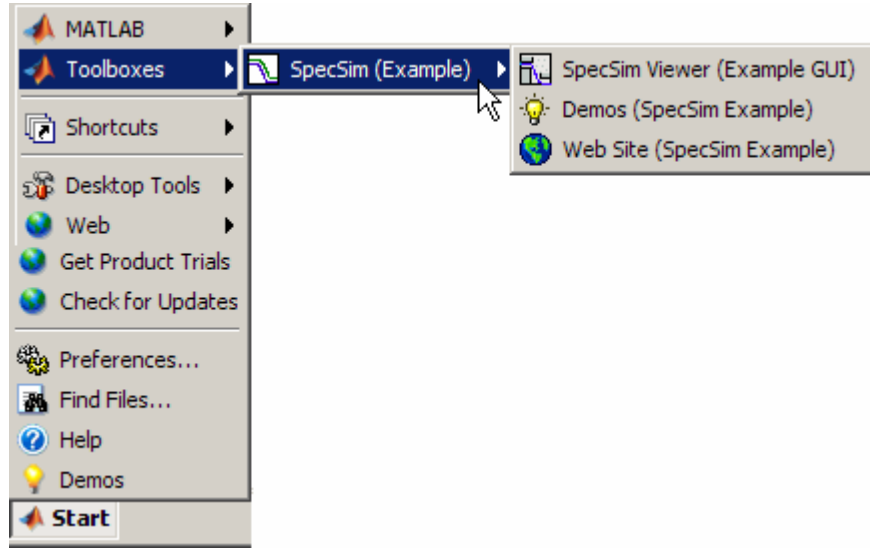
1. Create or choose a folder for storing your `info.xml`. You have write access to the folder.
2. Add the folder to the search path. The folder cannot be the current folder when you add it to the path.
3. Copy the example file *`matlabroot`*`/help/techdoc/matlab_env/examples/startinfo.xml` to the folder.
4. Rename the copy of `startinfo.xml` to `info.xml`.
5. Refresh the **Start** button so it can locate your `info.xml` file. Perform this step whenever you change the `info.xml` files for your toolboxes.
   a. Select **Start > Desktop Tools > View Start Button Configuration Files**.

      The Start Button Configuration Files dialog box opens.



b Click **Refresh Start Button**.

6 View the example `info.xml` file in the**Start** button. For the example, select **Start > Toolboxes > SpecSim (Example)**.



7. Get the info.xml file to edit it. A convenient way to get the file is from the Start Button Configuration Files dialog box:

   a. In the dialog box, select the `info.xml` file that you want to edit. For the example, select `SpecSim (Example)`.

   b. Click **Open**.

   The `info.xml` file for your selection opens in the Editor.

8. In the Editor, change the entries in the `info.xml` file so that they pertain to your toolbox. For details, see the sample code and table shown after the last step.

   For more information about the `info.xml` file structure, consult its schema, which is at *`matlabroot`*`/sys/namespace/info/v1/info.xsd`.

9. In the Editor, save the changes to `info.xml` by selecting **File > Save**.

10. Refresh the **Start** button so it can see the changes to the `info.xml` file. See step 5 for details.

    MATLAB automatically validates your `info.xml` file against the schema. If there is an invalid construct, MATLAB displays an error in the Command Window. For more information, see "Validating info.xml Files You Provide" on page 4-52.

11. View the updated entry in the **Start** button.

Description of info.xml File for Adding Your Own Toolbox to the Start Button

This table provides details about the example `info.xml` file.

Line	XML Tag	Value for Example	Notes
6	`<matlabrelease>`	`R2008b`	Release of MATLAB. Not currently used.
7	`<name>`	`SpecSim (Example)`	Name of the toolbox.

Line	XML Tag	Value for Example	Notes
8	`<type>`	`toolbox`	The product type. The value determines where the entry appears in the **Start** button. **SpecSim (Example)** appears under **Start > Toolboxes** Allowable values are `matlab`, `simulink, toolbox`, `blockset`, `links_targets`, `other`. Within a type, entries appear in alphabetical order.
9	`<icon>`	`$docroot/techdoc/matlab_env/examples/specsimicon.gif`	Shows an icon for your toolbox. You can specify a relative path, that is, relative to the location of the `info.xml` file. In this example, `specsimicon.gif` is in the `examples` folder provided with the documentation.
10	`<help_location>`	None	Location of HTML help files you provide for the toolbox. For details, see "Providing HTML Help Files" on page 4-37. Specify a relative path, that is, relative to the location of the `info.xml` file. In this example, there are no help files, so the entry has no value.

Line	XML Tag	Value for Example	Notes
12 to 15	<list> <listitem> <label>	SpecSim Viewer (Example GUI)	Name of a tool in your toolbox. You must have at least one list-listitem-label set of tags for the toolbox to appear in the **Start** button.
16	<callback>	specsim_viewer_example	The function that runs when you select the item from the **Start** button. You must have at least one callback for the toolbox to appear in the **Start** button. In this example, specsim_viewer_example is a function that starts the SpecSim Viewer GUI when you select **Start > Toolboxes > SpecSim (Example) > SpecSim Viewer (Example GUI)**. (This example file is not provided.)
17	<icon>	$docroot/techdoc/matlab_env/examples/guiicon.gif	Shows an icon for your callback. You can specify a relative path, that is, relative to the location of the info.xml file. In this example, guiicon.gif is in the examples folder provided with the documentation.
21	<label>	Demos (SpecSim Example)	For the examples you provide with the toolbox.

Line	XML Tag	Value for Example	Notes
22	`<callback>`	`specsim_demos_example`	The function that runs when you select the item from the **Start** button. In this example, `specsim_demos_example` is an example that runs when you select **Start > Toolboxes > SpecSim (Example) > Demos (SpecSim Example)**. (This example file is not provided.)
23	`<icon>`	`$docroot/techdoc/matlab_env/examples/demoicon.gif`	Shows an icon for your demos. You can specify a relative path, that is, relative to the location of the `info.xml` file. In this example, `demoimicon.gif` is in the `examples` folder provided with the documentation.
27	`<label>`	`Web Site (SpecSim Example)`	For a link to your toolbox Web site.
28	`<callback>`	`web http://www.specsimexample.com -browser;`	The statement that runs when you select the item from the **Start** button. In this example, select **Start > Toolboxes > SpecSim (Example) > Web Site (SpecSim Example)**. The Web site for the SpecSim (Example) Toolbox opens in the system browser. (This URL is intentionally invalid.)
30 to 34	`</listitem>` `</list>` `</productinfo>`	None	Designates end of `list` and `productinfo` entries in the **Start** button for the toolbox.

See Also

- "Validating info.xml Files You Provide" on page 4-52
- "Providing Your Files in the Help Browser" on page 4-37

Using Web Browsers from MATLAB

In this section...

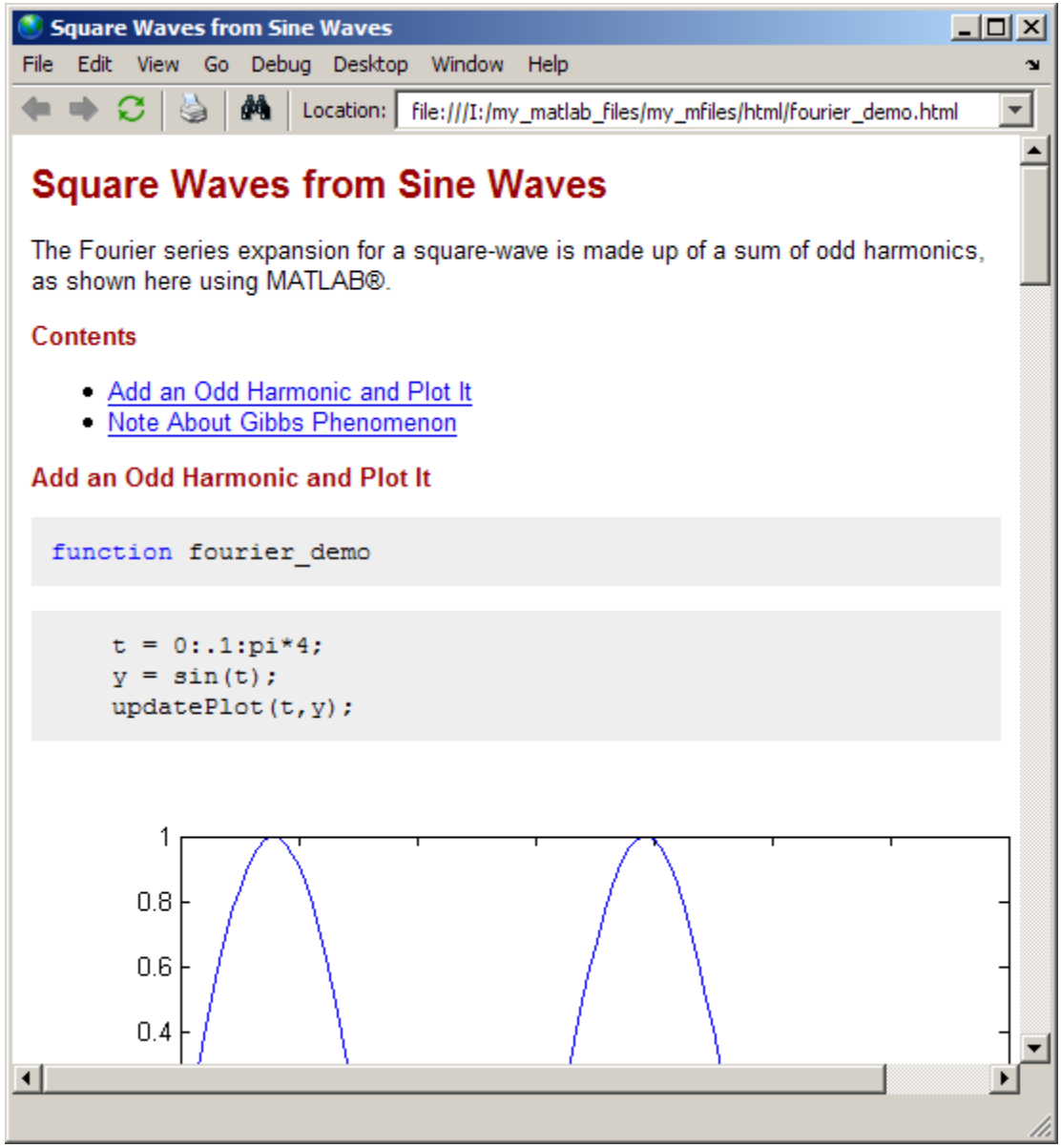
About Web Browsers in MATLAB

From MATLAB, Web sites and documents can display in any of the following browsers:

- MATLAB Web browser
- Help browser
- Your system Web browser, such as Mozilla® Firefox®

MATLAB uses different browser to display different types of information:

- Web sites display in your system browser.
- Documentation and demo pages display in the Help browser.
- Other HTML files display in the MATLAB Web Browser. For example, after publishing an M-file to HTML, the HTML file displays in the MATLAB Web Browser:


Square Waves from Sine Waves
File Edit View Go Debug Desktop Window Help
Location: file:///I:/my_matlab_files/my_mfiles/html/fourier_demo.html
Square Waves from Sine Waves
The Fourier series expansion for a square-wave is made up of a sum of odd harmonics, as shown here using MATLAB®.
Contents
Add an Odd Harmonic and Plot It
Note About Gibbs Phenomenon
Add an Odd Harmonic and Plot It
function fourier_demo
t = 0:.1:pi*4;
y = sin(t);
updatePlot(t,y);
1
0.8
0.6
0.4

MATLAB Web Browser

Because the MATLAB Web and Help browsers are desktop tools, you can perform desktop operations on them, such as docking the tools in the desktop. For more information, see Chapter 2, “Desktop”.

The MATLAB Web Browser may not support all the features that a particular Web site or HTML page uses. For example, the MATLAB Web Browser does not display `.bmp` (bitmap) image files. Instead use `.gif` or `.jpeg` formats for image files in HTML pages.

System Browser

The system browser that MATLAB uses depends on your platform:

- On Microsoft Windows and Apple Macintosh platforms, MATLAB uses the default browser for your operating system.
- On UNIX platforms, MATLAB uses the Mozilla Firefox browser. You can specify a different system browser for MATLAB using Web preferences.

Displaying Pages in Web Browsers

To display an HTML document in the MATLAB Web Browser, double-click the document name in the Current Folder browser.

To display a Web page or file in the MATLAB Web Browser:

1. Select **Desktop > Web Browser**.

   An empty MATLAB Web browser window opens.

2. Type a URL or full path to a filename in the **Location** field.

To open a link to another page in a separate MATLAB Web browser window, click the middle mouse button, if you have one.

To open any type of document in any type of Web browser MATLAB supports, use the `web` function. Specify a URL or file to display, and the type of browser to use.

Web Preferences

Use Web preferences to specify characteristics related to accessing the Internet:

Specifying Proxy Server Settings

If your network uses a firewall or another method of protection that restricts Internet access, provide information about your proxy server to MATLAB.

To specify the proxy server settings:

1. Select **File > Preferences > Web**.
2. Select the **Use a proxy server to connect to the Internet** check box:
3. Specify values for **Proxy host** and **Proxy port**. Examples of acceptable formats for the host are: `172.16.10.8` and `ourproxy`. For the port, enter only an integer, such as `22`. If you do not know the values for your proxy server, ask your system or network administrator for the information.

   If your proxy server requires a user name and password, select the **Use a proxy with authentication** check box. Then enter your proxy user name and password.

   Note MATLAB stores the password without encryption in your `matlab.prf` file.

   ---

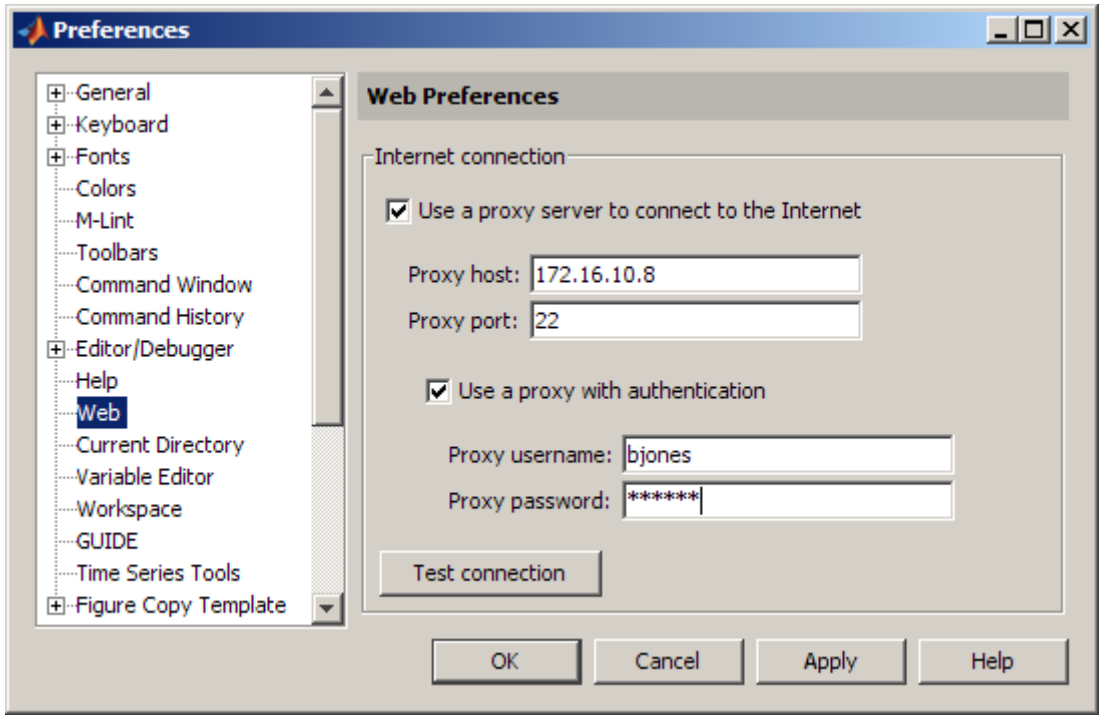
4. Ensure that your settings work by clicking the **Test connection** button.

   MATLAB attempts to connect to `http://www.mathworks.com`:

   - If MATLAB can access the Internet, **Success!** appears next to the button.

- If MATLAB cannot access the Internet, **Failed!** appears next to the button. Correct the values you entered and try again. If you still cannot connect, try using the values you used when you authenticated your MATLAB license.

5 Click **OK** to accept the changes.



After specifying the preference, you can access the Internet from MATLAB through your proxy server.

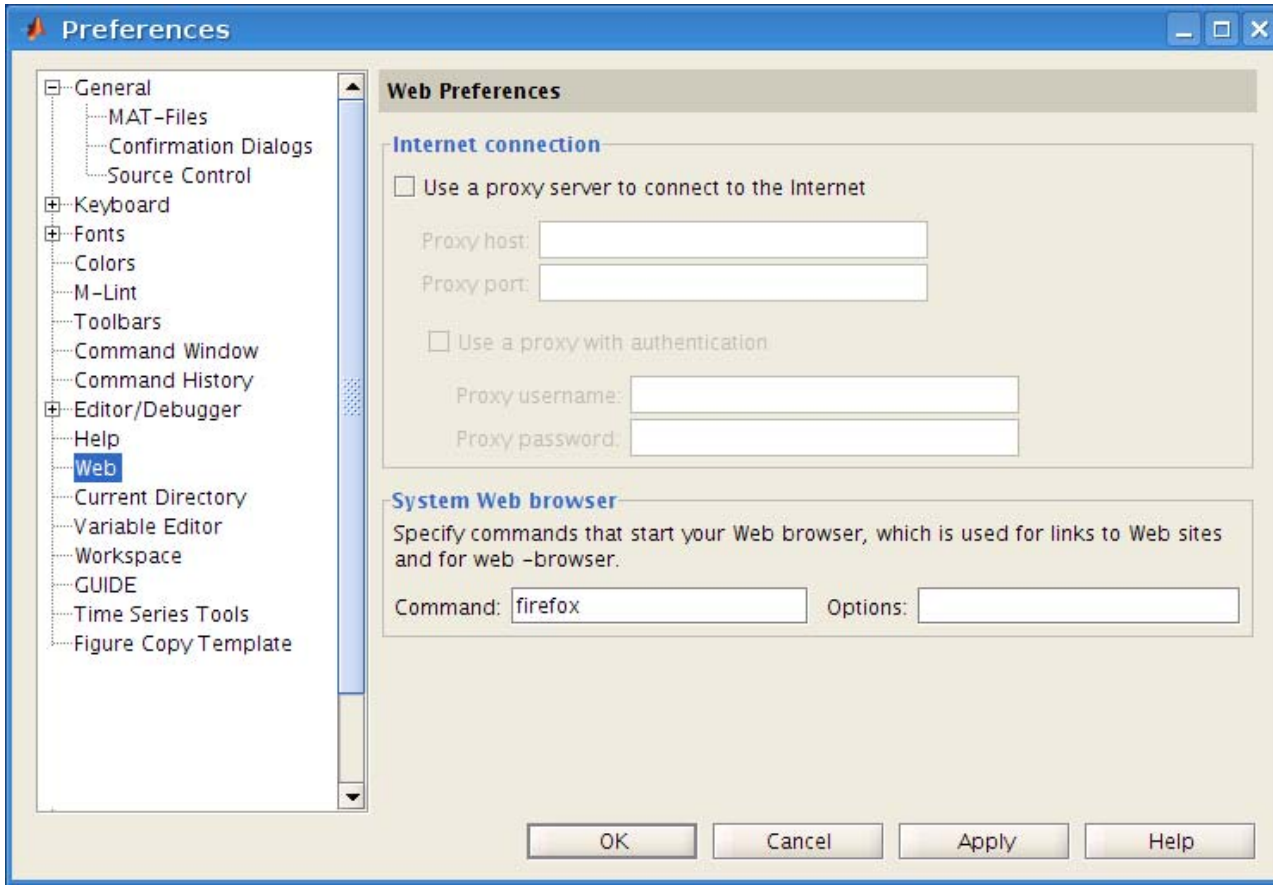
Limitations of Specifying Proxy Server Settings.

- You cannot specify the proxy server settings using a script.
- There is no automated way to provide the proxy server settings your system browser uses to MATLAB.

Specifying the System Browser for UNIX Platforms

For background information, see "System Browser" on page 2-105. To specify the system browser:

1 Select **File > Preferences > Web**. The Preferences dialog box opens to the Web pane.



Note The **System Web browser** preference is for UNIX platforms (excluding Macintosh) and does not appear in the preferences pane for other platforms.

2. Under **System Web browser**, in the **Command** field, specify the system command to open the browser, for example, `opera`, which opens the Opera Web browser.

3. Add options for opening your system browser in the **Options** field. For example, `geometry 1064x860` specifies the size of the window for Opera.

4. Click **OK**.

Specifying Fonts for the MATLAB Web Browser

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

Other Desktop Features

In this section...

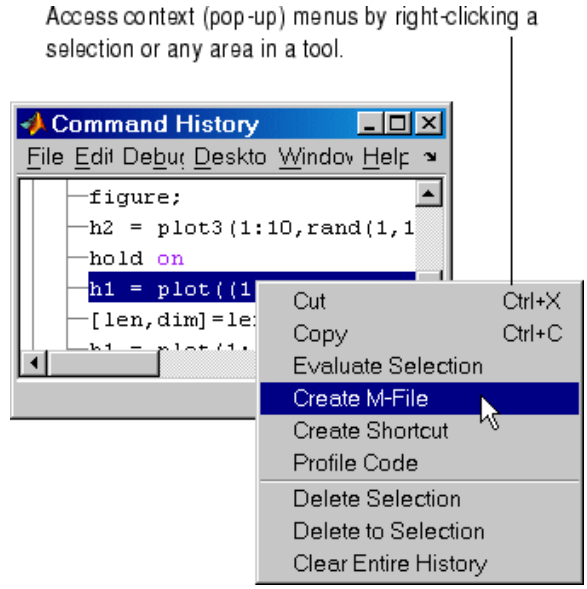
Using Menus and Context Menus

Understanding 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. However, 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

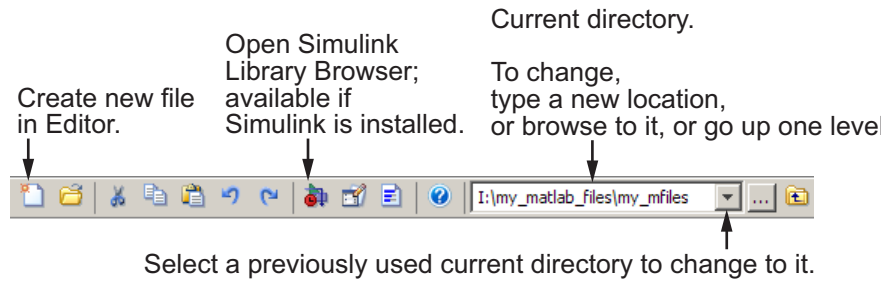
Many of the features in MATLAB desktop tools are available from context menus, also known as pop-up or right-click menus. To access a context menu, right-click a selection or an area. The context menu for the selection or tool appears, presenting the available actions. For example, following is the context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

Using Toolbar Features

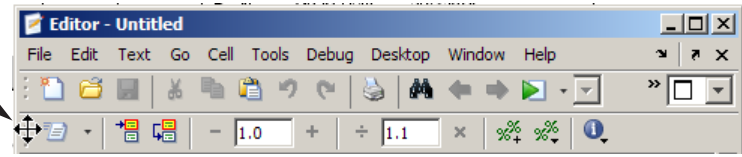
The toolbar in the desktop provides easy access to frequently used operations. Some other tools also provide toolbars. The following illustration shows some key features of the desktop toolbar.



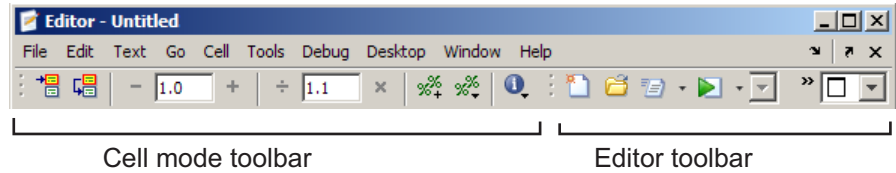
The following are the major toolbar features:

- Tooltips — Position the pointer over a button for a couple seconds and a tooltip appears describing the item.
- Customizing — You can customize the toolbar to show or remove controls, and to rearrange the controls. Use **File > Preferences > Toolbars**. For details, click **Help** in the resulting dialog box.
- Toolbars in Tools — Some tools have their own toolbars, which are located within that tool's window. For example, the Current Folder browser has its own toolbar. When you undock one of these tools, the undocked tool includes the toolbar.
- Hiding Toolbars — To hide a toolbar, or to show it again after previously hiding it, select **Desktop > Toolbars**, and select the toolbar of interest. As an alternative, right-click a toolbar or menu bar and select a toolbar from the context menu to hide or show it. In a figure window, use the **View** menu to select the toolbar of interest.
- Repositioning Toolbars — If a tool has more than one toolbar, you can change the position of the toolbars. For example, in the Editor, the default is for the Editor toolbar to be above the Cell Mode toolbar. To move a toolbar, grab the toolbar anchor (at the left end) and drag the toolbar to a different location.

To move a toolbar, grab the toolbar anchor and drag the toolbar to a new location.



Here, the Cell Mode toolbar has been moved before the Editor toolbar.



Viewing and Changing the Current Folder in the Desktop Toolbar

The current folder field in the desktop toolbar shows the current working folder in MATLAB. You can use this field to change the current folder. Click the down arrow in this field, for example, to select a folder that you previously used from the history.

Viewing Status in the Status Bar

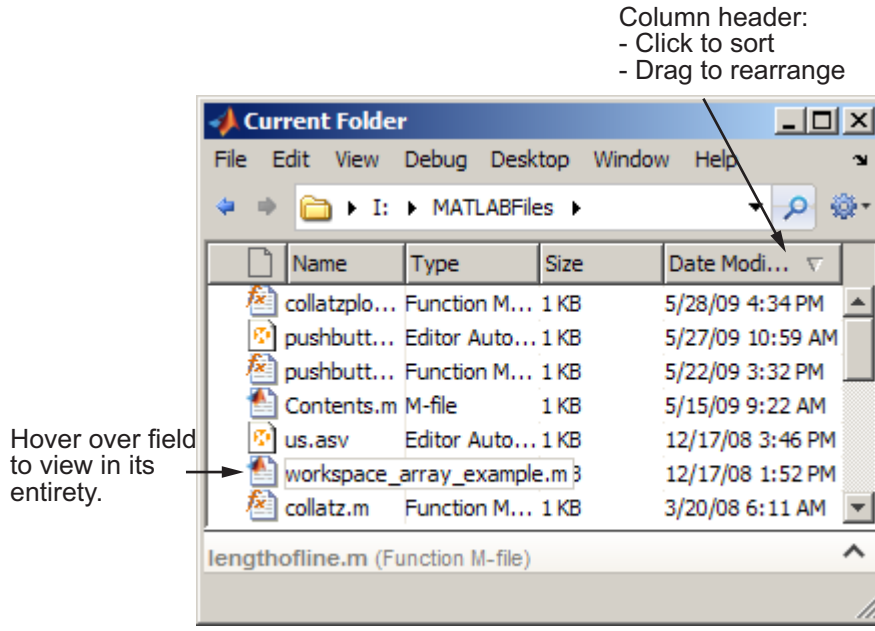
Along the bottom of the desktop is the status bar. It provides status information, such as when MATLAB is busy executing statements or when the Profiler is on.

You can construct your own functions to provide status information. One method is the `timer` function. Use the Help browser search feature to find other specific terms describing the status you want.

Sizing, Arranging, and Sorting Columns in Desktop Tools

Some desktop tools present information in columns, such as the Current Folder browser. The following table describes how you can resize and reposition the columns, as well as sort the information in the columns.

To...	Do This...
Change the column width	Drag the separator bar between two column headings.
View all the information in a column that is too narrow to show it all	Position the pointer over an item to view the full value for that item. It displays like a tooltip.
Rearrange the columns	Drag a column header to a different position.
Sort the information by a particular column	Click the column header. For example, in the Current Folder browser, click the **Date Modified** date to sort the items in date order. In some columns, you also can reverse the sort order by clicking the column header again. A small gray arrow in the header indicates the current sort order. For example, a down arrow in the **Date Modified** column header indicates a descending sort order. The newest files are at the top of the list.



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 you run on a Microsoft Windows platform, 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, **Ctrl**+click that item. To clear all selected items, click outside of the selection.

See also, "Performing Desktop Actions Using the Keyboard" on page 2-68

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. You can 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 appropriate MATLAB desktop tools, such as the Editor.

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 wizard to copy data from another application, such as the Microsoft® Excel® application, into MATLAB. For details, see "Using the Import Wizard" in the MATLAB Data Import and Export documentation.

When editing in the Command Window and the Editor, you can move text to a new location by selecting the text and dragging 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 also can 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 open a file in the Editor by dragging the file name from the Current Folder browser to the Editor. If you drag editable text (for example, text in the Editor), 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 document created using the Microsoft Word application into the Editor cuts and pastes it into the open file. Dragging an M-file from Windows Explorer tool 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 to the Word application.

Macintosh Platform — Differences

GUI Conventions in the Documentation and Macintosh Platforms

MATLAB on the Apple Macintosh platform sometimes uses conventions that are standard for the Macintosh platform, but might be different from what the MATLAB documentation states. The documentation typically presents conventions for Microsoft Windows platforms. The intended action for the Macintosh platform is typically obvious. For example, the documentation might instruct you to do the following, which is the convention on Windows platforms:

1. Select **File > Save**.
2. Select**Yes**, **No**, or **Cancel** from the Save dialog box.

However, on Macintosh platforms, the Save dialog box presents the options **Don't Save** and **Save**.

Pointer Device Instructions and Macintosh Platforms

The standard mouse for Macintosh platforms 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 instructions for the Macintosh platform. When the documentation instruction is right-click, use **Ctrl**+click on the Macintosh platform. When

the documentation instruction is middle-click, use **Command**+click on the Macintosh platform.

Using File Browser GUIs on Macintosh Platforms to Navigate Within the MATLAB Root Folder

On Macintosh platforms, you cannot use a file browser GUI to navigate directly to a file or folder within the MATLAB root folder. (The MATLAB root folder also called *`matlabroot`*, which is the folder where MATLAB is installed). When you use the Macintosh Finder or a file browser GUI in MATLAB and select `Applications/R2009b_MATLAB`, no contents appear. On Macintosh platforms, the MATLAB root folder is `R2009b_MATLAB.app`, and the Macintosh operating system does not display the contents of applications (items with the `.app` extension).

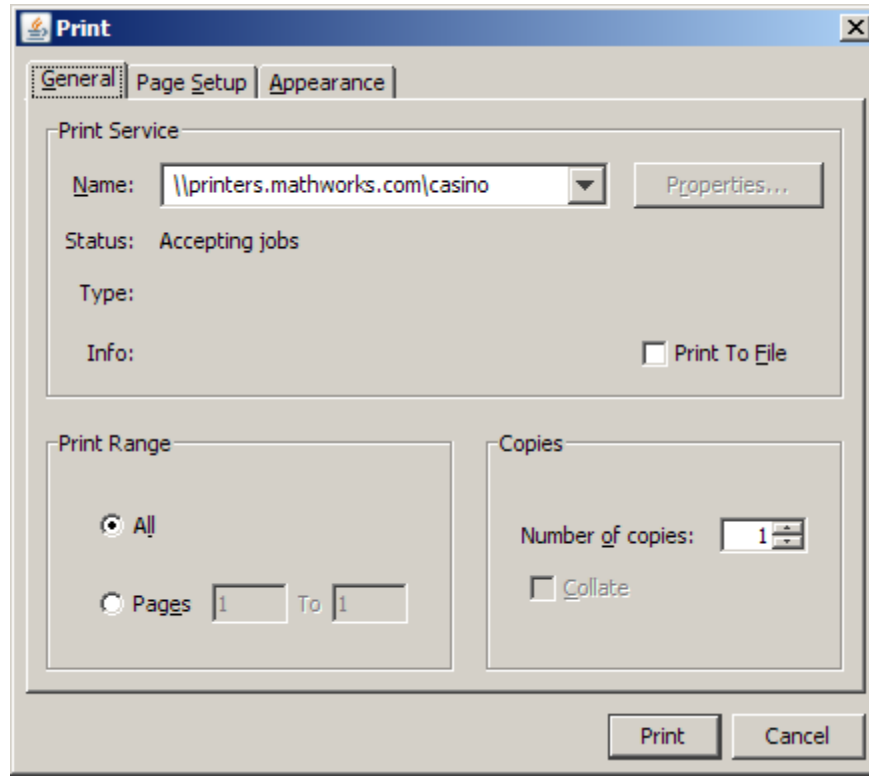
Here are some ways to view or open the contents of the MATLAB root folder via a file browser GUI:

- In the Macintosh Finder, right-click (or **Ctrl**+click) `MATLAB_R2008b`, and from the context menu, select **Show Package Contents**.
- In a MATLAB GUI where you cannot access the contents of `MATLAB_R2008b`, press **Command+Shift+G**, which opens the Go To Folder dialog box. In the Go To Folder dialog box, enter the full path to *`matlabroot`*, for example, `/Applications/MATLAB_R2008b.app/`. While typing the path in the Go To Folder dialog box, you can use autocomplete. That is, you can type the first few characters in the path or file name, and pause; the remaining characters matching an existing name appear. For example, type `/App` and autocompletion displays `/Applications`. Then add `/MAT`, and autocompletion displays `/Applications/MATLAB_R2008b.app/`. After entering the path, press **OK** in the Go To Folder dialog box. The MATLAB GUI then displays the contents of the MATLAB root folder.
- Use an alternative to the GUI. For example, instead of using **File > Open** to open an M-file in the Editor, use the `edit` function.
- In the Command Window, change the current folder to *`matlabroot`* by running `cd(matlabroot)`, and then open the GUI. Some GUIs then display the contents of *`matlabroot`*.

Printing and Page Setup Options for Desktop Tools

You can print from all desktop tools, except the Current Folder 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 browser, the Help browser, and the Profiler. However, it 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 Variable Editor, select **File > Page Setup**. A standard page setup dialog box for your platform opens.

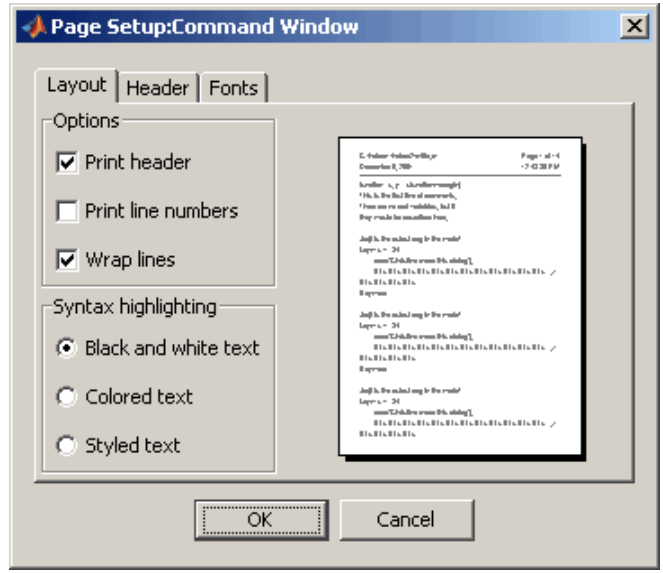
MATLAB provides special page setup options for printing from the Command Window and Editor. The setup options are essentially the same for both tools, with minor variations. This section covers their use:

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 print, 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**. Next, 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 use 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 Command Window font. 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.

Accessing The MathWorks on the Web

You can access popular pages on the MathWorks Web site from the MATLAB desktop.

To download trial versions of products that you do not have, select **Help > Get Product Trials**.

For access to other popular Web site pages, select one of the following items from the **Help > Web Resources** menu. The selected Web page opens in your system Web browser:

- **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, or report new problems.
- **Training** — List of courses for learning to use MathWorks products (`http://www.mathworks.com/services/training/courses/`).
- **MathWorks 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 developments for MATLAB.

- **MATLAB Central** — The user community for MATLAB (`http://www.mathworks.com/matlabcentral/`) for . It includes contests for MATLAB users and a screen saver with the logo for MATLAB.
- **MATLAB File Exchange** — Code library of files contributed by MathWorks customers and employees, available for free download and use with MathWorks products. You can also access the repository using the File Exchange desktop tool. For more information, see Chapter 7, "File Exchange — Finding and Getting Files Created by Other Users".
- **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 users of MATLAB. 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`.

Managing Your Licenses

You can use the MATLAB licensing features to perform license management activities, such as activating licenses, deactivating licenses, or updating licenses. You also can visit the License Center at the MathWorks Web site to perform other license-related activities.

To access the licensing feature:

1. Select **Help > Licensing**.
2. Select the activity you want to perform from the **Licensing** menu. The following table describes the options. Depending on your license type, the **Licensing** menu on your system might not include all options.

Note Some options require an Internet connection. If your Internet connection requires a proxy server, use MATLAB Web preferences to specify the server host and port. See "Specifying Proxy Server Settings" on page 2-106 for more information.

Option	Description
Activate Software	Starts the activation application, which walks you through the activation process. Answer the questions on each dialog box, select the license you want to activate, and click **Activate**.
Deactivate Software	Displays a list of all your MathWorks licenses on this computer, with their current status. When you select a license and click **Deactivate Selected License**, MATLAB deactivates all releases on this computer associated with the license, and updates the licensing information at the MathWorks Web site. You will not be able to use MathWorks software with that license on this computer. If you are not connected to the Internet, MATLAB deactivates the licences on your computer but cannot update the corresponding license information stored at the MathWorks Web site. In this scenario, MATLAB returns a *deactivation string*. To complete deactivation, save a copy of this string, go to a computer with an Internet connection, and visit the License Center at the MathWorks Web site. There you can login to your MathWorks Account and enter the deactivation string.

Option	Description
Update Current Licenses	Displays a list of all your MathWorks licenses on this computer, with their current status. When you select a license and click **Update Selected License**, MATLAB contacts The MathWorks to retrieve the most current version of the License File for the license. The update process overwrites the current License File on your system. You will need to restart MATLAB.
Manage Licenses	Starts a Web browser, opening the My Licenses page associated with your MathWorks Account. You can use this page, called the License Center, to perform many licensing activities.

Check for Updates

To determine if more recent versions of your MathWorks products are available, and to view latest version numbers for all MathWorks products, use the Check for Updates feature.

To access the Check for Updates feature, you must have an active Internet connection. Then, follow these steps:

1. Select **Help > Check for Updates**. The Check for Updates dialog box displays.
2. From the **Select View** list, choose to view the latest version numbers for all MathWorks products installed on your system, or all MathWorks products. The latest versions are displayed.
3. Click any column heading to sort or reverse the sort order by that column.
4. Use the What's New column to access the release notes for a product. Release notes document new features and changes, bug reports, and compatibility considerations.
5. To upgrade to the most recent version, click **Download Products at MathWorks.com**, which links to the Downloads area of the MathWorks Web site. If you do not want to upgrade at this time, click **Close**.

Specifying Options for MATLAB Using Preferences

In this section...

Setting Preferences for MATLAB

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

1. Select **File > Preferences**. Alternatively, click the Preferences button on the desktop toolbar; if the button is not on the toolbar, you can add it—for information, see "Setting Toolbars Preferences for Desktop Tools" on page 2-156.

2. From the left pane of the Preferences dialog box, choose a tool or product and click the **+** to display more preferences for that item. 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 sessions of MATLAB.

Note that some tools allow you to control these settings from within the tool without setting a preference. Use that method if you want the change to apply only 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, and source control.
Keyboard	Tab completion, function hints, and delimiter matching for the Command Window and Editor. Keyboard shortcuts for desktop tools.
Fonts	Font type, style, and size for desktop tools. Customize for any tool.
Colors	Colors for text, background, syntax highlighting for M-files, and hyperlinks in desktop tools.
M-Lint	Show or hide M-Lint messages in the Editor M-Lint automatic code analyzer and in the M-Lint Code Check Report.
Toolbars	Remove, add, and rearrange controls on toolbars for desktop tools.
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 syntax highlighting colors for all files other than MATLAB files), code folding, and autosave.
Help	Product filter, help on selection window, and PDF reader for Linux platforms.
Web	Internet proxy server settings.
Current Folder	Number of entries in history and refresh options.
Variable 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. For details, click the **Help** button in the Preferences dialog box.
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

MATLAB and other MathWorks products store their preferences in the file `matlab.prf`. Type `prefdir` in the Command Window to see the full path for the folder where `matlab.prf` is located, called the preferences folder. The preference folder also contains other related files.

On Apple Macintosh platforms, the folder might be in a hidden folder, for example, `myname/.matlab/R2009b`. To access the folder, select **Go > Go to Folder** in the Apple Mac OS® Finder tool. In the resulting dialog box, type the path returned by `prefdir` and press **Enter**.

The `matlab.prf` file loads when you start MATLAB. When you make changes to preferences while using MATLAB, it makes the changes to `matlab.prf`. When you closeMATLAB, it saves those changes to `matlab.prf`.

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

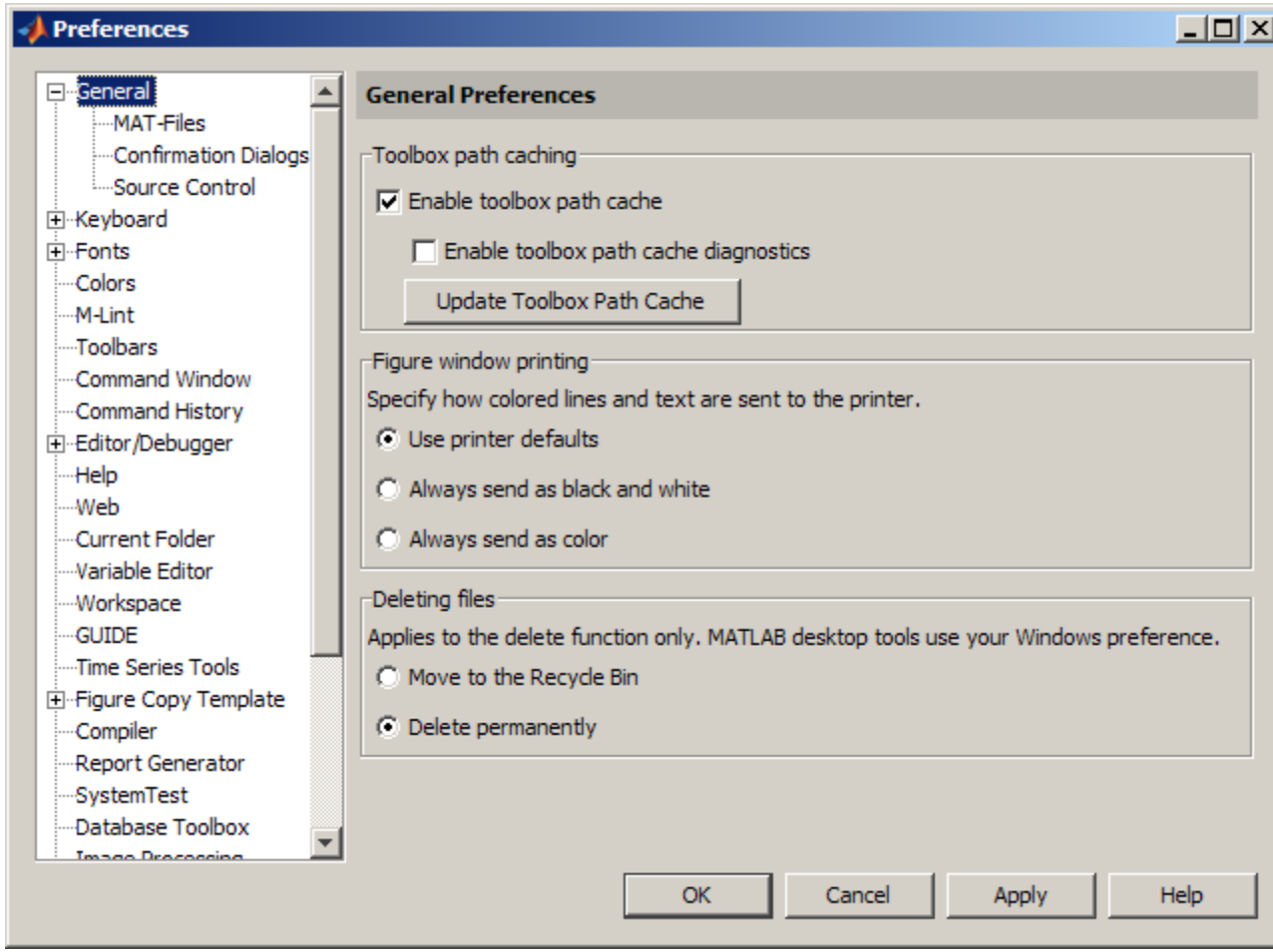
Setting General Preferences for the MATLAB Application

In this section...

General Preferences

Select **File > Preferences > General** from any desktop tool to access **General Preferences**.

These preferences apply to all relevant tools in the MATLAB application.



Toolbox Path Caching

See “Toolbox Path Caching in the MATLAB Program” on page 1-22.

Figure Window Printing

See “Printing and Exporting” in MATLAB Graphics documentation.

Deleting Files

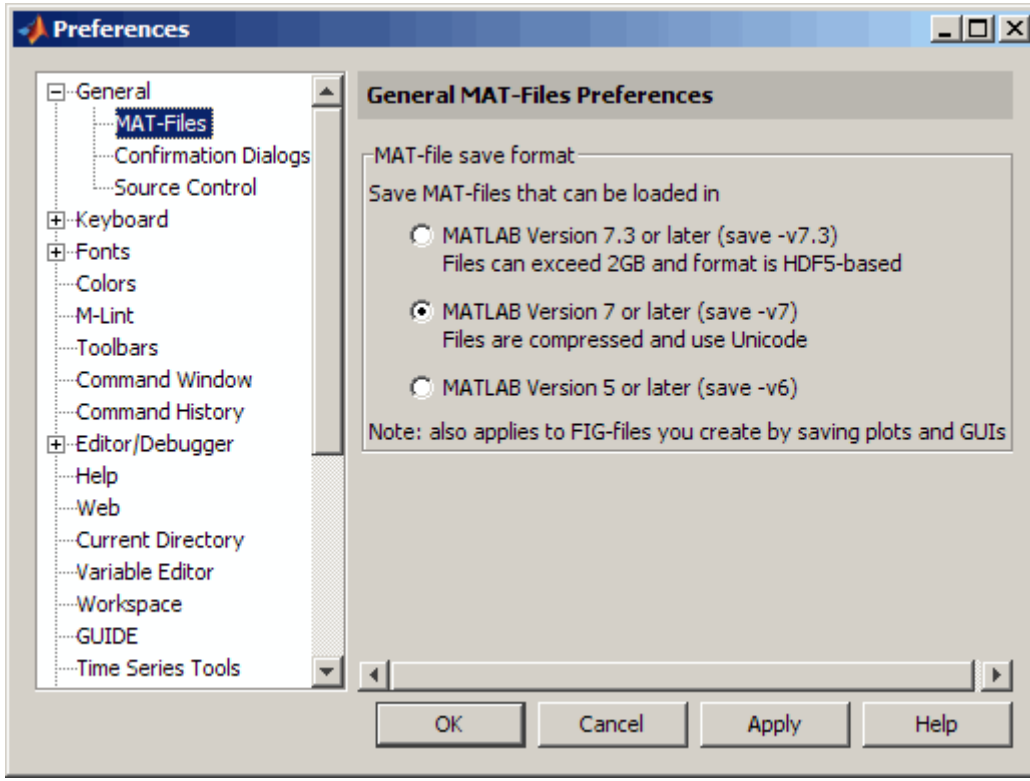
See:

- "Deleting Files and Folders Using the Current Folder Browser" on page 6-32
- "Deleting Files and Folders Using Functions" on page 6-33

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 users of MATLAB, 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 7.6 preference is `-v6`, when you upgrade from MATLAB Version 7.6 (R2008a) to MATLAB Version 7.8 (R2009a), the initial value in Version 7.8 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`.

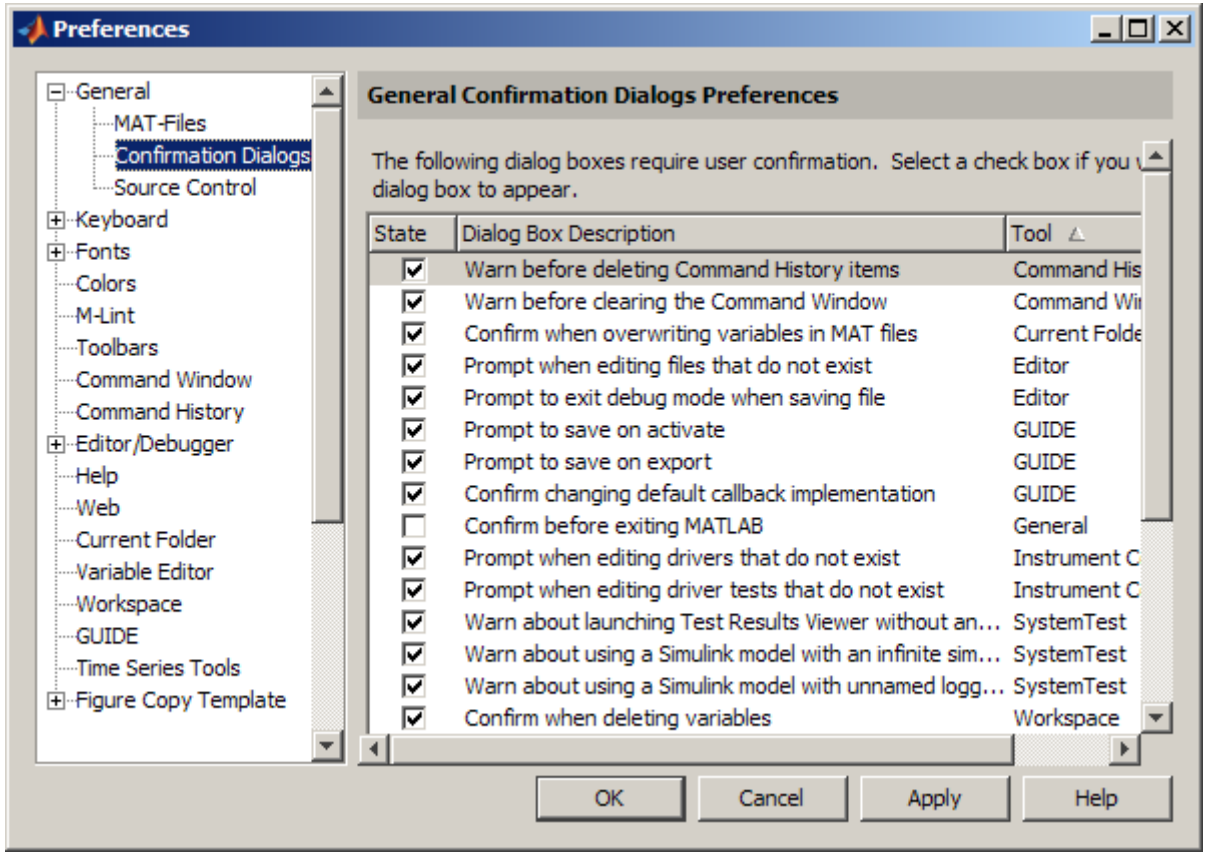
Note For more information about MAT-file save formats, including restrictions, see Version Compatibility Options and Remarks in the `save` reference page.

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

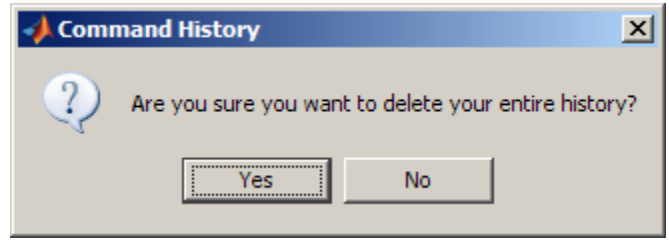
These preferences instruct MATLAB to display or not display specific confirmation dialog boxes.



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 > Clear Command History**. MATLAB displays the following confirmation dialog box.



If a confirmation dialog box includes a **Do not show this prompt again** check box and you click **OK**, the confirmation dialog box will not appear the next time you perform the action. In addition, the check box in the **Confirmations Dialogs** preferences pane is cleared.

The following table summarizes the confirmation dialog boxes for MATLAB. There might be additional confirmation dialog boxes listed for other products you have installed.

Confirmation Dialogs Check Box Item	Description of 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 3-70
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 3-47
Confirm when overwriting variables in MAT-files	Appears when you save variables by dragging them from the Workspace browser onto a MAT-file in the Current Folder browser.	"Creating and Updating MAT-Files" on page 6-31
Prompt when editing files that do not exist	Appears when you type `edit filename`, if `filename` does not exist in the current folder or on the search path.	"Function Alternative for Creating New Files" on page 8-9

Confirmation Dialogs Check Box Item	Description of the Confirmation Dialog Box	For More Information
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 8-169
Show linking and brushing message bar	Displays the message bar in a figure window that provides links to help for data brushing and linked plots.	"Exploring Data in Graphs"
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 MATLAB Creating Graphical User Interfaces 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 MATLAB Creating Graphical User Interfaces documentation
Confirm changing default callback implementation	Appears after you have modified a callback signature in GUIDE.	"Changing Callbacks Assigned by GUIDE" in the MATLAB Creating Graphical User Interfaces documentation
Confirm before exiting MATLAB	Appears when you quit MATLAB.	Quitting MATLAB

Confirmation Dialogs Check Box Item	Description of the Confirmation Dialog Box	For More Information
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 5-9

Source Control Preferences

For information, see Chapter 12, "Source Control Interface".

Customizing the Desktop Using Preferences

In this section...

Setting Keyboard Preferences for Desktop Tools

Select **File > Preferences > Keyboard** to set the following preferences for the Command Window and Editor/Debugger:

- Tab completion
- Function hints
- Delimiter matching

See also "Customizing Keyboard Shortcuts" on page 2-78.

Setting Tab Completion Preferences

Enable in Command Window. Select the check box to use tab completion when typing functions in the Command Window—for more information about the feature, see "Completing Statements in the Command Window — Tab Completion" on page 3-21. Clear the check box if you do not want to use the tab completion feature. With the tab completion preference cleared, 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 3-60.

Enable in Editor/Debugger. Select the check box to use tab completion when typing functions in the Editor—for more information about the feature, see "Completing Statements in the Command Window — Tab Completion" on page 3-21. Clear the check box if you do not want to use the tab completion feature. With the tab completion preference cleared, when you press the **Tab** key, MATLAB moves the cursor to the next tab stop rather than completing a function. For related information, select **File > Preferences > Editor/Debugger > Tab**, and click **Help**.

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 3-24.

View Command Window tab key preferences. Click the link to set preferences for the **Tab** key size in the Command Window, which MATLAB uses when the tab completion preference is not enabled.

View Editor/Debugger tab key preferences. Click the link to set preferences for the **Tab** key size and indenting preferences in the Editor/Debugger.

Setting Function Hints Preferences

To show function hints in the Command Window and Editor, select the function hints check boxes. If you do not want to use function hints, clear the check boxes. Function hints are a reminder of the syntax for a function that you use while entering a statement. The hints appear in a temporary pop-up window when you enter the opening parenthesis after a function name. For more information, see "Viewing Function Syntax Hints While Entering a Statement" on page 3-30.

Setting Delimiter Matching Preferences

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

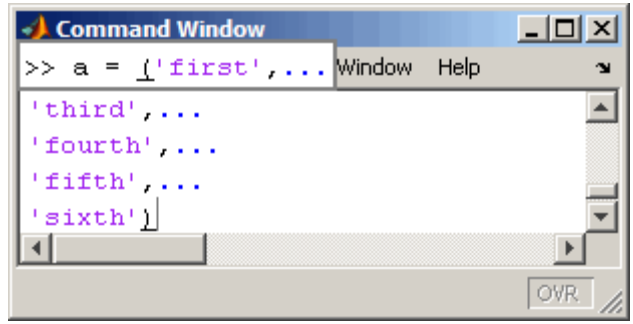
With these preferences selected, MATLAB alerts you to matched and unmatched delimiters based on the MATLAB language syntax rules. For example, when you type a parenthesis or another delimiter, MATLAB highlights the matched parenthesis or delimiter in the pair.

Delimiter pairs are parentheses ( ), brackets [ ], and braces { }. For the Editor, 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, 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, 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 crossed out briefly.
- `None` — There is no action.

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, 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.

Setting Fonts Preferences for Desktop Tools

You can specify your preferences for fonts that the desktop tools use. The first time MATLAB uses or displays the list of available fonts, it gets the operating system's font list. If a font exists, but MATLAB cannot display it, then MATLAB excludes it from its list. The system fonts are installed in one of the following locations:

- The operating system's standard location

  Ask your system administrator where this is on your system.
- The `/jre/lib/fonts` folder where Java software is installed on your system.

See the following sections for details on setting fonts preferences:

- "Desktop Fonts Preferences" on page 2-142
- "Custom Fonts Preferences" on page 2-146
- "Changing the Font — Example" on page 2-147
- "Antialiasing for Desktop Fonts on Linux and UNIX Platforms" on page 2-149
- "Making Fonts Available to MATLAB Tools on Windows Platforms" on page 2-150

Desktop Fonts Preferences

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:

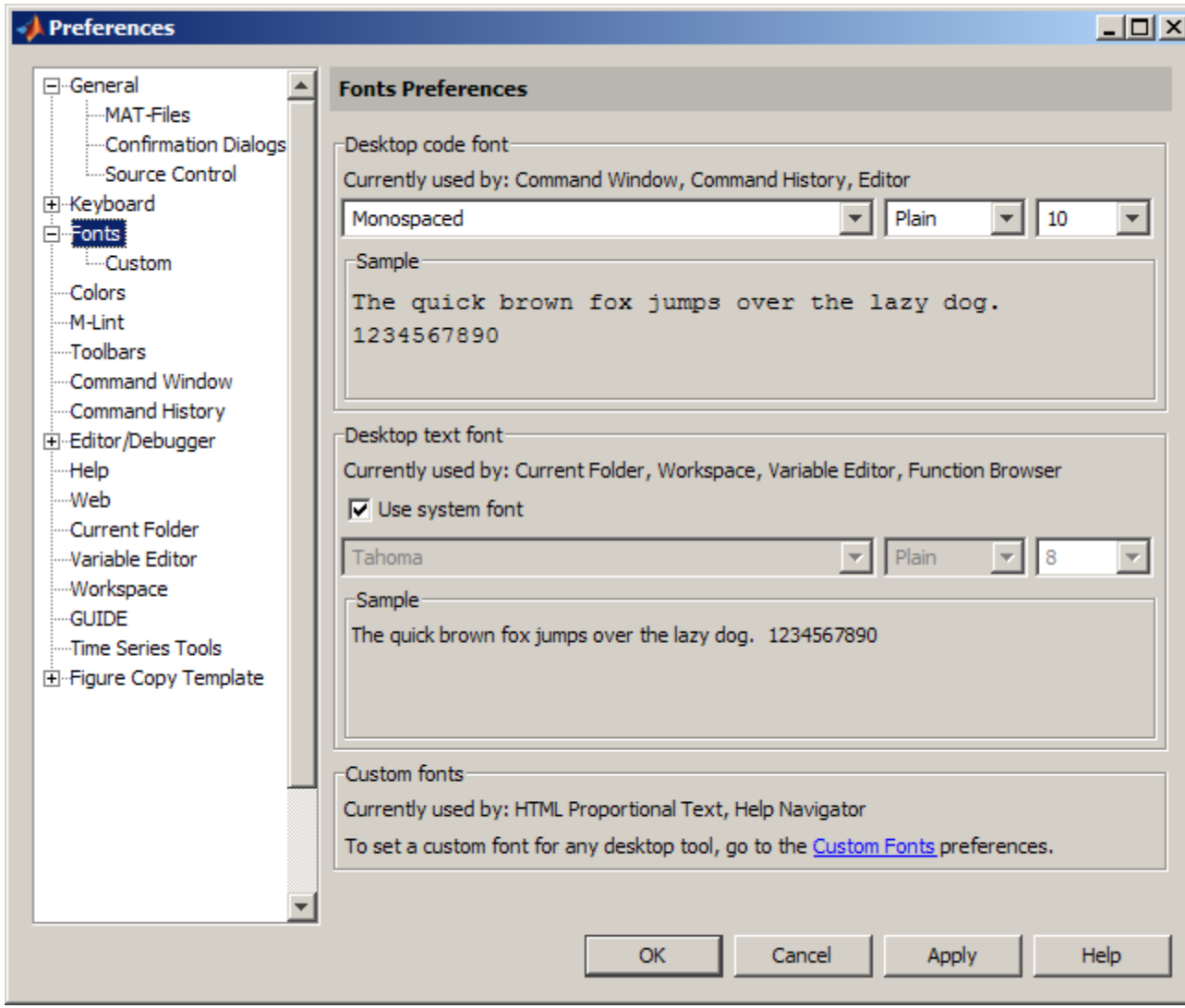
- A font for all the tools that primarily display code, such as the Command Window
- A font for all the tools that display text, such as the Current Folder
- Custom fonts, including a font for all the tools that use HTML Proportional text, such as the Help display pane and the MATLAB Web browser

If you want, you can separately specify the font for each desktop tool.

Select the font characteristics from the lists shown. For font size, you can type or select a size. You can type a size not shown as a choice in the drop-down menu.

You can set some font options differently for printing — see "Printing and Page Setup Options for Desktop Tools" on page 2-119.

For information about making additional fonts available to MATLAB, see “Making Fonts Available to MATLAB Tools on Windows Platforms” on page 2-150.



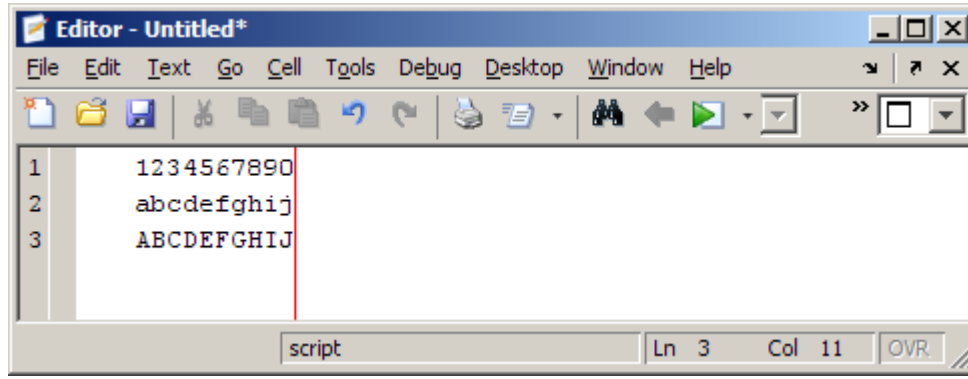
Desktop Code Font and Desktop Text Font. You specify separate font characteristics for tools that primarily display code (**Desktop code font**), such as the Command Window, and tools that primarily display text (**Desktop text font**), such as the Current Folder browser. (For other tools, such as the Help display pane and the MATLAB Web Browser you use custom fonts—see "Custom Fonts Preferences" on page 2-146.)

Many users prefer that code display in a monospace font to provide better alignment, and to distinguish it from other 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).

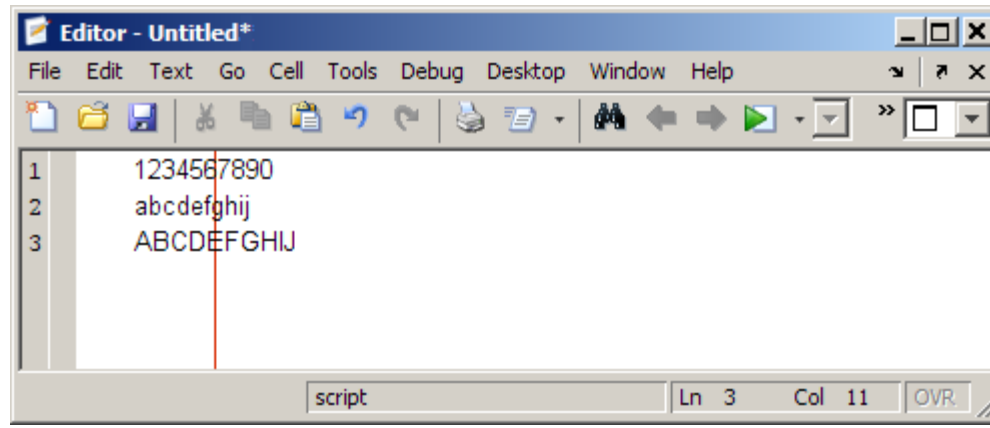
Typically, users specify a proportional font for tools that display little or no code. You use the desktop text font preference to set just one preference that applies to all tools that display little or no code. If you want to use the system font as the desktop text font, select **Use system font**.

The following illustrations show how the Editor looks using a monospace font compared to a proportional font. A monospace font is useful when you care about alignment, while 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'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. SanSerif. Each line contains 10 characters, but the length of each line differs. The Editor's right-hand text limit, at column 10, is not relevant.



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 font 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.

Factory Default Font Settings

The following table lists the factory default code and text font settings, and the tools that use those font settings. If you have made changes but want to revert to the default font characteristics, make changes using the values in the table.

Font Type	Factory Default Characteristics and Sample	Tools Using Font Type by Factory Default
Desktop code font	Monospaced, Plain, 10 point `Sample text font`	• Command History • Command Window • Editor (which also applies to the Shortcuts Editor)
Desktop text font	Your system's current font.	• Current Folder browser (which also applies to the Path browser) • Function Browser in the Command Window and Editor • Help Navigator • Workspace browser • Variable Editor

See Also

"Specifying Options for MATLAB Using Preferences" on page 2-126

Custom Fonts Preferences

Use custom font preferences to specify the font for HTML Proportional Text, and to override font settings for individual desktop tools. Desktop tools otherwise use the settings that the **Fonts** pane specifies. The **Fonts** pane is described in "Desktop Fonts Preferences" on page 2-142.

HTML Proportional Text is the default font for the following tools and portions of tools:

- The MATLAB Help display pane
- The small window that opens for the help on selection feature.
- The MATLAB Web browser— which displays the HTML output generated from publishing
- The Profiler

- The Function Browser function help
- Extended M-Lint messages

To specify custom fonts preferences:

1. Select **File > Preferences > Fonts > Custom**.

   The **Fonts Custom Preferences** pane appears.

2. Select the tool for which you want to specify custom fonts from the **Desktop tools** list. The type of font the tool currently uses appears under **Font to Use**.

3. For **Font to Use**, select one of the following:
   - **Desktop code**, which you can customize using the **Fonts** pane.
   - **Desktop text**, which you can customize using the **Fonts** pane.
   - **Custom**, and then specify the font characteristics.

4. Click **OK**.

Note If you change the font style (for example, to bold or italic) for **HTML Proportional Text**, it has no effect. If you change the font size, it affects both noncode and code text for tools using the HTML Proportional Text font.

Changing the Font — Example

This example:

- Changes the settings for the desktop code font from the factory default settings. (See Factory Default Font Settings on page 145.)
- Changes the Command History font preference so that it uses the desktop text font instead of the code font.
- Specifies a custom font for the Current Folder browser.

1. Select **File > Preferences > Fonts**.

2. Under **Desktop code font**, select `Times New Roman`, `Plain`, `14` point.
3. Under **Desktop text font**, select **Use system font**.
4. Click **Apply**.
5. Make the Command History window use the desktop text font:
   a. Click the **Custom Fonts** link.
   b. From **Desktop tools**, select `Command History`.
   c. Select the **Desktop text** radio button.
   d. Click **Apply**.
6. Apply a custom font to the Current Folder browser:
   a. From **Desktop tools**, select `Current Folder`.
   b. Select the **Custom** radio button.
   c. Select `Arial Narrow` and `Plain`, and then type `11` in the size field.
   d. Click **OK**.

The following table details the results of the changes.

Tool	Font Type	Font Characteristics
Command Window	Desktop code	Times New Roman® font, Plain, 14 point
Command History	Desktop text	Same as your current system font, which appears in the dimmed fields below the **Use system font** check box.
Editor	Desktop code	Times New Roman font, Plain, 14 point
Help Navigator	Custom	SanSerif, Plain, 8 point
HTML Proportional Text	Custom	SansSerif, Plain, 10 point

Tool	Font Type	Font Characteristics
Current Folder browser	Custom	Monotype Corporation Arial® Narrow font, Plain, 11 point
Workspace browser	Desktop text	Same as your current system font, which appears in the dimmed fields below the **Use system font** check box.
Variable Editor	Desktop text	Same as your current system font, which appears in the dimmed fields below the **Use system font** check box.
Function Browser	Desktop text	Same as your current system font, which appears in the dimmed fields below the **Use system font** check box.

Notice that on the Fonts preferences pane, the descriptive text reflects your changes. For example, under **Desktop text font**, the text reads, `Currently used by:  Command History, Workspace, Variable Editor, Function Browser`.

See Also. For information about how MATLAB stores preferences, and to get help for other preferences, see "Specifying Options for MATLAB Using Preferences" on page 2-126.

Antialiasing for Desktop Fonts on Linux and UNIX Platforms

To give the desktop a smoother appearance on Linux[7] and UNIX[8] platforms, select the antialiasing preference on the **Preference > Fonts** pane. The preference applies to all fonts.

7. Linux is a registered trademark of Linus Torvalds.

8. UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Making Fonts Available to MATLAB Tools on Windows Platforms

Under the following circumstances, consider updating fonts on your Windows platform as described:

- If a new compatible font is available to MATLAB

  A compatible font on Windows platforms for desktop components (such as the Command Window), figure windows, and uicontrols is one compatible with TrueType® and Microsoft OpenType® fonts. A compatible font for graphics objects, such as `xlabel`, `ylabel`, `title`, and `text` is a bitmapped font.

  Use the Windows Control Panel to install the font. Then, restart MATLAB so that it can use the font.

- If you open a file created by someone else, and you see boxes or meaningless symbols instead of text.

  When you see boxes or meaningless symbols instead of text, it is probably because you are using different language fonts from the file creator. This situation can occur, for example, if you open a file created by someone whose native language is Japanese and your native language is English. The Japanese user is probably using fonts for East Asian languages and you are not.

  In the Windows Control Panel, find the region and language options, and then install the supplemental files for East Asian languages.

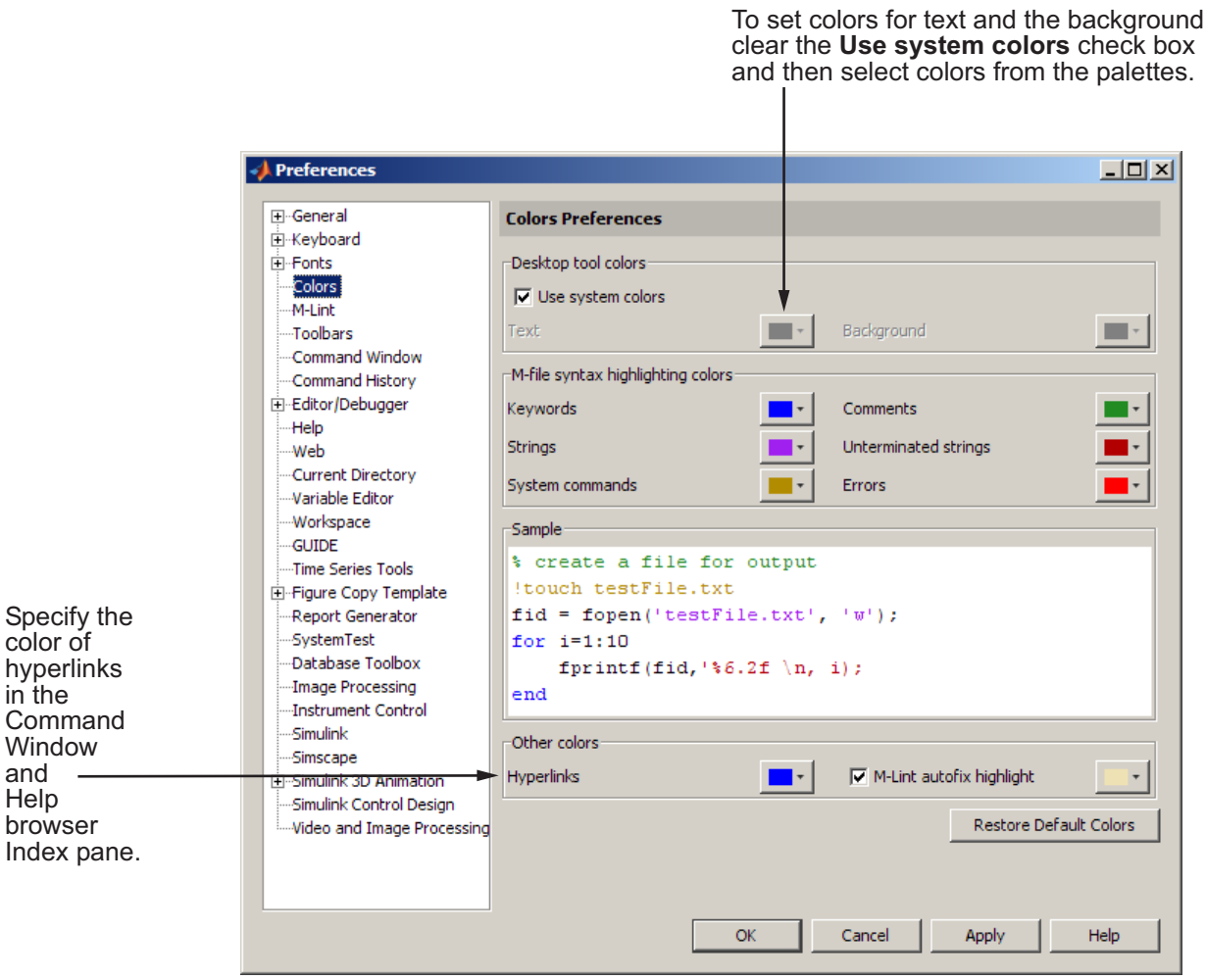
For more information, refer to the Windows help.

Setting Colors Preferences for Desktop Tools

- "Setting Colors Used in Desktop Tools" on page 2-151

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 2-119.



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, Microsoft 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.

Gray Background Color. For some UNIX[9] platforms, there is a gray background color for desktop tools, such as the Editor. 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.

M-File Syntax Highlighting Colors

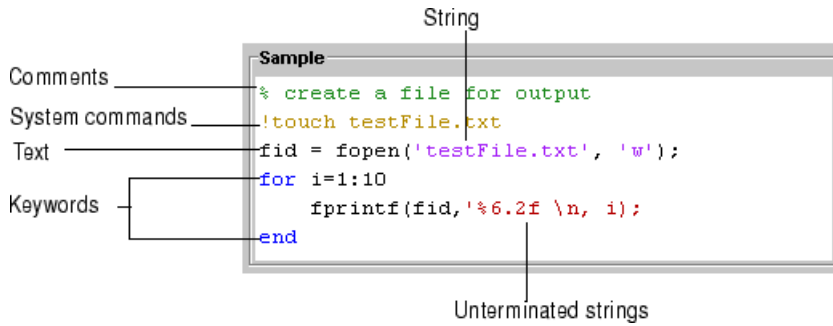
In the Command Window, Command History, Editor, and Shortcuts callback area, MATLAB conveys syntax information via different colors to help you easily identify elements, such as `if/else` statements. This is known as syntax highlighting.

9. UNIX is a registered trademark of The Open Group in the United States and other countries.

In the Command Window, only the input you type is highlighted; the output from running MATLAB functions is not highlighted.

Note To set syntax highlighting colors for files you create for the TLC, C, C++, or Sun Microsystems Java languages, or for HTML, and XML, use the Editor/Debugger language preferences. Select **File > Preferences > Editor/Debugger > Language**. Click the **Help** button to get information on "Setting Language Preferences" on page 8-20 in the online documentation.

When you choose a color under the **M-file syntax highlighting colors** area, 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 blue.
- **Comments** — All lines beginning with a `%`, designating the lines as comments in MATLAB, are 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 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 Fundamentals documentation.

- **Unterminated strings** — A single quote without a matching single quote, and whatever follows the quote, are maroon. This might alert you to a possible error.
- **System commands** — Commands such as `!` (shell escape) are gold.
- **Errors** — Error text that appears after you run code, including any hyperlinks, is red.

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

Note Syntax highlighting for M-files is enabled by default. If you find it is disabled, follow these steps to reenable it:

1. Select **File > Preferences**, and then click **Language**.

   The Editor/Debugger Language Preferences dialog box opens.

2. In the **Language** drop-down menu, select `MATLAB`.

3. In the **Syntax** area, select **Enable syntax highlighting**.

   Note that from this area, you can access the Colors Preferences dialog box, by clicking **Set syntax colors**.

4. Click **Apply**.

Other Colors

Specify the color for **Hyperlinks**, which applies to links in the Command Window. If you use a dark background color for the Command Window, 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. Use the palette to change the highlight color. For more information, see "Using M-Lint Automatic Code Analyzer in the Editor" on page 8-120.

See Also

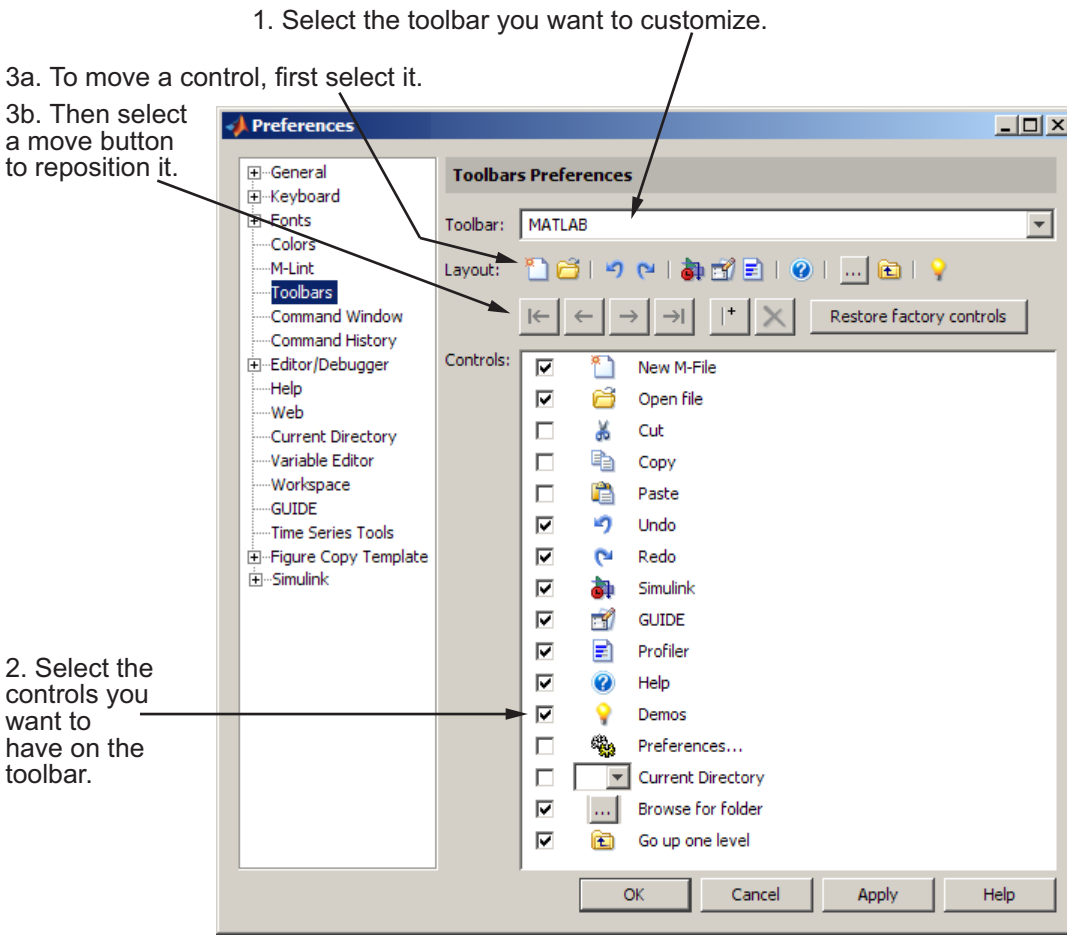
For information about other preferences and how MATLAB stores preferences, see "Specifying Options for MATLAB Using Preferences" on page 2-126.

Setting Toolbars Preferences for Desktop Tools

You can customize some toolbars in the MATLAB application using Toolbars preferences. You can add and remove buttons and other controls, as well as change their position on the toolbar.

To access Toolbars preferences, select **File > Preferences** and select **Toolbars**.

The following figure summarizes how to modify toolbars.



Following is an example of a customized MATLAB desktop toolbar.

Example of MATLAB desktop toolbar customized using preferences.

To customize a toolbar, follow these steps:

1. Select **File > Preferences > Toolbars**. You also can access Toolbars Preferences by right-clicking a toolbar and selecting **Customize** from the context menu.

2. In the resulting Toolbars preferences pane, choose the toolbar to modify by selecting it from the **Toolbar** list:

   - `MATLAB` — the toolbar in the MATLAB desktop
   - `Editor` — the toolbar in the MATLAB Editor
   - `Editor Cell Mode` — a specialized toolbar in the Editor. For more information, see "Using Cells for Rapid Code Iteration and Publishing Results" on page 8-185.
   - `Workspace`, the toolbar in the Workspace browser
   - `Current Folder`, the toolbar in the Current Folder browser

   The controls for the selected toolbar appear in the **Layout** and **Controls** sections of the Toolbars Preferences pane.

3. Customizing controls:

   - To add controls — choose a control to add to the toolbar by selecting its check box in the **Controls** section of the dialog box. For example, select the `Demos` check box to add its button to the toolbar.
   - To remove controls — choose a control to remove from the selected toolbar by clearing its check box in the **Controls** section of the dialog box. For example, to remove the Cut, Copy, and Paste toolbar buttons, clear the check box for each.
   - Restoring defaults — if you want to show the controls that appeared on the selected toolbar and in the same order as when MATLAB was first installed, click **Restore Factory Controls**.

   The **Layout** section of the dialog box displays the controls you chose for the toolbar, in the order they will appear.

4. Rearrange the order of the controls and separator bars on the selected toolbar using the **Layout** area:

   - Drag a control or separator bar to another position.

- Select a control or separator bar; then use one of the move buttons. For example, select the Demos button  , and then the Move to the End button  . The Demos button moves to the right end.
- Add a separator bar after the selected control using this button:  . To remove a separator bar, select it, and then use the Remove button  . You can use the Remove button to delete any control selected in the **Layout** section of the dialog box.

The **Layout** area displays the controls in the order you specified.

5 Click **Apply** or **OK**. The toolbars in the desktop and Editor update to reflect the changes you made.

For information about hiding, showing, and moving toolbars, see "Using Toolbar Features" on page 2-112.

Accessibility

In this section...

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 2-162
- 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 the JAWS® application software from Freedom Scientific BLV Group:

- MATLAB (many sections, but not the function reference pages (however, M-file help is accessible))
- Spreadsheet Link™ EX
- 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:

- The JAWS screen reader application software 5.0, 6.0, and 7.0 for Windows platforms 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 the JAWS application software , you can skip this section.

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

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 software from Sun Microsystems is installed automatically when you install MATLAB.

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

Setting Up JAWS Software

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

Setting up JAWS software 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 Java Access Bridge Software 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 Java Access Bridge to your Windows path.

Note This procedure assumes the **Start** button in your Windows preferences 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 the JAWS software is already running, exit and restart.

Note The JAWS software 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\`

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

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

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

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

Testing

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

1. Start the JAWS software.
2. Start MATLAB.

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

Troubleshooting

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

JAWS Software Does Not Detect When Installation of the MATLAB Software 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, the JAWS software 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 the JAWS software to perform the installation.

JAWS Software Stops Speaking

When many desktop components are open, the JAWS software sometimes stops speaking for MATLAB.

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

Command Output Not Read

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

This problem is likely to be caused by the way you have keyboard shortcuts defined in MATLAB. Keyboard shortcuts activate a certain command in MATLAB when you press a key or combination of keys. By default, MATLAB assigns the **Up Arrow** and **Down Arrow** keys to the `Previous History` and `Next History` commands in the Command Window. However, JAWS needs these two keys to be assigned to the `Cursor Up` and `Cursor Down` commands to be able to automatically read the results of commands.

You can check the shortcut setting for the **Up Arrow** key simply by pressing this key while in the Command Window. If the cursor moves up on the screen, then your settings are correct and this is not the cause of the problem. If the **Up Arrow** key displays your most recent command, or performs any action other than moving the cursor up, then follow the steps below to correct the problem.

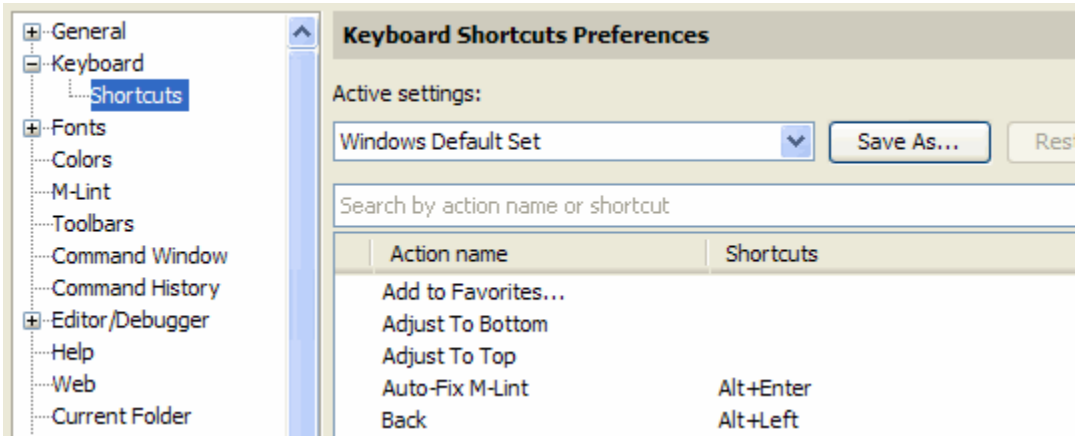
Note Reassigning the arrow key shortcuts means that you will no longer be able to use these keys to activate the `Previous History` and `Next History` commands. You can, however, assign any two unused keys to these actions. See "Reassigning Shortcuts for Previous History and Next History" on page 2-170 in the documentation below.

For more information on either of the procedures described below, see "Performing Desktop Actions Using Keyboard Shortcuts" on page 2-72 in the "Desktop Tools and Development Environment" documentation.

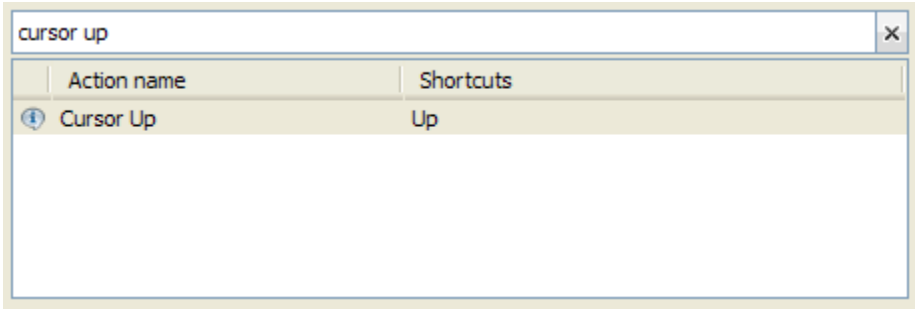
Assigning Shortcuts for Cursor Up and Cursor Down. Follow these steps to make the **Up Arrow** key a shortcut to `Cursor Up` and **Down Arrow** a shortcut to `Cursor Down`. You only need to execute this procedure once. When you close the dialog box with the **OK** button or the **Apply** and then **Cancel** buttons, MATLAB saves your settings and restores them in your next MATLAB session.

1 Begin by opening the MATLAB **Preferences** dialog box and clicking **Keyboard** and then **Shortcuts**. This displays the **Keyboard Shortcuts Preferences** dialog box.

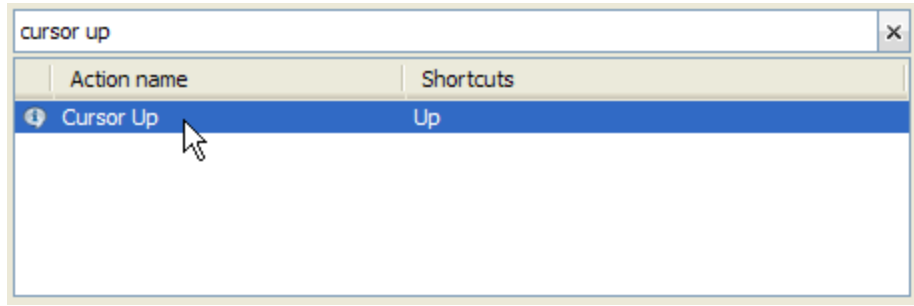
Near the top, the **Action name** and **Shortcuts** columns list all available actions and the shortcut keys currently assigned to them. Above **Action name**, locate the filter field that reads "Search by action name or shortcut" in greyed-out text:



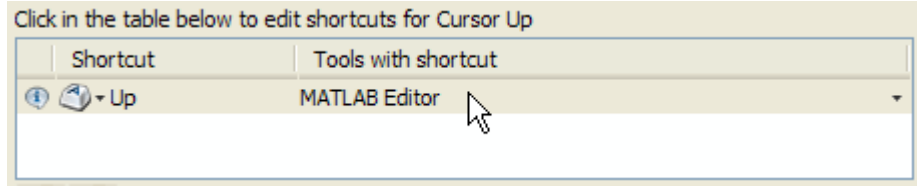
2 Enter the words "cursor up" in this field. The **Action name** and **Shortcuts** columns should now show only the one selected action (`Cursor Up`) and any shortcut keys that have been assigned to that action (`Up`, in this case):



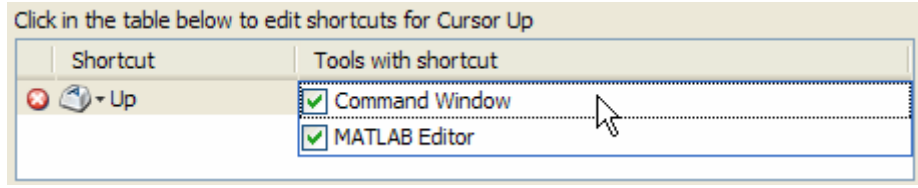
Click the line that shows this action-to-shortcut key association:



3 The next panel down shows that the `Up` (i.e., **Up Arrow**) shortcut is already assigned to the selected action (`Cursor Up`). However, as shown in the **Tools with shortcut** column, this association applies only in the MATLAB Editor, and not in the Command Window. Click the text under **Tools with shortcut** that reads `MATLAB Editor`:



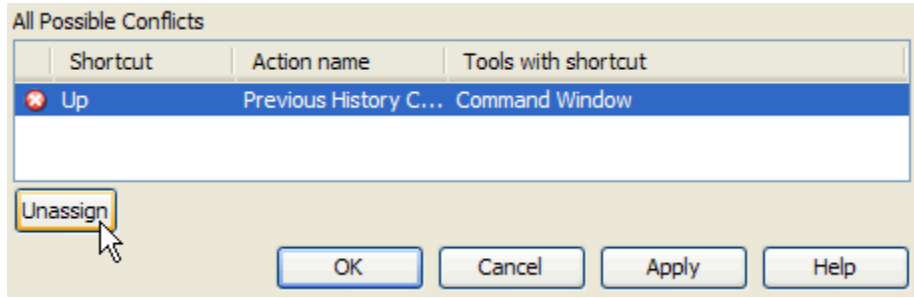
4 This displays a checkbox for each application in which you can associate the **Up Arrow** key with `Cursor Up`. Click the line that reads Command Window to put a check in the checkbox for that selection:



Move the cursor away, and the **Tools with shortcut** column now reads **All Tools**.

5 The panel below that, labeled **All Possible Conflicts**, shows that the shortcut you have just established conflicts with an existing shortcut in the Command Window. You now have two actions, `Cursor Up` and `Previous`

`History Command`, assigned to the same key (**Up Arrow**). To resolve this conflict, select the line that shows `Previous History Command`, and then click the **Unassign** button:

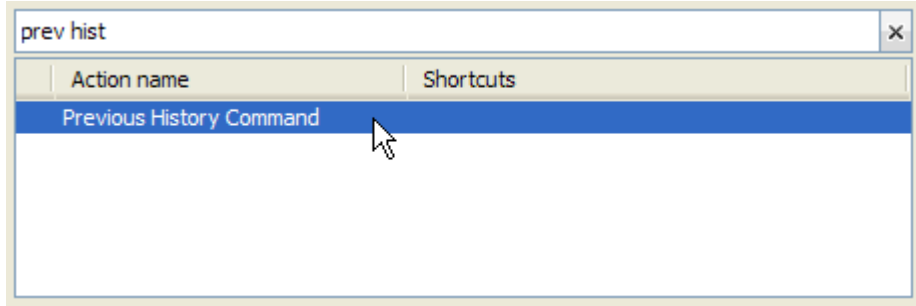


Click **Apply** to make this setting active.

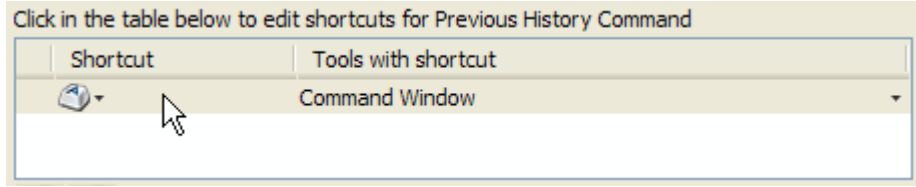
6 Repeat steps 2 through 5 to assign the **Down Arrow** key to the `Cursor Down` command in the Command Window. This also removes the existing shortcut between **Down Arrow** and `Next History` command.

Reassigning Shortcuts for Previous History and Next History. At this point, you have made assignments for the `Cursor Up` and `Cursor Down` desktop actions. The previously defined shortcuts for the `Previous History` and `Next History` commands have been removed. If you would like to retain these latter two shortcuts, you need to assign new keys to them. To do this, follow the steps below:

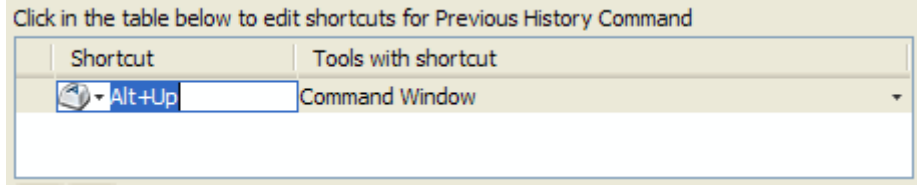
1 Type “previous history” in the box above **Action name**. Note that just the abbreviated “prev hist” will suffice. Next, click the line below **Action name** that reads `Previous History Command`:



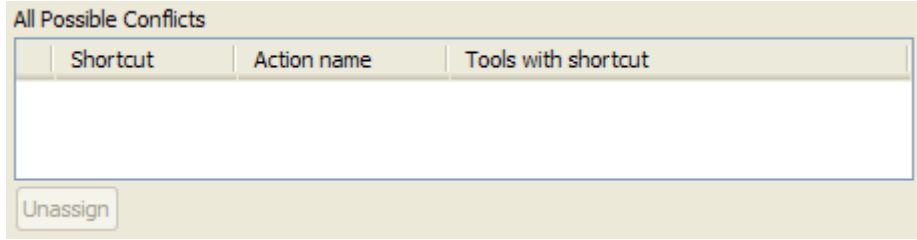
2 In the next panel down, click the line that displays the icon. This opens a key entry field at this location:



3 Press the key or key combination you want to use as the shortcut for the `Previous History` action. For example, if you want **Alt+Up Arrow** to be the shortcut for `Previous History`, hold down the **Alt** key and press the **Up Arrow** key:



4 View the **All Possible Conflicts** panel to see if there are any conflicts with this assignment. If there are no conflicts, click **Apply** to make the setting active. If you do have conflicts, then either choose a different key to use as the shortcut, or see the documentation on "Evaluating and Resolving Keyboard Shortcut Conflicts" on page 2-84 in the "Desktop Tools and Development Environment" documentation.



5 Repeat steps 1 through 4 to assign a shortcut key to the `Next History` command in the Command Window.

Some GUI Menus Are Treated as Check Boxes

For some GUIs (for example, the figure window), menus are treated by the JAWS software 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, the JAWS software 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, the JAWS software ignores the label of the field. However, the JAWS software will read any text in the text box.

3

Running Functions — Command Window and History

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 Command Window is where you run (execute) MATLAB language statements, while the Command History is a log of the statements you have run.

Using The Command Window

In this section...

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 watch the 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 visible, select one of the following:

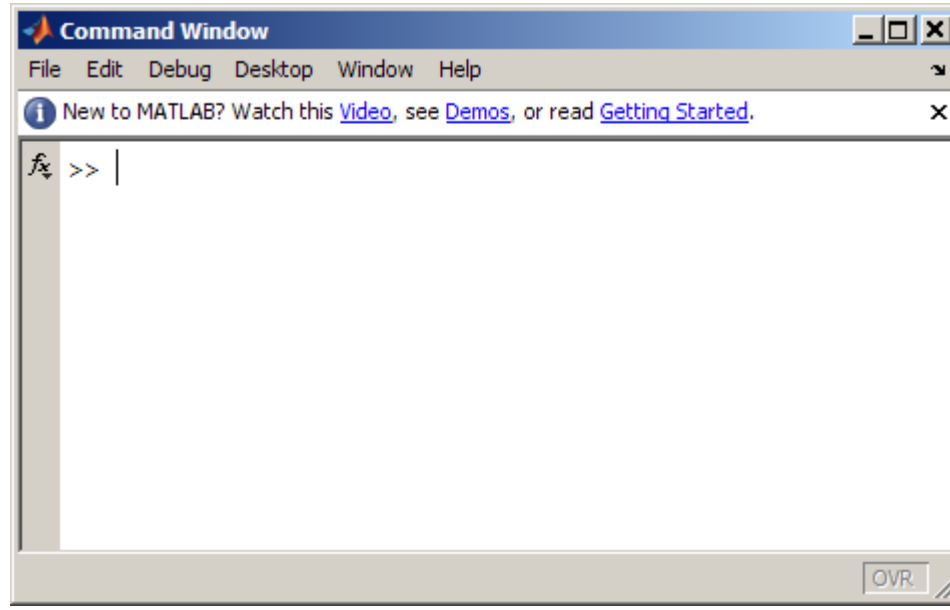
- **Desktop > Desktop Layout > Command Window Only**.

  The desktop contains only the Command Window, as shown in the image that appears after this list. MATLAB appears similar to how it looked in older versions.

- **Desktop > Desktop Layout > All but Command Window Minimized**.

  All tools open and are minimized in the desktop, except the Command Window, which is maximized. The Editor also remains maximized if it was open and contained a document at the time you chose the **All but Command Window Minimized** menu item.

For more information, see "Opening and Arranging Desktop Tools" on page 2-6.



To restore the desktop to the factory default arrangement, select **Desktop > Desktop Layout > Default**.

Using the 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 8, “Editing and Debugging M-Files”

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

Changing the Way the Command Window Looks

The Command Window is a desktop tool so you can move it, resize it, and change its font the same way you would any desktop tool. For more information, see “Opening and Arranging Desktop Tools” on page 2-6 and “Setting Fonts Preferences for Desktop Tools” on page 2-141.

There are other ways you can change the display of the Command Window, including hiding the message bar and the Function Browser button. To make changes, select **File > Preferences > Command Window** and use the **Display** options. For more information, click the **Help** button in the Preferences dialog box.

Running Functions and Programs, and Entering Variables

In this section...

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, the MATLAB software responds with

```
A =

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

To run a function, it must be in the current folder or in a folder on the search path. By default, functions included with MATLAB are on the search path. Therefore, you do not need to do anything special to run functions provided with The MathWorks products.

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
```

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

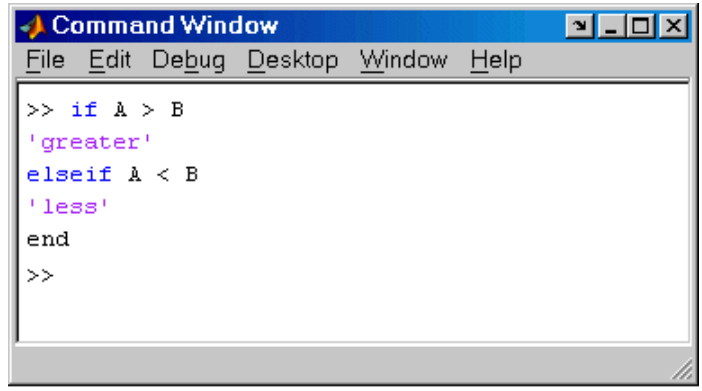
To find the name of a function you want to run that was provided by The MathWorks, use the function browser—see "Finding Functions Using the Function Browser" on page 3-37.

What Is a Statement?. All the information you type before pressing **Enter** or **Return** is known as a statement. Statements can include:

- Variable assignments: For example, `a = 3`
- Functions and their arguments: M-files that can accept input arguments and return output arguments, for example, `magic`.
- 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`.

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 That Were Not Provided By The MathWorks

You can run M-files, files that contain code in the MATLAB language, that you created or that were created by other users of MATLAB. The M-file must be in the current folder or on the search path before you run it or you get an `Undefined function or variable` error or a `File not found` error. For more information, see "Determining If MATLAB Can Access a File" on page 6-40. When the M-file is in the current folder or on the search path, you run it the same way that you would run functions supplied with MATLAB. Type the name of the M-file in the Command Window, complete the statement by adding arguments, and press **Enter** or **Return**.

For an M-file script, you can also use the `run` function and specify the full pathname to an M-file script.

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 file name and press **Ctrl+Enter**. The offending M-file opens in the Editor, 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 runs when the previous statement finishes.

Stopping Execution

You can stop execution of whatever is currently running by pressing **Ctrl+C** or **Ctrl+Break** at any time. On Apple 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 Microsoft Windows platforms rather than UNIX[10] 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 start MATLAB with the `-nodesktop` option.

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 platforms, 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 Microsoft Excel software and returns control to the Command Window so you can continue running MATLAB language statements.

10. UNIX is a registered trademark of The Open Group in the United States and other countries.

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 Microsoft Windows XP operating system, for example, the length of the argument list input to the bang command cannot exceed 8189 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 platforms. 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® (from Apple) directly from MATLAB. However, you can run the Apple Mac OS X `osascript` function from the MATLAB `unix` or `!` (bang) function to run AppleScript from MATLAB.

UNIX Platforms System Path for Running UNIX Programs from the MATLAB Software

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

Change the Current Folder in MATLAB. Change the current folder in MATLAB to the folder that contains the program you want to run.

Modify the UNIX System Path that MATLAB Uses. Add the folders 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:<myfolder>"
```

where `<myfolder>` is the folder that contains the program you want to run.

2. Start MATLAB.

3. In the MATLAB Command Window, type

```
!echo $PATH
```

The folder 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 the System Path When MATLAB Starts. If you want to add a folder 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:<myfolder>"
```

   where `<myfolder>` is the folder 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.
- Figure files (`.fig`) open in a figure window.
- Variables open in the Variable Editor.
- Models open in Simulink software.

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. Use `type` to display the M-file in the Command Window.

Displaying Hyperlinks in the Command Window

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

- Open an HTML page in a MATLAB Web 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.

Clicking a Hyperlink to Run MATLAB Functions

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 “Setting Colors Preferences for Desktop Tools” on page 2-150. For more

information, including examples, see the `matlabcolon (matlab:)` reference page.

Entering Statements in the Command Window

In this section...

Case and Space Sensitivity

Uppercase and Lowercase for Variables

With respect to case, the MATLAB language 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, file names, objects, and classes on the search path or in the current folder, 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 Microsoft Windows and UNIX[11] platforms because their file systems behave differently with regard to case.

11. UNIX is a registered trademark of The Open Group in the United States and other countries.

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 toSun Microsystems Java software do use mixed case and the M-file help and documentation accurately reflect that.

Examples. The folder `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 folder `second` after `first` on the search path, with the file `a.m` in `second`. The folder `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
```

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.

Press the **Esc** (escape) key to clear everything you entered on the current line.

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.

Entering Multiple Lines Without Running Them

To enter multiple statements on multiple lines before running any of them, use **Shift+Enter** or **Shift+Return** after typing a statement on a line. The cursor moves down to the next line, which does not show a prompt, where you can type the next statement. Continue for more lines. Then press **Enter** or **Return** to run all of the lines. This allows you to edit any of the statements before you pressing **Enter** or **Return**. This is an example:

```
>> a=1 % Press Shift+Enter to advance without executing this statement.
b=2    % Press Shift+Enter to advance without execution. You can edit this or the above line.
c=a+b  % Press Enter to execute all three statements.
```

MATLAB executes all three lines and returns the following:

```
a =
     1
b =
     2
c =
     3
>>
```

When you enter a paired keyword statement on multiple lines, such as `for` and `end`, you do not need to use **Shift+Enter**. You can use the typical process of pressing **Enter** after each line in the set to advance to the next line. MATLAB executes the keyword statement after complete it on the last line. For example

```
>> for r=1:5  % Press Enter. MATLAB advances a line where you continue the paired keyword statement.
a=pi*r^2      % Press Enter. MATLAB advances a line where you continue the paired keyword statement.
end           % Press Enter to execute the paired keyword statement.
```

MATLAB executes all three lines and returns the following:

```
a =
   3.141592653589793
a =
  12.566370614359172
a =
  28.274333882308138
```

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 Multiple-Line (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 8-43.

Recalling Previous Lines in the Command Window

Assuming you have not changed the default keyboard shortcuts for the arrow, tab, and control keys, you can recall, edit, and reuse functions you typed earlier by using these keys. 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 keys do not behave as documented here, check the actions currently assigned to them, as described in "Identifying Keyboard Shortcuts" on page 2-75.

Another way to view and access commands from the current and previous sessions of MATLAB is with the Command History window — see "Using the Command History Window" on page 3-61.

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, and so by default, you cannot use these keys 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, customize the keyboard shortcuts for the **Cursor Up** and **Cursor Down** actions in the Keyboard Shortcuts preferences. See also, "Customizing Keyboard Shortcuts" on page 2-78.

See Also

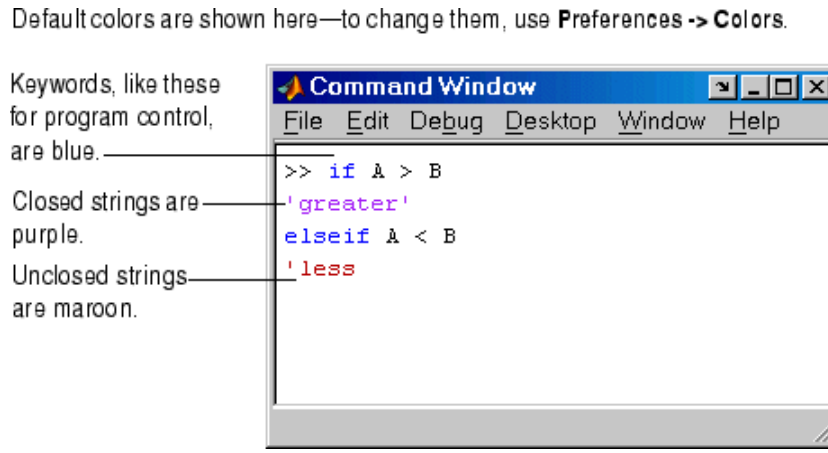
- "Assistance While Entering Statements" on page 3-20
- "Finding Text in the Command Window" on page 3-49

Assistance While Entering Statements

In this section...

Highlighting Syntax to Help Ensure Correct Entries

To help you better find elements, such as matching `if/else` statements, some entries appear in different colors in the Command Window. This is known as syntax highlighting. You can change the colors using preferences. Note that output in the Command Window does *not* appear with syntax highlighting, except for errors. Syntax highlighting is also available in other desktop tools. For more information, see "Setting Colors Preferences for Desktop Tools" on page 2-150



Matching Delimiters (Parentheses)

You can instruct the MATLAB software 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 use the delimiter matching feature, select **File > Preferences > Keyboard > Delimiter Matching**. This feature is also available in the Editor.

For more information, see "Setting Delimiter Matching Preferences" on page 2-139.

Completing Statements in the Command Window — Tab Completion

MATLAB helps you complete the names of known items as you type them in the Command Window so that you can avoid spelling mistakes and do not have to spend time looking up the information in other tools. These are the items for which MATLAB can complete the names:

- Functions or models on the search path or in the current folder
- MATLAB objects
- File names and folders, including object-oriented programming package and class folders

- Variables, including structures, in the current workspace
- Handle Graphics property names

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 "Setting Keyboard Preferences for Desktop Tools" on page 2-138.

Tab completion is also available in the Editor, but there are some slight differences in usage. See "Completing Statements in the Editor — Tab Completion" on page 8-45.

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

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
```

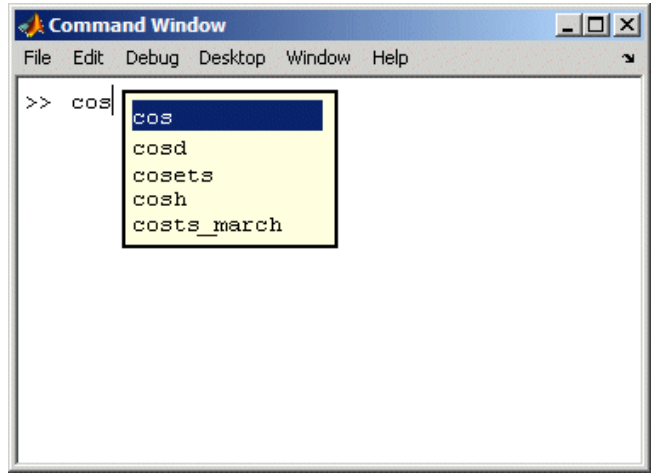
and press **Tab**, MATLAB completes `costs_march`. You can also select `co` or position the cursor after `co` and press **Tab** to complete `costs_march`.

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



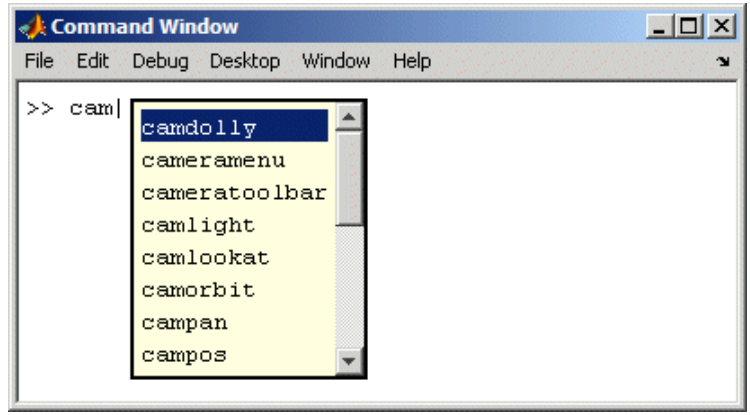
The resulting list of possible completions includes the variable name you created, `costs_march`, but also includes functions that begin with `cos`,

including `cosets` from the Communications Toolbox™ software, if it is installed on the system and on the search path in MATLAB. MATLAB completes variable names in the currently selected workspace, and the names of functions and models on the search path or in the current folder.

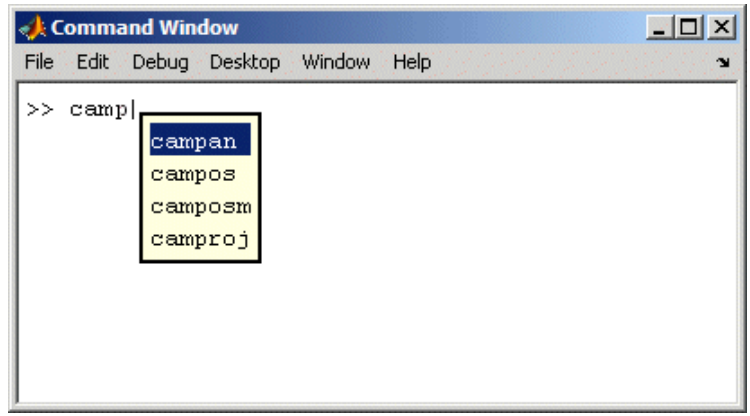
Continue typing to make your entry unique. For example, type the next character, such as `t` in the example. MATLAB selects the first item in the list that matches what you typed, in this case, `costs_march`. Press **Enter** (or **Return**) or **Tab** to select that item, which completes the name at 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 **Esc** (escape key). 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 Folders and File names

Tab completion works for folders and file names in MATLAB functions.

For example, type

```
edit d:/
```

and press **Tab**.

MATLAB displays the list of folders 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 folder on your `d` drive whose name begins with `mym`. Continue using tab completion to display and complete folder names or file names until you finish the `edit` statement.

Tab completion for folders and file names is not supported for functions you write.

Tab Completion for Class Folders and File Names

Tab completion for class folders (including @-folders), package folders, and file names works the same as for standard folders.

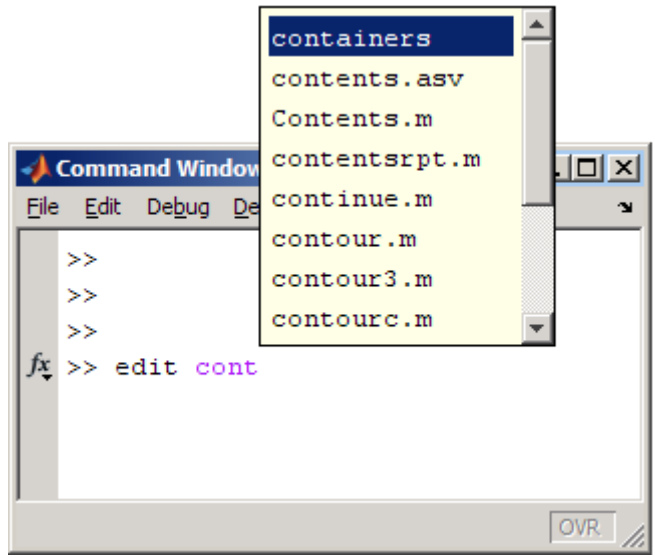
For example, consider the `containers` package in the `matlabroot` folder, which is the folder in which MATLAB is installed. The `containers` package contains a `Map` @- folder and a `Map.m` file. The folder structure appears as follows:

```
+containers\@Map\Map.m
```

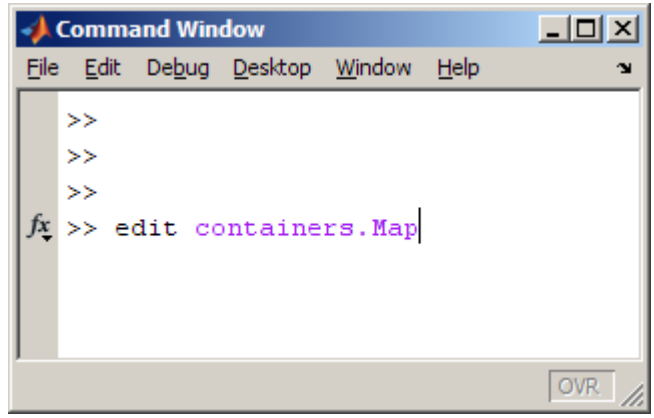
If you type:

```
edit cont
```

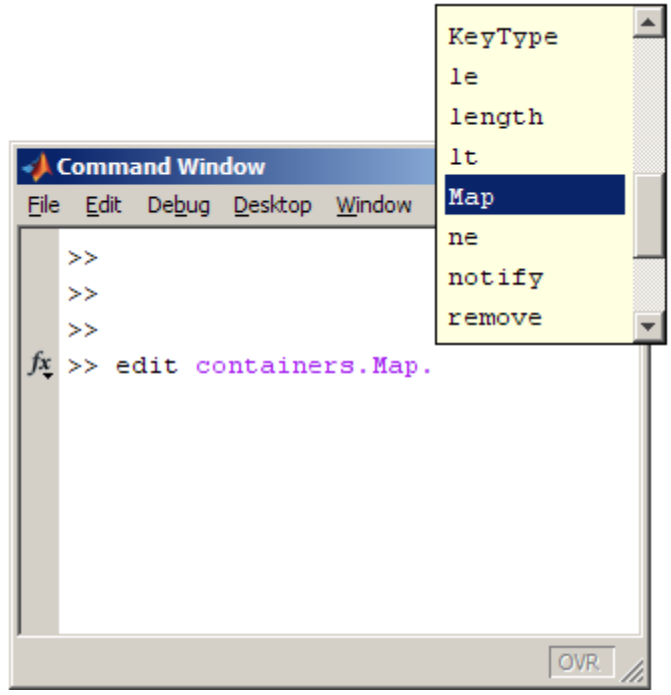
and press **Tab**, MATLAB adds characters to `cont` until MATLAB finds a character that is not common to any names on the MATLAB path. MATLAB displays a list from which to choose, such as the following



If you select `containers`, add a dot, and then press **Tab**, MATLAB displays



If you add a dot, press **Tab**, and scroll down a bit, MATLAB displays



Tab Completion for Structures

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

```
mystruct.
```

and then 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
```

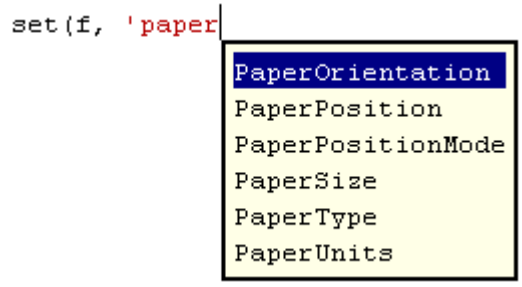
and press **Tab**, which completes the entry `mystruct.name`, where `mystruct` contains no other fields that begin with `n`.

Tab Completion for Handle Graphics Properties

Complete the names of Handle Graphics properties using tab completion, as in this 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.

Tab Completion for MATLAB Objects

You can use tab completion with MATLAB objects to select from available properties and methods.

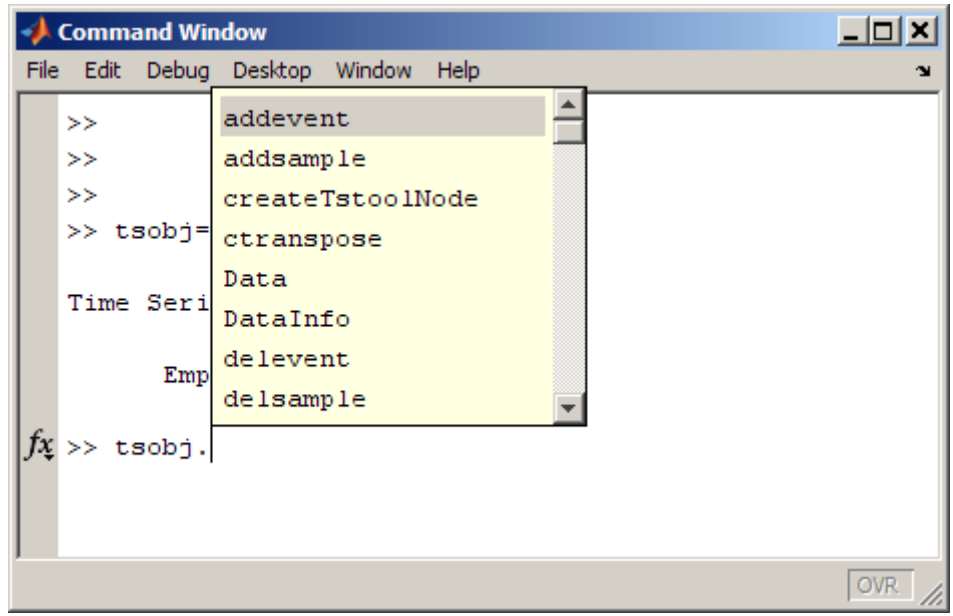
For example, create a time series object, `tsobj`.

```
tsobj = timeseries
```

Then enter the object name followed by a period separator (`.`), such as

```
tsobj.
```

Press **Tab**. MATLAB displays a list of properties and methods for the object, `tsobj`.



Viewing Function Syntax Hints While Entering a Statement

What Are Function Hints?

Function hints display allowable input arguments for a function while you enter a statement in the Command Window or Editor. Function hints:

- Appear in a temporary pop-up window.
- Are useful when you only need a reminder of the syntax for a function.
- Provide a link to the reference page for more information.
- Are available for all functions provided with MathWorks products. The syntax comes from the function reference page.
- Work for functions you create. The syntax comes from the function definition statement (first executable line) in the M-file. The M-file must be on the search path or in the current folder.

Function hints only display input argument syntax. They do not show output argument syntax. To access output argument syntax while using the function hints, clink the **More Help** link to view the reference page.

Basing Steps for Using Function Hints

These steps illustrate using function hints in the Command Window for the `size` function.

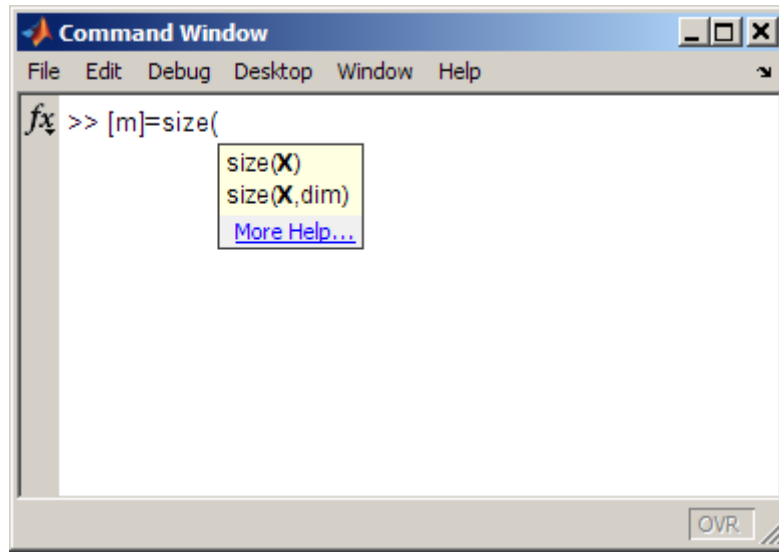
1 Type any output arguments and the equal sign. For example, type:

```
[m]=
```

2 Type the function name and the opening (left) parenthesis, and then pause. For example, type:

```
size(
```

A yellow pop-up window appears, displaying the syntax options for the function. In this example, the pop-up window indicates that you can enter a single argument, `X`, or two arguments, `X` and `dim`.



3 Type a variable name for the first input argument. You can type a variable for any argument that appears in bold in the pop-up.

Note Enter your variable names, and not the argument names shown in the pop-up window.

In this example, for the input argument `X`, type your variable:

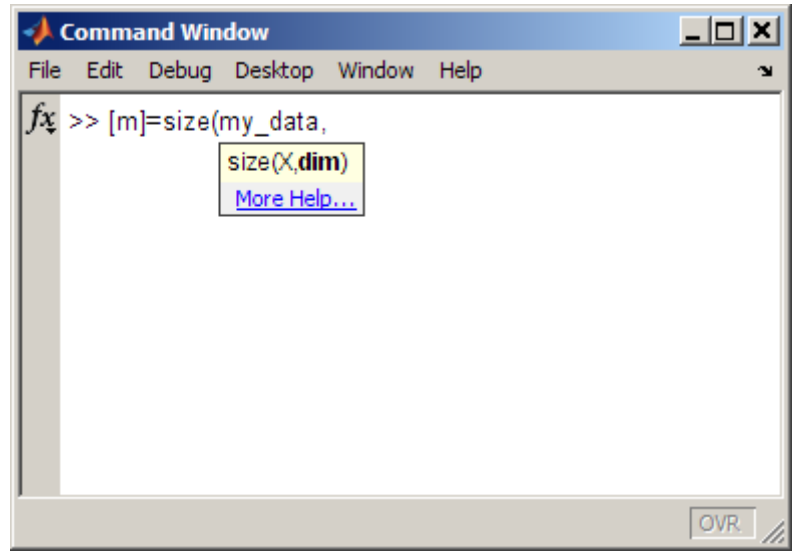
```
my_data
```

4 Enter more arguments:

a Type a comma after the argument you just entered.

The syntax options in the pop-up window change, based on the argument you just entered.

For this example, the pop-up window now shows only the `X,dim` option. `dim` is bold now, because you can enter it.



b Type the next input argument. It can be any argument that is bold in the pop-up.

For example, to run `size` for the third dimension, enter `3` for `dim`.

Continue entering more input arguments in the same way.

5 When you finish entering arguments, type the closing (right) parenthesis.

The pop-up window closes.

Interpreting Function Hints

- For an overloaded function name, the syntax for the method includes valid objects of the method currently in the workspace.
- MATLAB cannot always determine the appropriate hints correctly. Some allowable arguments may not appear, or could be in plain text when they should be bold.

Getting More Information While Using Function Hints

- Click **More Help** in the pop-up window. MATLAB replaces the pop-up window with a small help window containing the complete reference page for a function. See also "Help on Selection and More Help — Specifying Where the Help Displays" on page 4-19.
- Use function hints along with other methods for getting assistance, including tab completion, the Function Browser, and Help on Selection.

Note When you click anywhere, or move the cursor away from the arguments you are entering, the pop-up window automatically closes.

Modifying Statements While Using Function Hints

Modify your statement while the pop-up window is open, and the function hints change based on your modifications. For example, delete a variable you already typed and that argument appears in the pop-up window.

Closing the Pop-Up Window

- Click anywhere, or move the cursor away from the arguments you are entering.
- Close the pop-up window by pressing **Esc** (escape).
- After closing the pop-up window, reopen it by clearing the arguments you entered and clearing the left parenthesis. Then, enter the left parenthesis and pause to redisplay the hints.

Enabling or Disabling Function Hints

To prevent function hints from appearing, or to show them if they are not appearing, use keyboard preferences.

1. Select **File > Preferences > Keyboard**.
2. In **Function hints**, select the check boxes for the tools where you want hints to appear. Clear the check boxes for the tools where you do not want hints to appear.

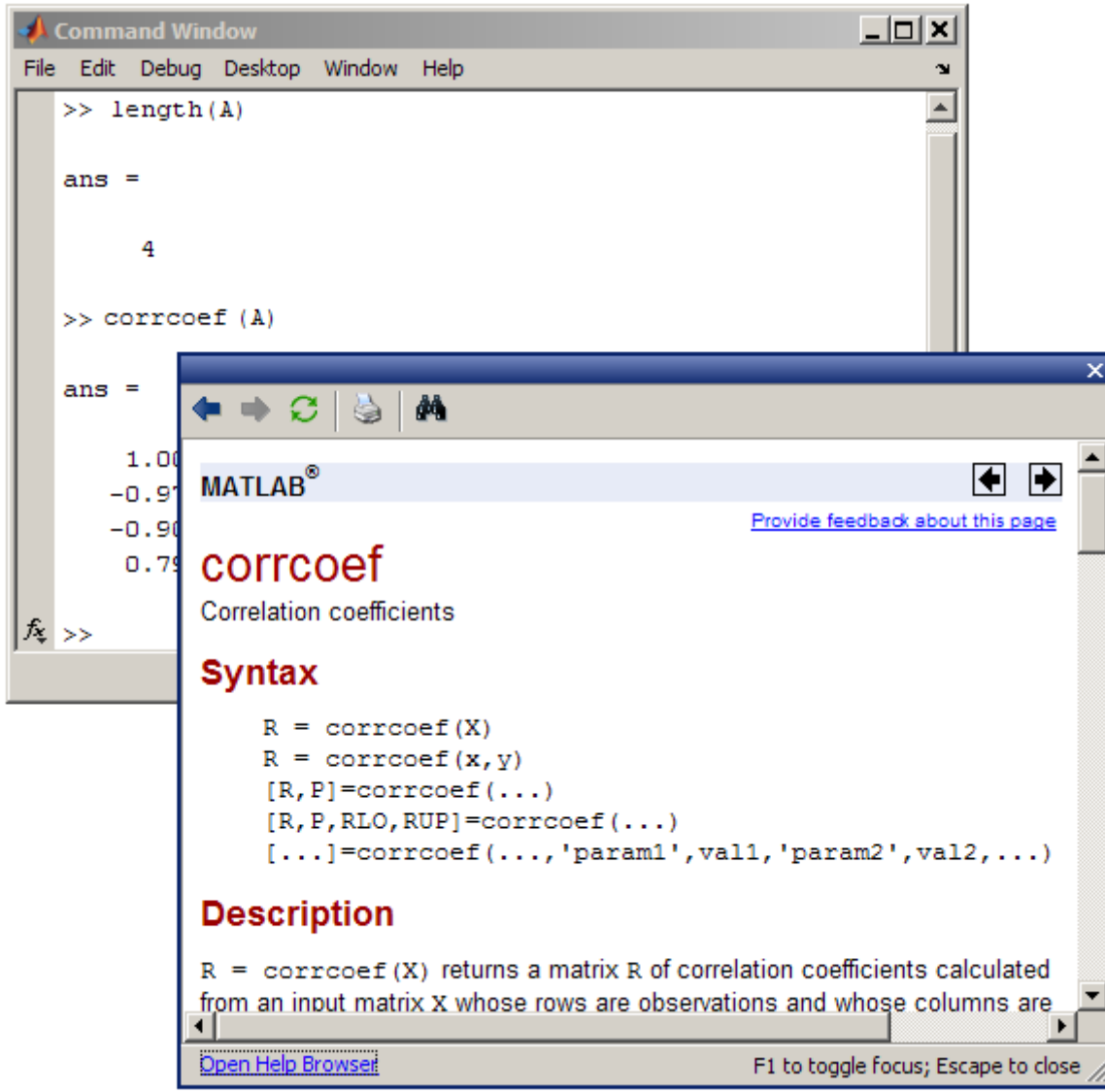
3 Click **OK**.

Getting Help for a Function Shown in the Command Window or Editor

From the Command Window or Editor, display the complete reference page for a function:

1 Right-click in a function name and select **Help on Selection**.

The reference page for the function opens in a small help window. The following illustration shows help on selection for the `corrcoef` function in the Command Window.



2 Close the small help window by pressing **Esc** (escape).

Customize the Help on Selection feature:

- Specify a preference so that help on selection opens the reference page in the Help browser instead of the small help window. To set the preference, select **File > Preferences > Help**.

  When help on selection opens the reference page in the Help browser, you cannot toggle focus between the reference page and the Command Window or Editor.

- Specify the products to look in. See "Filter by Product — Specifying Products Used in the Help Browser" on page 4-18

  and "Help for Overloaded Functions" on page 4-4.

- Change the small help window font. To change the font, select **File > Preferences > Fonts > Custom**. The small window uses the HTML proportional text font.

Finding Functions Using the Function Browser

What Is the Function Browser?

- Provides quick access to the syntax for a function and a description of the syntax.
- Helps you find the names of functions.
- Works in the desktop.

The Function Browser looks for functions using the reference pages for installed MathWorks products.

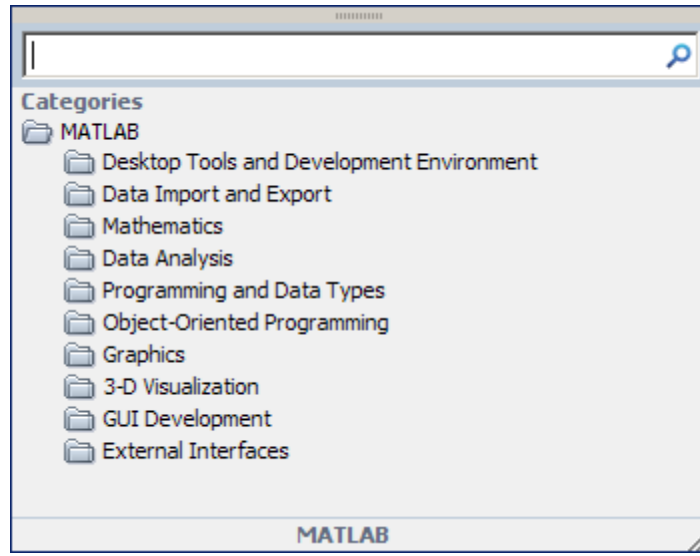
- You cannot use the Function Browser for blocks. Instead use the `doc` function or the Help browser.

- You cannot use the Function Browser to find functions you created or that other users provided. Instead, use “Finding Files and Folders” on page 6-20.

Basic Steps for Using the Function Browser

1 Open the Function Browser by selecting **Help > Function Browser**.

In the Command Window or Editor, another way to open it is by clicking the Browse for functions button,  . To show or hide the button, see “Customizing the Function Browser” on page 3-43.



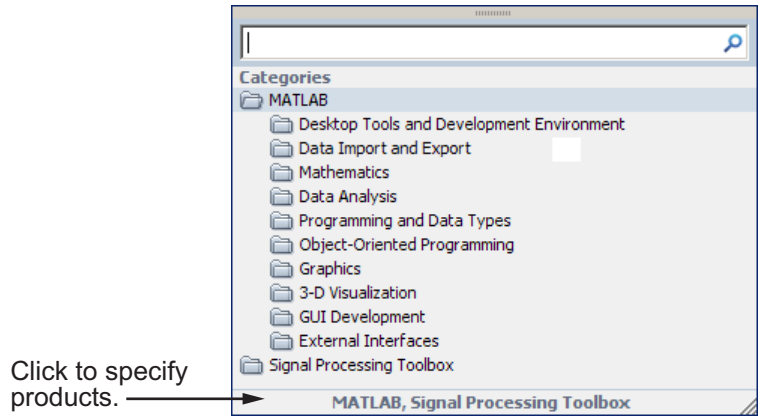
The Function Browser closes when you move the pointer outside of it. To keep the Function Browser open, drag it by the top edge to a different location.

After moving the Function Browser, access it by pressing **Shift+F1**. Close it by pressing **Esc**.

2 Restrict the products the Function Browser looks in by using the **Filter by Product** option in Help Preferences.

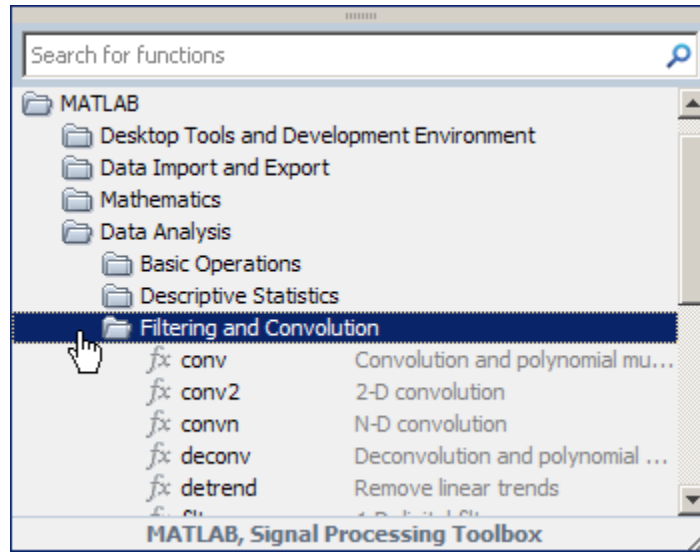
To access the preference, click the product area at the bottom of the Function Browser.

The following illustration shows the filter set to MATLAB and the Signal Processing Toolbox only. The

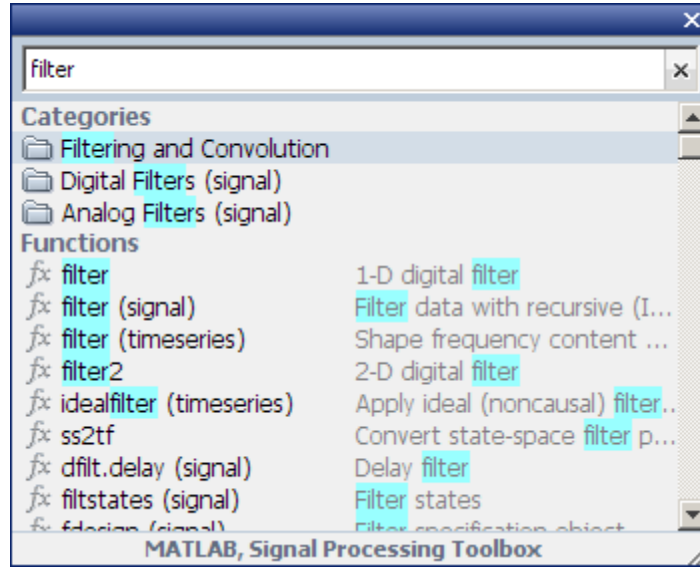


3 Find functions by browsing in the list of categories or by typing a search term. You can switch between these two options at any point.

The following illustration shows how you browse for functions by expanding categories of interest.

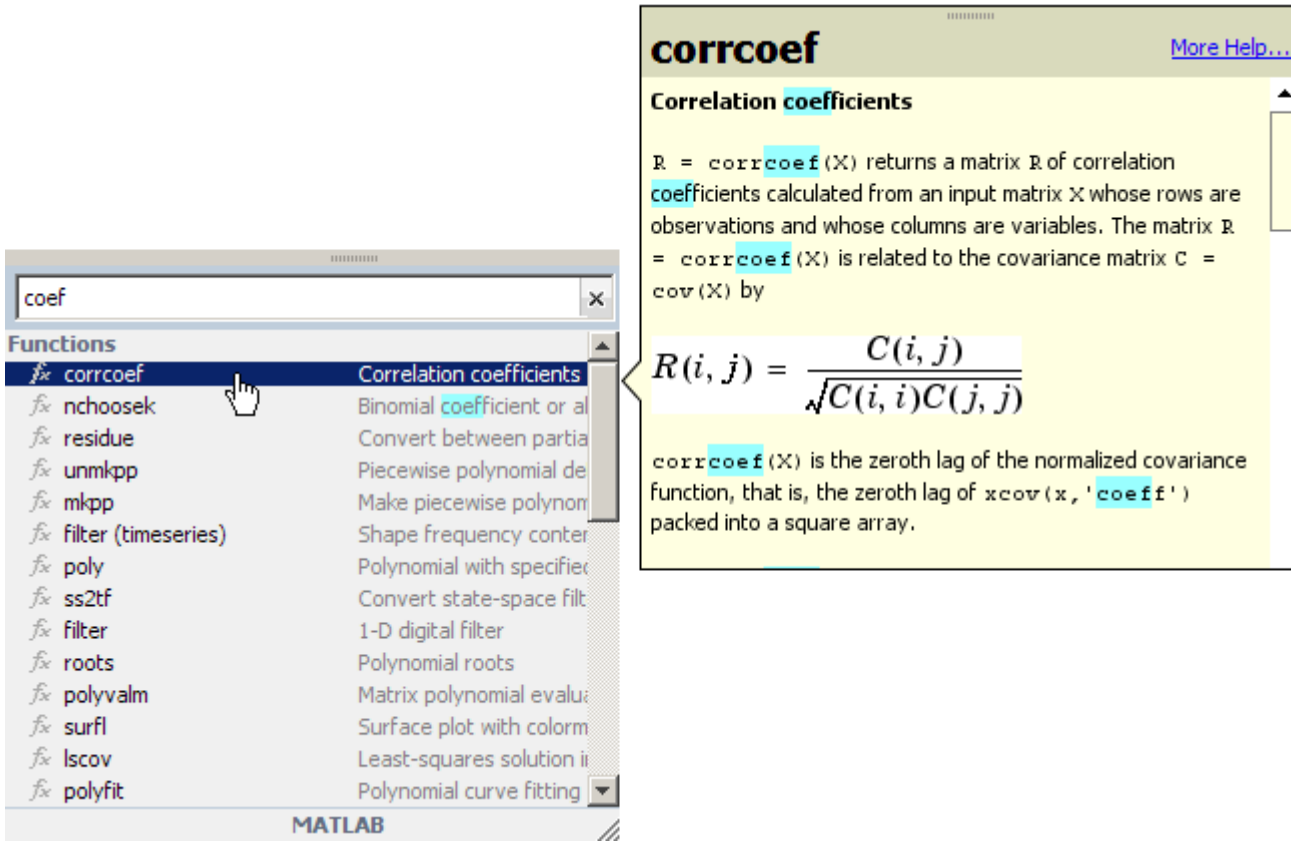


The following illustration shows results when you search, for example, for `filter`.



4 View more information about a function by moving the pointer over a function. A brief description for each of the syntax options displays in a yellow pop-up window.

The pop-up window automatically closes when you move your pointer to a new item in the results list. To keep the pop-up window open, drag it by the top edge to a different location.



5 Choose a function you want to use and perform any of the following:

- Add a function name after the cursor in the Command Window or Editor by double-clicking the name in the Function Browser results list.

- Copy and paste the function name into any tool or application by dragging the name from the Function Browser.
- Fight-click the function name in the Function Browser to display other options.

Interpreting Search Results in the Function Browser

Parentheses Indicate Location of Function. For results in products other than MATLAB, the product folder appears in parentheses.

When more than one function in MATLAB has the same name, the folder for the overloaded function appears in parentheses.

For example::

- `filter` is in MATLAB
- `filter (signal)` is in the Signal Processing Toolbox
- `filter (timeseries)` is in the `timeseries` folder in MATLAB

Highlights in Search Results. In the results, the search term is highlighted in blue.

When a result does not include any blue highlighting, the matching term is not part of the name, syntax, or brief description. The term is somewhere else in the reference page.

Results for a Two-Letter Search Term. For faster performance, when you type only two letters, the Function Browser looks only for exact matches.

Viewing the Full Reference Page from the Function Browser

View the complete reference page for a function in a small help window by doing either of the following:

- From the Function Browser, right-click the function. From the context menu, select **Help on Function**.
- From the Function Browser pop-up window, click **More Help**.

To open the reference page in the Help browser instead of the small window, set the preference in **File > Preferences > Help**.

Repeating a Search

As you type, the Function Browser displays a history of similar search terms that you previously entered, just below the search field. To perform a search again, select an item from the list and press **Enter**.

To view the entire search history for the current session, press the down arrow key when the search field is empty. To close the history, press **Esc** (escape).

Customizing the Function Browser

- Show or hide the Browse for functions button in the Command Window using Command Window preferences.
- Show or hide the Browse for functions button in the Editor using Toolbars preferences for the Editor.
- Cange the font used in the Function Browser by selecting **File > Preferences > Fonts**, The Function Browser uses the desktop text font. The pop-up window uses the HTML proportional text font.

See Also

- "M-Lint Code Check Report" on page 9-22
- "Getting Help for Functions and Blocks" on page 4-3
- Chapter 4, "Help, Demos, and Related Resources"

Controlling Output in the Command Window

In this section...

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**, the MATLAB software 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.

See also the `display` reference page.

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 specify keyboard shortcuts for the Cursor Up and Cursor Down actions, for example. For more information on keyboard shortcuts, see "Customizing Keyboard Shortcuts" on page 2-78.

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
```

Number of Characters in Command Window Display

The maximum line length for Command Window display is 25,000 characters. If the output from a statement exceeds this limit, the Command Window truncates the output and displays the following message at the end of the line of output:

```
Output truncated.  Text exceeds maximum line length of 25,000
characters for Command Window display.
```

Clearing the Command Window

Clear the Command Window view without clearing the workspace by selecting **Clear Command Window** from the **Edit** menu or context menu. A confirmation dialog box appears if the preference for displaying the dialog box is selected.

Afterwards, unless you have changed the keyboard shortcut for it, you can use the up arrow it to recall previous functions from the command history.

For more information, see:

- "Confirmation Dialogs Preferences" on page 2-133.
- "Customizing Keyboard Shortcuts" on page 2-78.

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 2-119.

Keeping a Session Log

The diary Function

The `diary` function creates a copy of your session in MATLAB on 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 MATLAB Editor. To create a file on your disk called `sept23.out` that contains all the functions you enter, as well as output from MATLAB, 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 contains a log of all functions executed in the current and previous sessions—see "Using the Command History Window" on page 3-61
- The `logfile` startup option—see "Startup Options" on page 1-17.

Finding Text in the Command Window

In this section...

Introduction

You can search for specified text that appears in the Command Window, where the text is either part of input you supplied, or output displayed by the MATLAB software. After finding the 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.

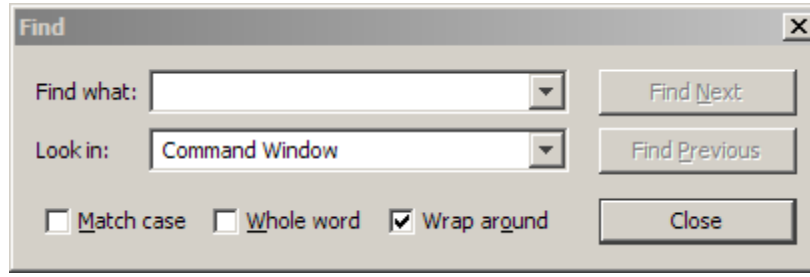
For techniques to reuse previous statements and navigate in the Command Window, see also "Recalling Previous Lines in the Command Window" on page 3-18, and "Completing Statements in the Command Window — Tab Completion" on page 3-21. To find files and text in files, see "Viewing Files and Folders" on page 6-11.

Finding Text Currently Displayed in the Command Window

To search for specified text currently displayed in the Command Window, follow these steps:

1 Select **Edit > Find** when the Command Window is active.

The Find dialog box opens.



.

2 Complete the dialog box, and then click **Find Next** or **Find Previous**.

   The search begins at the current cursor position. MATLAB finds the text you specified and highlights it.

3 Repeat step 2 to find another occurrence.

   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.

To search for the specified text in other MATLAB desktop tools, change the selection in the **Look in** field.

Increasing the Amount of Information Available for Searching in the Command Window

To increase the amount of information displayed in the Command Window so that more text is available for searching, follow these steps:

1 Select **File > Preferences > Command Window**, and then increase the setting for the "Number of lines in command window scroll buffer" on page 3-60.

2 Do *not* clear the Command Window.

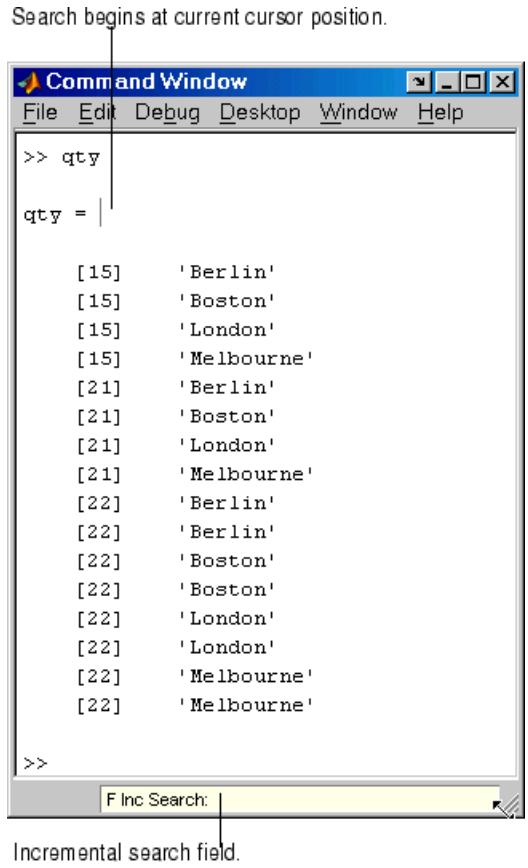
   In other words, do not enter `clc` or select **Edit > Clear Command Window**.

Performing an Incremental Search in the Command Window

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, follow these steps:

1. Position the cursor where you want the search to begin.
2. Begin the incremental search using one of the following, depending on your setting for the **Command Window key bindings**. (Available from **File > Preferences > Keyboard**.)
   - If you set the preference to **MATLAB Standard (Emacs)**, press **Ctrl+S**
   - If you set the preference to **Windows** , press **Ctrl+Shift+S**
3. To look for the previous occurrence, press **Ctrl+R** or **Ctrl+Shift+R** instead.

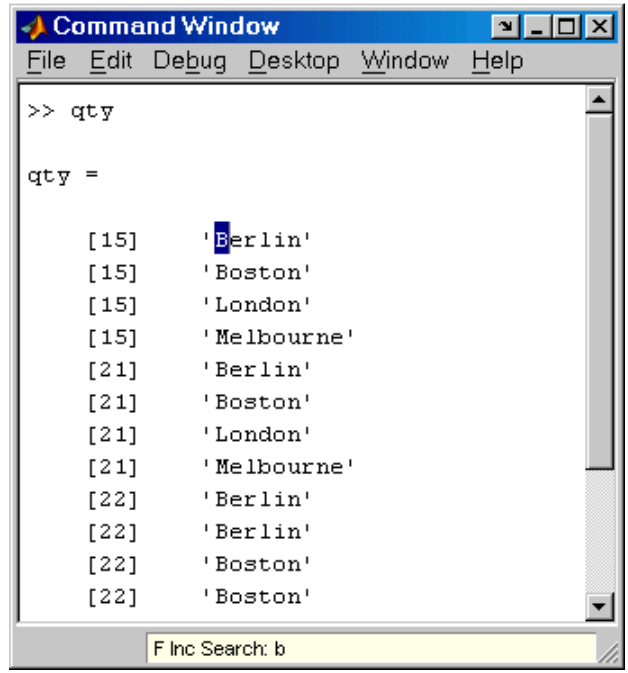
   An incremental search field, **Inc Search**, appears at the bottom of the Command Window and is preceded by **F** for a forward search, or **R** when you are looking for the previous occurrence (reverse search).



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 3-54.

MATLAB finds the next 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** (escape) or **Enter**, or any other key that is not a character or number.

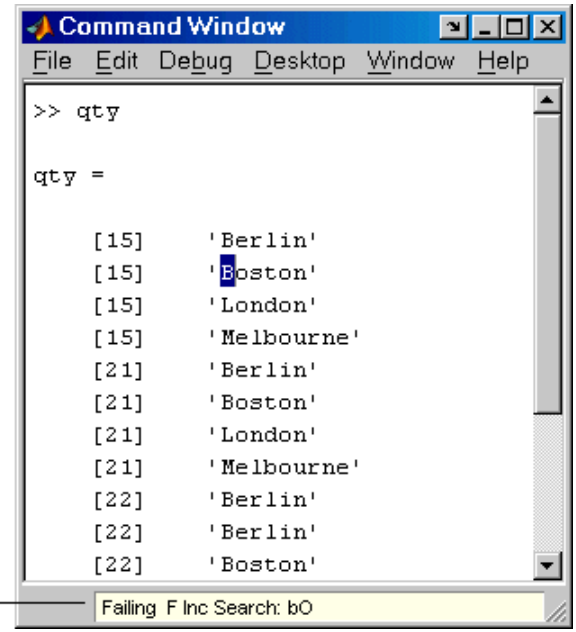
The **Inc Search** field no longer appears. The cursor is at the position where the text was last found, with the search text highlighted.

Incremental search is also available in the Editor — see "Performing an Incremental Search in the Editor" on page 8-77.

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.

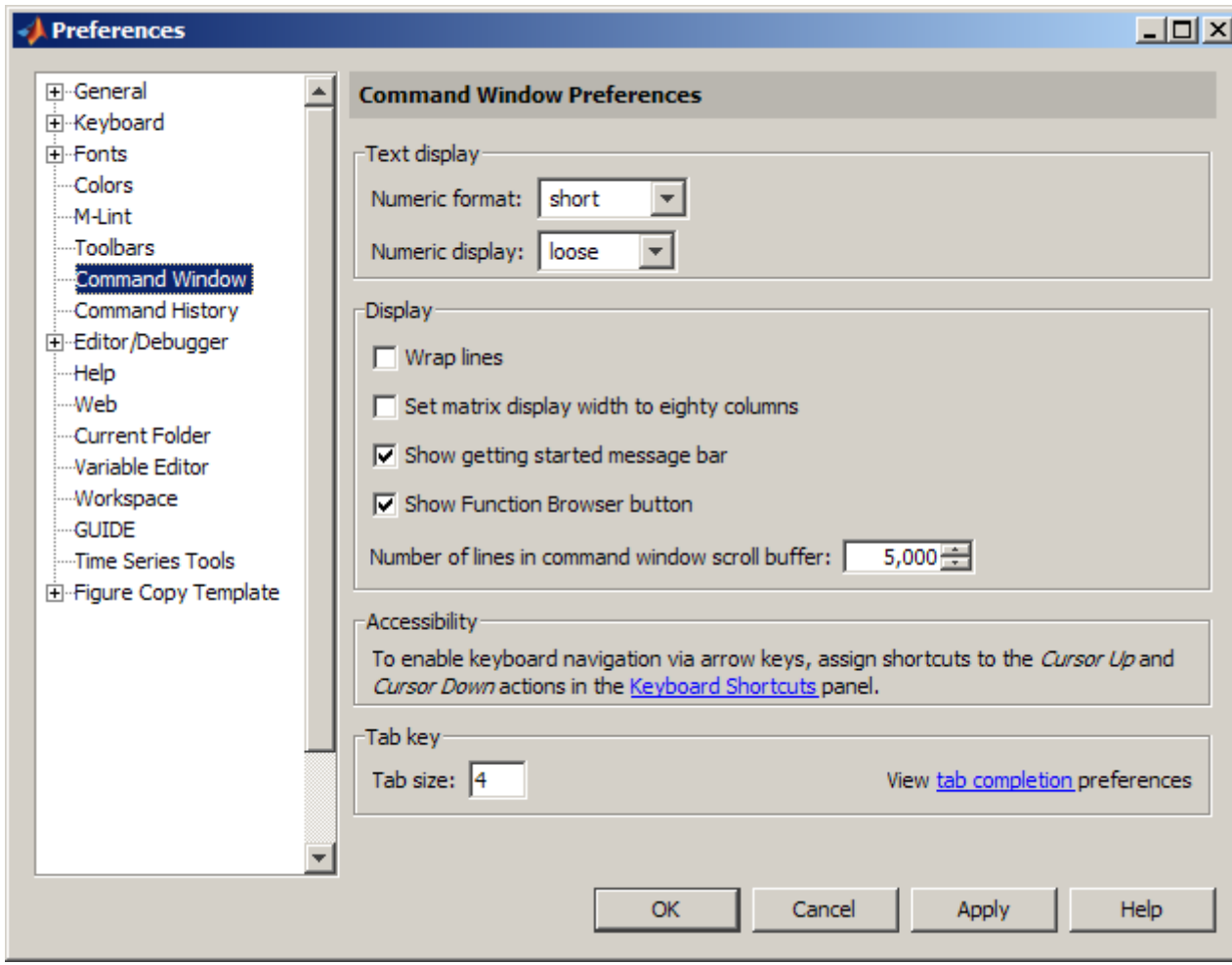


Preferences for the Command Window

In this section...

Text, Display, Accessibility, and Tab Size Preferences

To set 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 the MATLAB software 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. With this option cleared, use the horizontal scroll bar to view the entire contents of the line.

Set matrix display width to eighty columns. When selected, MATLAB displays 80 characters of matrix output in a single row, and then continues displaying output in a new row, regardless of the width of the Command Window. Use the horizontal scroll bar if the width of the Command Window is less than 80 characters.

With the check box cleared, a row of matrix output fills the width of the Command Window, and then continues displaying output in a new row. Note that if the **Wrap lines** preference is also selected, and the width of the Command Window is less than 80 characters, each row of 80 characters of matrix output wraps to fit within the width of the Command Window.

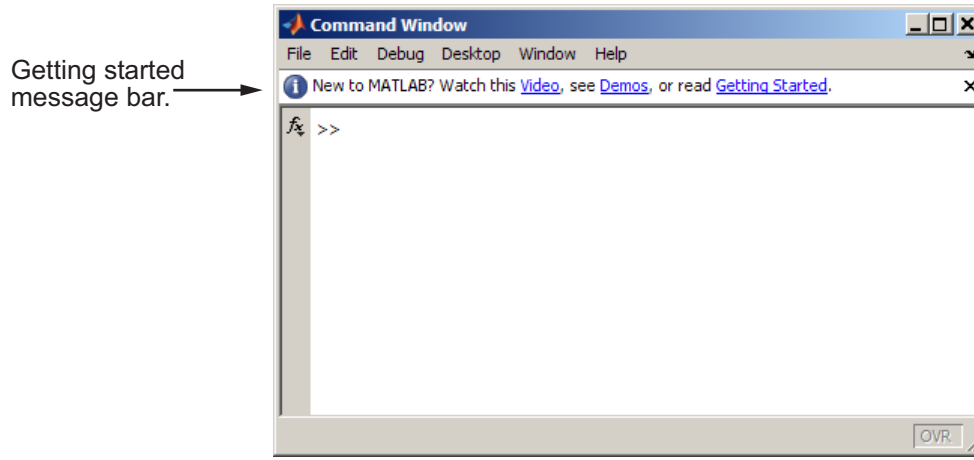
To determine the number of characters and lines that will display in the Command Window, given its current size, use

```
get(0,'CommandWindowSize')
```

When the matrix display width preference is *not* selected, the number of characters for the width is based on the current width of the Command Window. For example, a result of `50, 25` means 50 characters will display across the Command Window, and 25 lines will display. However, with the preference selected, the result for that same size Command Window is `80, 25`.

Show getting started message bar. The message bar in the Command Window 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 you close the message bar, you can still access the documentation and demos it linked to—for more information, see Chapter 4, "Help, Demos, and Related Resources".

If you closed the message bar and want to show it again, select the **Show getting started message bar** check box in the Command Window **Display** preferences.



Show Function Browser button. The Function Browser button appears to the left of the prompt in the Command Window. You use it to access the Function Browser. If you do not want the button to appear because of the space it requires, you can hide it by clearing the **Show Function Browser button** check box. When the button is not shown, you can access the Function Browser by pressing **Shift+F1** or by right-clicking in the Command Window and selecting **Function Browser** from the context menu. For more information about the Function Browser, see "Finding Functions Using the Function Browser" on page 3-37.

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 (unless you have changed the keyboard shortcut for the up arrow key). By default, you can use the up arrow key, to recall all lines shown in the Command History window, regardless of how many lines you can see in the Command Window.

Accessibility

Click the **Keyboard Shortcuts** link to assign keyboard shortcuts to the **Cursor Up** and **Cursor Down** actions in the Command Window. These actions enable you to move the cursor in the lines above the command line prompt without using the mouse. For more information, see "Customizing Keyboard Shortcuts" on page 2-78. For information about using Command Window keyboard shortcuts and JAWS software, see "Command Output Not Read" on page 2-167.

Tab key

Tab size. Number of spaces assigned to a tab stop when displaying output. The default is four spaces, except on UNIX[12] platforms where the default is eight spaces. This does not apply when the tab completion preference is selected.

See Also

- "Setting Fonts Preferences for Desktop Tools" on page 2-141
- "Confirmation Dialogs Preferences" on page 2-133
- "Setting Keyboard Preferences for Desktop Tools" on page 2-138

12. UNIX is a registered trademark of The Open Group in the United States and other countries.

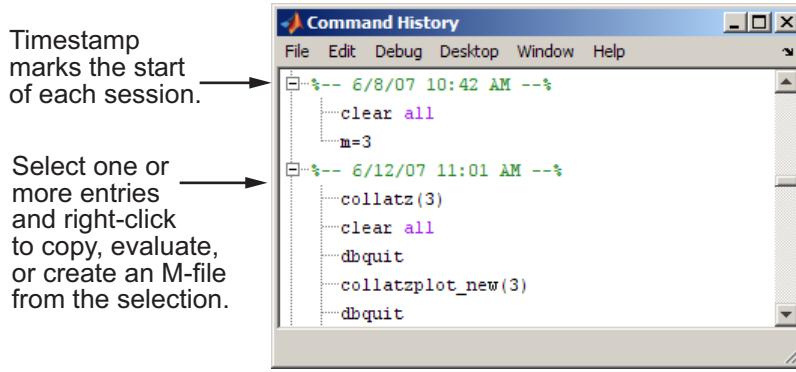
Using the Command History Window

In this section...

Overview of the Command History Window

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 Working in the Development Environment 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 "Opening and Arranging Desktop Tools" on page 2-6.



MATLAB provides other options for viewing a history of statements. See also the following sections:

- "Recalling Previous Lines in the Command Window" on page 3-18, which describes using the up arrow in the Command Window
- The `diary` function reference page
- "Startup Options" on page 1-17, which includes the `logfile` startup option

Command History File

The Command History window and the Command Window's feature for "Recalling Previous Lines in the Command Window" on page 3-18 both use the command history file, `history.m`, which is stored in the same folder as preferences for MATLAB. Type `prefdir` in the Command Window to see the location of the file. The command history 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.

Statements saved to the history are those that run in the Command Window. This includes statements you run using the **Evaluate Selection** item on context menus in tools such as the Editor, Command History, and Help browser. The history does not include every action taken in MATLAB, however, For example, if you run the statement

```
a = 1:10
```

and then modify the value of `a` in the Variable Editor, there is no record in the history that you modified the value of `a`.

MATLAB automatically saves the command history file throughout the session according to the **Saving** preference you specified. You can choose to automatically exclude certain statements from being written to the command history file with the **Settings** preference. For details, see "Preferences for Command History" on page 3-72.

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.

Performing Actions on Statements in 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. 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 or any application. Alternatively, drag the selection from the Command History window to an open M-file or another application.

Action	How to Perform the Action
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 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 "Running Frequently Used Statement Groups with MATLAB Shortcuts" on page 2-59.

Searching in the Command History Window

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

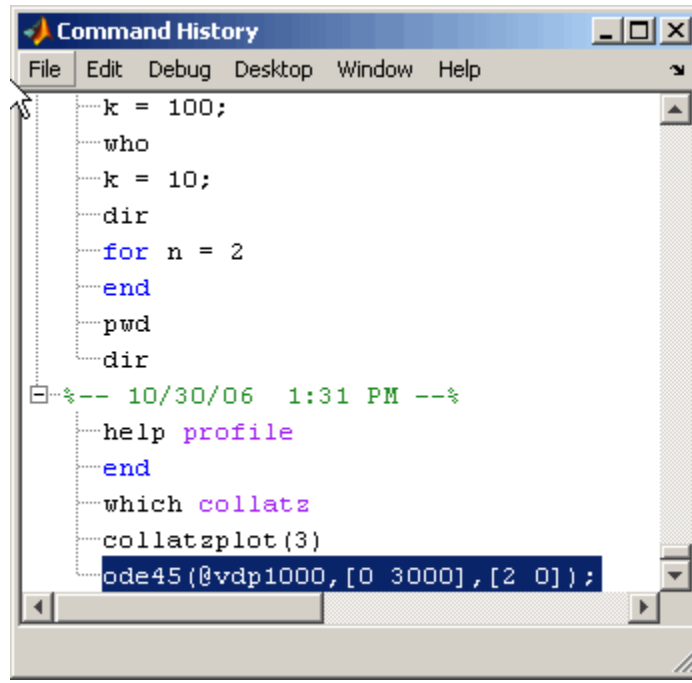
- "Finding Next Entry By Letter" on page 3-64
- "Finding Text" on page 3-69

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.

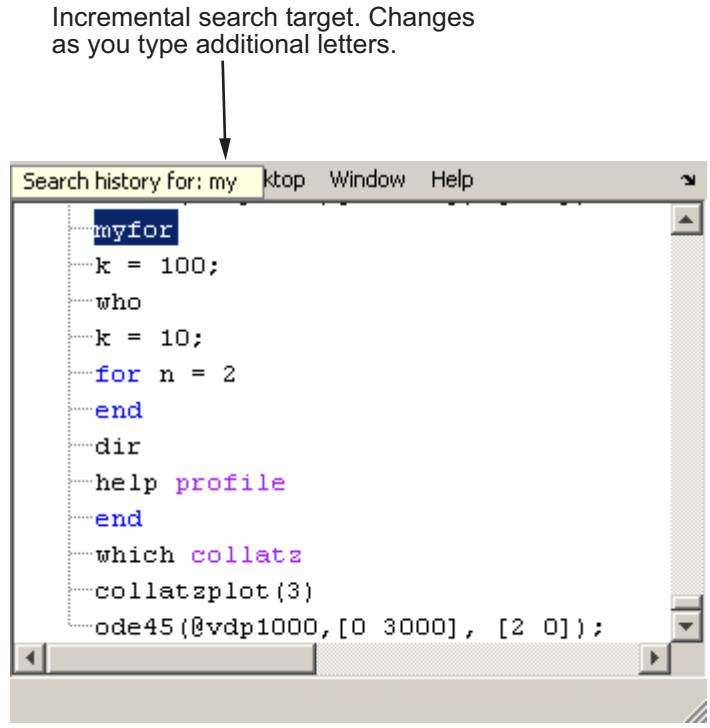


2 Type the first letters of the entry you want to find. For example, type `my`.

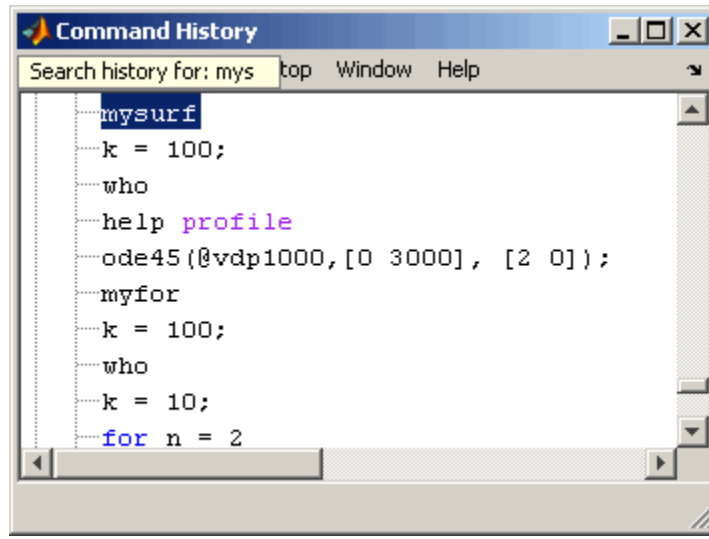
The Command History window searches backwards and selects the previous entry that begins with the letters you typed; in this example, you typed `my`, and the Command History finds `myfor`.

As you begin typing a ToolTip with the text: `Search history for:`, appears at the top of the Command History window. This ToolTip 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 session that is collapsed, it expands the session and selects the entry.

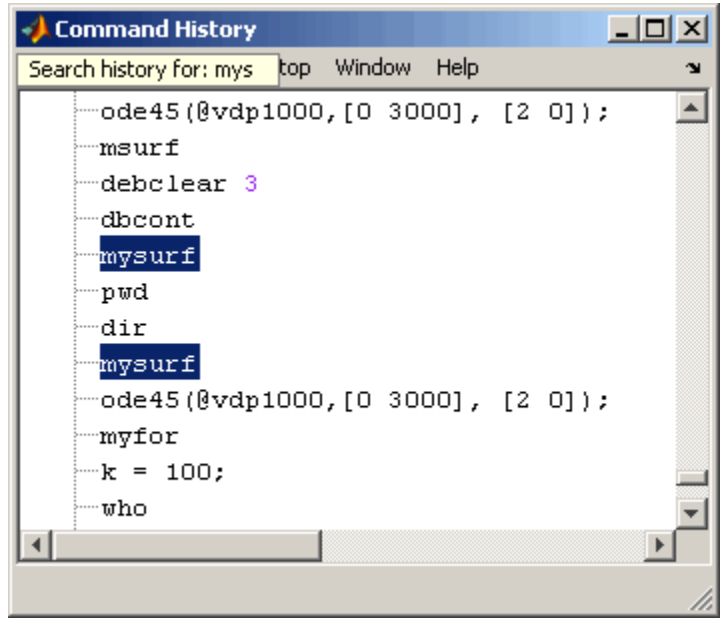


3 Now type an `s` to extend the search to `mys`. The Command History window continues to search backwards, stopping next at the function `mysurf`.

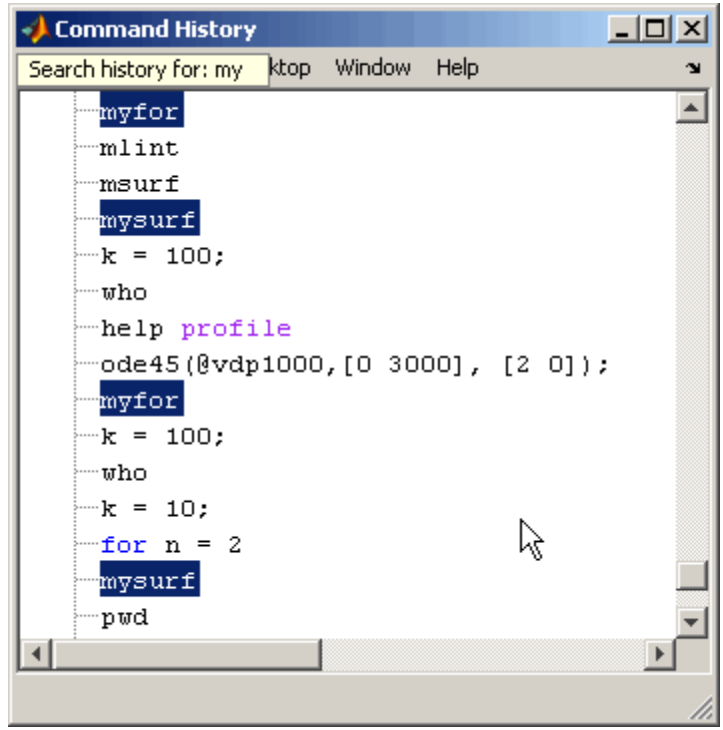


Finding Multiple Occurrences of the Entry. You can use the up and down arrow keys to search for the next or the previous occurrence of the entry you just found.

When you press **Ctrl** and the up or down arrow key, each occurrence of the entry remains highlighted while you search for additional instances.

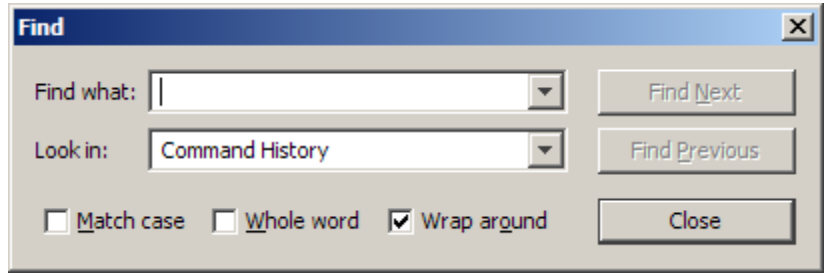


To highlight all instances of the entry, press **Ctrl+A**. In the example below, all instances of entries beginning with `my` are highlighted.



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. Find looks for visible entries only, that is, it does not find entries in collapsed nodes.



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 2-119.

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 think there are too many and it becomes inconvenient to find the ones you want. All entries remain until you delete them, or until the command history file exceeds its maximum size, at which point MATLAB automatically deletes the oldest entries. For more information, see "Viewing Statements in the Command History Window" on page 3-63.

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 their entries.

Then, right-click and select **Delete Selection** from the context menu, or press the **Delete** key. A confirmation dialog box might appear; for more information, see “Confirmation Dialogs Preferences” on page 2-133.

To delete all entries, select **Edit > Clear Command History**, or select **Clear Command 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 in the Command Window” on page 3-18.

Preferences for Command History

In this section...

Overview of Command History Preferences

Using Command History preferences, you can choose to exclude statements from the command history file, `history.m`, and specify how often to save it. The command history file is used for both the Command History window and statement recall in the Command Window.

To set preferences for the command history file, select **File > Preferences**, and then select **Command History** in the Preferences dialog box.

Settings

Specify the types of statements to exclude from the command history file. Note that when you exclude statements from the command history file, you cannot recall them in the Command Window as described in "Recalling Previous Lines in the Command Window" on page 3-18, 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 command history file.

Save Consecutive Duplicate Commands

Select the check box if you want consecutive executions of the same statement to be saved to the command history file.

For example, with this option selected, run `magic(5)`, and then run `magic(5)` again. The command history 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 command history file during a session of running the MATLAB software. By default, MATLAB saves the history after every statement. This allows you to more easily recover your state in the event of an abnormal termination, because you can reconstruct it using the history.

Save History File On Quit

Select this option to save the command history file when you end the session of MATLAB. 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 command history 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 instead of **Save History File on Quit** 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 command history 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 3-70.

See Also

- "Using the Command History Window" on page 3-61
- Additional preferences that relate to the Command History:
  - "Setting Fonts Preferences for Desktop Tools" on page 2-141
  - "Confirmation Dialogs Preferences" on page 2-133

4

Help, Demos, and Related Resources

Overview of Getting Help

For the main ways to get help, see “Getting Help” in the MATLAB Getting Started guide.

Getting Help for Functions and Blocks

In this section...

Ways to Get Help for Functions and Blocks

When You Are Using...	How to Get Help
Any desktop tool	See "Finding Functions Using the Function Browser" on page 3-37
Help browser **Contents** pane	**1** Expand the listing for a product. **2** Expand the **Functions** or **Blocks** entry to view all functions or blocks in the product. **3** Select a function or block to view its reference page. See "Browsing for Documentation and Demos".
Help browser search	**1** Search for a function or block name. **2** Sort results by type, and then view results for the Reference type. **3** Select a function or block to view its reference page. See "Searching for Documentation and Demos".

When You Are Using...	How to Get Help
Page displayed in the Help browser	1 Select a function name on the page. 2 Right-click, and select **Help on Selection**. The Help browser goes to the reference page for the selected function. To view the code for the function, in step 2, select **Open Selection**. The M-file opens in the Editor.
Current Folder browser	See: • "Viewing Descriptions" on page 6-13 • "Viewing Help for an M-File" on page 6-16
Command Window or Editor	See "Getting Help for a Function Shown in the Command Window or Editor" on page 3-35
Command Window	See: • `doc` function to display reference page • `help` function to display brief help in the Command Window

Help for Overloaded Functions

When there is more than one function with the same name (called an *overloaded* function), the help features find pages for all the functions. Choose the reference page for the specific function you want.

For example, when you get help for a selected function from the Editor, the Help browser displays the reference page for the first function on the search path. At the top of the reference page is a message bar containing links to references pages for the overloaded functions.

When M-File Help Displays in the Help Browser

When a reference page for a function does not exist and M-file help for the function exists, the Help browser displays M-file help. For example, when you

get help for a selected function from the Editor, M-file help could appear. It could be for an M-file provided by a user.

You can use the M-file help in the Help browser, or you can access it in the Command Window using the `help` function.

See Also

- "Getting Help and Demos for Files Created By Users" on page 4-26
- "Providing Your Own Help and Demos" on page 4-27
- "Documentation for Products That Are Not Installed" on page 4-55

Searching for Documentation and Demos in the Help Browser

In this section...

Simple Search

For the key steps to perform a Help browser search, see "Searching for Documentation and Demos" in the MATLAB Getting Started Guide.

Getting Better Search Results

- "Too Many Results?" on page 4-6
- "Too Few Results?" on page 4-6
- "Understanding Search Rules" on page 4-7

Too Many Results?

To reduce the number of results, try:

- "Searching in Specified Products" on page 4-9
- Adding words in the search field
- "Finding an Exact Phrase" on page 4-8
- "Excluding Pages Containing Specified Words — Boolean NOT" on page 4-9

Too Few Results?

To increase the number of results, try:

- Searching in more products. See “Searching in Specified Products” on page 4-9.
- “Searching for Part of a Word — Using Wildcards” on page 4-8
- Looking in bug reports, solutions, and technical notes at the MathWorks Web site. Click **Search Online Support** at the bottom of **Search Results** pane.

See also “Additional Resources” on page 4-54.

Understanding Search Rules

If you do not obtain the results you want, it could be because of how search works:

- Search ignores insignificant words, such as `a`, `an`, `the`, and `of`, unless they are part of an exact phrase.
- Search is not case sensitive.
- Search finds letters and digits, but not symbols. See “Searching for Special Characters and Symbols” on page 4-10.
- Search looks in the following places:
  - Documentation — the text and code you see on the page in the Help browser
  - M-file and Model demos — comments and code
  - M-GUI demos — comments in the M-file help
  - Video demos — the title

Advanced Search Techniques

- “Searching for Part of a Word — Using Wildcards” on page 4-8
- “Finding an Exact Phrase” on page 4-8
- “Searching in Specified Products” on page 4-9
- “Finding Any of the Words — Boolean OR” on page 4-9
- “Excluding Pages Containing Specified Words — Boolean NOT” on page 4-9

- "Using Multiple BOOLEAN Operators" on page 4-10
- "Searching for Special Characters and Symbols" on page 4-10

Searching for Part of a Word — Using Wildcards

To find a partial match, use the wildcard character (*) in place of characters in the search words. Search accepts more than one wildcard character.

For example:

- Find *plot*, *plots*, or *plotting*:

```
plot*
```

- Find *plot tools* or *plotting tool*:

```
plot* tool*
```

Requirements for Using Wildcards.

- Use two or more characters with a wildcard. For example, `p*` fails.
- Do not use wildcards within an exact phrase. For example, `"plot* tool"` fails.
- Do not begin a search word with a wildcard character. For example, `*tool` fails.

Finding an Exact Phrase

To reduce the number of irrelevant results, specify an exact phrase by enclosing the words in quotation marks. Search accepts more than one exact phrase.

For example:

- Find pages that contain `plot tools`, in that sequence, with no words between them:

```
"plot tools"
```

- Find pages that contain both `"plot tools"` and `"figure palette"`:

```
"plot tools" "figure palette"
```

Searching in Specified Products

By default, the Help browser search feature looks in the documentation and demos for all installed products.

To specify which products to search, use the product filter. See "Filter by Product — Specifying Products Used in the Help Browser" on page 4-18.

After performing a search, to organize results by product, click the **Product** column header.

To search for documentation in products that are not installed, use the documentation on the MathWorks Web site. See "Documentation for Products That Are Not Installed" on page 4-55.

Finding Any of the Words — Boolean OR

For more search results, look for pages that contain any of the search words by including `OR` between words:

- Include a space before and after `OR`
- Use all capital letters for `OR`.

For example, find pages that contain `plot` and pages that contain `graph`:

```
plot OR graph
```

Excluding Pages Containing Specified Words — Boolean NOT

To find pages that contain specified search words, but do not contain other specified words, include `NOT` before the words to exclude. Using `NOT` reduces irrelevant search results.

For example, find pages that contain `"plot tools"`, but do not contain `"time series"`:

```
"plot tools" NOT "time series"
```

Using Multiple BOOLEAN Operators

Search evaluates `NOT` operators first, `OR` operators second, and `AND` operators last.

By default, search looks for pages that contain all the search words, which is called a Boolean `AND`. If there is not an `OR` or `NOT` before a word, search assumes that there is an `AND` before it.

When you construct a search with multiple operators, you can include `AND` before a word to help you understand the search.

For example, find pages that contain either `plotting tool` or `plot tools` and contain `workspace`, but do not contain `time series`:

```
"plotting tool" OR "plot tools" NOT "time series" AND workspace
```

Searching for Special Characters and Symbols

To find a symbol or special character, look for the word instead of the symbol or character.

For example, look for `plus` instead of `+`.

Other ways to find information about special characters and symbols:

- Operators and Special Characters category in MATLAB Function Reference
- Search in the PDF documentation, which supports searching for special characters and symbols. For details, see “Accessing and Printing PDF Documentation” on page 4-23.

Searching Within a Page

Using Highlighted Search Words

After performing a Help browser search, when you view a page from the results, search words on the page appear with highlights.

Any of the following clears the highlighting:

- Right-click on the page and select **Refresh**.

- Click a link on the page.
- Go to another location using the navigation bar.

To restore the highlights after clearing them, go to a different page. Then select the search result again.

Using Find in Page

Find search words or any word on a page by selecting **Edit > Find** and using the **Find** dialog box. To go to the next instance on the page, use the keyboard shortcut, **F3**. Use **Shift+F3** to go to the previous instance. The find in page feature looks for partial words. For example, `plot` finds `plot` and `plotting`.

Using Demos and Code Examples

In this section...

Overview of Demos and Code Examples

For an introduction to accessing and using demos and code examples, see:

- "Browsing for Documentation and Demos"
- "Running Demos and Code in Examples"

Types of Demos

Icon	Type	Description	Example
	M-file	M-File demos: • Tell a story using source code, commentary, and output. • Are created by publishing an M-file script to HTML output using the Editor. • Can run without stopping, or cell-by-cell. A cell is a section of code that begins with two comment symbols (%%).	In the Help browser **Contents** pane, select **MATLAB > Demos > Graphics > Square Wave from Sine Waves**
	M-GUI	Standalone tool for exploring a feature.	In the Help browser **Contents** pane, select **MATLAB > Demos > Graphics > Vibrating Logo**.

Icon	Type	Description	Example
	Model	Simulink block diagram.	In the Help browser **Contents** pane, select **Simulink > Demos > Automotive Applications > Engine Timing Simulation**.
	Video	Video demos: • Are movies that highlights key features. • Plays in your system Web browser using the Adobe® Flash Player plug-in. • Could require an Internet connection.	In the Help browser **Contents** pane, select **MATLAB > Demos > Getting Started > Importing Data from Files**.

Ways to Access Demos

- Browse for demos in a product using the Help browser **Contents** pane.
- Search for a topic of interest. Sort results by type, and then select a result from the Demos type.
- The `demo` function

Ways to Run M-File Demos

To run an M-file demo from start to finish, see “Running an M-File Demo”.

To run an M-file demo section-by-section:

1. On the demo page displayed in the Help browser, click **Open *`filename`* in the Editor**.
2. In the Editor, select **Cell > Evaluate Current Cell and Advance**.

To run an M-file demo in the Command Window:

1. Click **Run in the Command Window**.

2 Scroll up in the Command Window to see the start of the instructions.

3 Follow the instructions.

Going Directly to a Page

In this section...

Bookmarking Your Favorite Pages

You can bookmark your favorite pages in the documentation and M-file demos. In MATLAB, bookmarks are called favorites. MATLAB saves them as shortcuts, with special values for the callback and category.

Use the **Favorites** menu to add, go to, and organize favorites.

When you save a favorite, do *not* change **Callback** or **Category**.

Note You cannot migrate favorites saved in one MATLAB release to a new release.

Getting the Link to a Page

To tell someone else about a specific page in the documentation, send them a link to the page.

To get the link for a page (URL) in the Help browser:

1. With the page displayed in the Help browser, select **View > Page Location**. The Help Page Location dialog box opens.
2. Copy the link from one of the fields:
   - If the link is for someone with the same release of MATLAB, use the link to the Help browser.
   - If the link is for someone with a different release of MATLAB, or who does not have the product, use the link to the Web site. To view the

page on the Web site, click **Go**. See "Product Documentation at the MathWorks Web Site" on page 4-54.

See Also

- `doc` function to go directly to a reference page or roadmap page
- `demo` function to go directly to a demo page
- `docsearch` to go directly the first search result for the search term you specify

Adjusting the Help Browser Layout

In this section...

Providing More Space for Viewing a Page

To provide more space for viewing a page in the Help browser, click  , located to the right of the search field. The left pane closes.

To reopen the left pane, click  .

Positioning the Help Browser Next to a Tool

To position the Help browser for easy reference while you work with a tool, dock the Help browser and the tool in the desktop. The Help browser **Contents** and **Search Results** pane moves above the display pane, providing more space for viewing a page.

Setting Preferences for Help

In this section...

Filter by Product — Specifying Products Used in the Help Browser

By default, the Help browser and Function Browser use the documentation and demos for all installed MathWorks products. To find relevant information more quickly, limit the products that the Help browser and Function Browser use:

1. Select **File > Preferences > Help**.
2. For **Filter by Product**, select the products to include from the list of installed products.
3. Click **OK**.

Note The `Release Notes` entry refers to the general Release Notes overview document for all products in a release. It does not apply to the product-specific release notes, which are part of the documentation for a product.

PDF Reader — Specifying Its Location

To display the PDF version of the documentation, the Help browser requires the location of the PDF reader on your system. For example, the Adobe® Acrobat® product is a PDF reader.

On Microsoft Windows and Apple Macintosh platforms, MATLAB obtains the PDF reader location from the operating system.

On UNIX[13] platforms, the default PDF reader is Acrobat® and MATLAB automatically determines its location, if it exists. To use a different PDF reader:

1. Select **File > Preferences > Help**.
2. For **PDF reader**, enter the full path to the application.
3. Click **OK**.

Note The **PDF reader** preference is only for UNIX platforms. It does not appear in the **Help Preferences** pane for other platforms.

See also "Accessing and Printing PDF Documentation" on page 4-23.

Help on Selection and More Help — Specifying Where the Help Displays

By default, MATLAB displays the reference page in a small window when you:

- Get help on a selection from the Command Window or Editor.
- Click **More Help** from the Function Browser or function hints pop-up windows.

To display the reference page in the Help browser:

1. Select **File > Preferences > Help**.
2. For **Help on Selection and More Help**, select **In Help browser**.

13. UNIX is a registered trademark of The Open Group in the United States and other countries.

This preference does not apply when you get help for a selected function from the Current Folder browser or Help browser. The reference page opens in the Help browser.

Setting the Fonts and Colors for the Help Browser

Set fonts and colors for the Help browser the same way you do for other desktop tools, with the exceptions described in the following sections.

Specifying the Font Name, Style, and Size for the Help Browser

You specify fonts for the Help browser **Contents** and **Search Results** panes separately from the fonts for the display pane.

By default, the **Contents** and **Search Results** panes use the desktop text font. You can change the font family, style (bold or italic), and size.

The Help browser display pane and all tools that display HTML are HTML Proportional Text tools in MATLAB. By default, HTML Proportional Text tools use a custom font (Sans Serif, 10 pt.). However, not all changes to the font characteristics for HTML Proportional Text tools affect the display pane in the Help browser:

- Changing the font size applies to all text and code in the display pane.
- Changing the font family applies to *text*, but not *code* in the display pane.
- Changing the font style to bold or italic does not apply to the display pane.

See also “Setting Fonts Preferences for Desktop Tools” on page 2-141.

Example — Specifying Fonts for the Help Browser. Specify Microsoft Comic Sans® MS, italic, 14 pt. font for the Help browser display pane:

1. Select **File > Preferences > Fonts > Custom**.
2. From the **Desktop tools** list, select `HTML Proportional Text`.
3. For **Font to Use**, select **Custom**, and specify the characteristics:
   - Family — `Comic Sans MS`

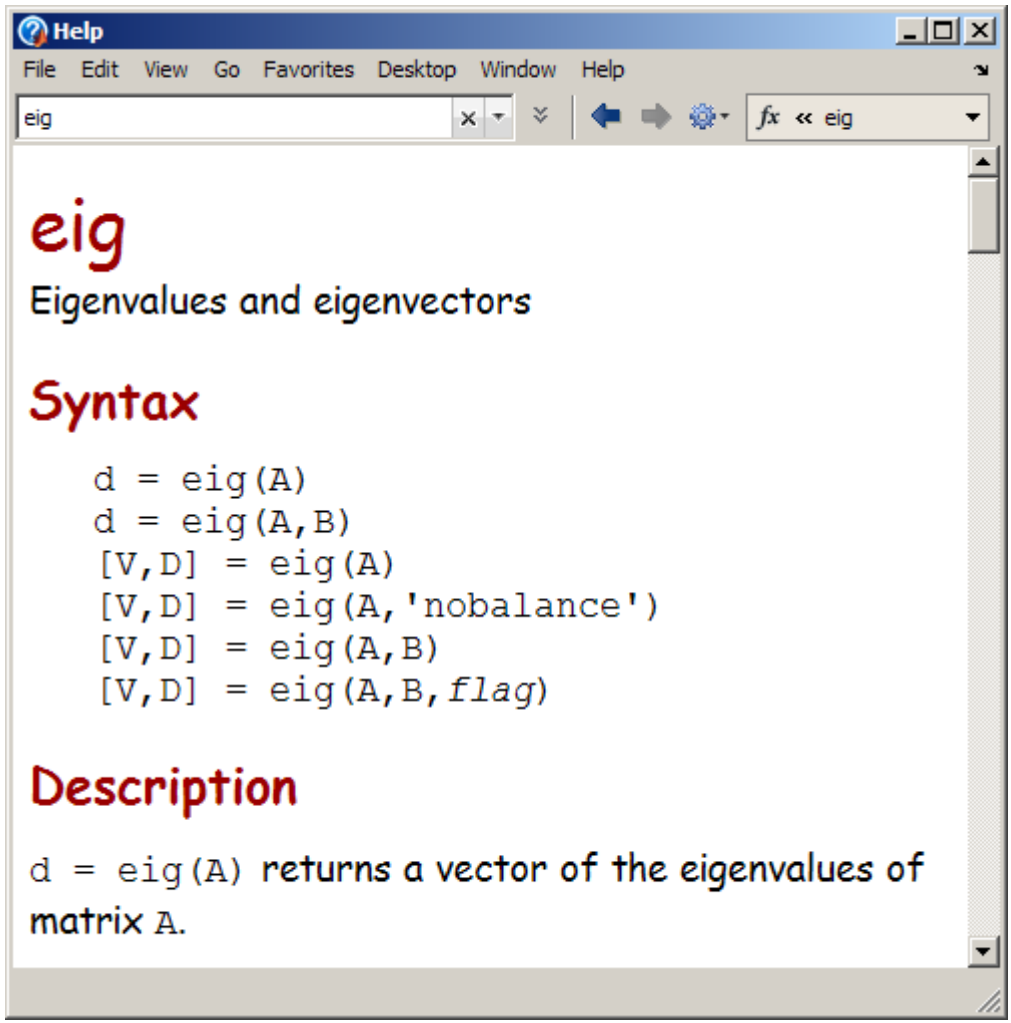
- Size in points — 14
- Style — bold

4 Click **OK**.

The font in the right pane uses the new settings. The font in other HTML Proportional Text tools, like the MATLAB Web Browser also use the new settings.

The change to the bold style has no effect.

Code in the page does not use the HTML Proportional Text font. The change to Comic Sans MS does not affect it.



Specifying Colors for the Help Browser

Specify the background and text color used in the left of the Help browser the same way you do for other desktop tools. See “Setting Colors Preferences for Desktop Tools” on page 2-150.

You cannot specify colors for the Help browser display pane.

Using Printed Documentation

In this section...

Printing from the Help Browser

1. Display the first page to print.
2. Select **File > Print**.

Accessing and Printing PDF Documentation

Some of the documentation you access from the Help browser is available in PDF format.

Use PDF documentation to:

- Print more than a few pages of documentation
- Print pages using a book style rather than a Web page style

Getting the PDF documentation from the Help browser requires:

- A PDF reader, for example, an Adobe Acrobat product.
- An Internet connection, because the PDF documentation is at The MathWorks Web site.

To access and print the PDF documentation:

1. In the Help browser **Contents** pane, select a product. The product roadmap page opens.
2. For **Printable (PDF) Documentation on the Web**, select the link for the document you want to print.

The Help browser accesses the PDF document from the MathWorks Web site and opens it in your PDF reader.

On UNIX[14] platforms, if the PDF documentation fails to open, check the **Help Preference**s. See "PDF Reader — Specifying Its Location" on page 4-18.

3. Locate the pages to print. You can use the Table of Contents and the Index, both of which provide links.

4. Print the documentation from the PDF reader.

Printed Manuals

Some MathWorks products include printed manuals, which:

- Contain a subset of the online documentation.
- Might not include the most current information.

14. UNIX is a registered trademark of The Open Group in the United States and other countries.

Getting Version and License Information

Type of Information You Want	To Get the Information
Version and license for Installed product	From the product, select **Help > About**. Or use functions: • `license` • `ver` — for MATLAB and libraries • `version` — forMathWorks products
Processor speed for the MATLAB version current running	In MATLAB, select **Help > About MATLAB**. The About MATLAB dialog box shows 32-bit or 64-bit.
`arch` value used for the `mex` function	In MATLAB, select **Help > About MATLAB**. The About MATLAB dialog box shows the value, for example `win32`. Or use the `computer` function.
Passcodes and licenses	From any desktop tool, select **Help > Web Resources > MathWorks Account**.

Getting Help and Demos for Files Created By Users

When you use files that were not provided by The MathWorks, you can get help and demos for them if the creator of the files provided help and demos to you. Use the following ways to access the help and demos for the files .

Type of Help	How to Access the Help
M-file help	Run `help` *`filename`*. The Command Window displays help when the file is in the current folder or a folder on the search path.
Summary of functions in a folder	Run `help` *`foldername`*. A summary description of all files in a folder displays. The folder must be the current folder or a folder on the search path, and contain a `Contents.m` file. To display the same information in the Help browser, run `helpwin` *`foldername`*. To see the version number for the collection of files, run `ver`.
For MATLAB class files	Run `doc` *`classname`*. Help for the class displays in the Help browser. Similarly, run `doc` *`classname.methodname`*, `doc` *`classname.propertyname`*, or `doc` *`classname.eventname`*. You can also access the help when you create an instance of a class and open the object in the Variable Editor. Click the class name link, which is below the toolbar in the Variable Editor.
Documentation and demos in the Help browser	Look in the Help browser **Contents** pane for any products that are not from the MathWorks. Searching finds words in the documentation and demos when the user included a search database.

Providing Your Own Help and Demos

In this section...

Overview of Help and Demos for Files You Create

You can provide help and demos for the files you create that are like the help and demos MATLAB provides. The information aids you and other users you share your files with.

There are different types of help you can provide.

Type of Help	Description	See
M-file	• Provide formatted comments at the start of an M-file. • View the help comments by running `help` *`filename`*. • Easy to provide. Easy to use. Suited for individual M-files.	"Creating Help for Your M-Files" on page 4-28
`Contents.m` file	• Provide a summary M-file for all files in a folder. • View the summary by running `help` *`foldername`*. • Allows you to provide a version number that users can access. • Easy to provide. Easy to use. Suited for a collection of M-files.	"Providing a Help Summary for Your M-Files" on page 4-30

Type of Help	Description	See
MATLAB class files	• Provide help in the M-file for classes you create, and optionally for the class methods, and properties and events. • View the help by running `doc classname`. • Easy to provide. Easy to use.	"Providing Help for Classes You Create" on page 4-31
Documentation in the Help browser	• Provide HTML documentation, and optionally a search database, for use in the Help browser. • View the documentation using the **Contents** pane in the Help browser, and optionally using search. • More effort than providing help in M-files. Supports graphics, images, stylized text, and page formatting. Suited for how-to and conceptual information that helps others run your files.	"Providing Your Files in the Help Browser" on page 4-37
Demos in the Help browser	• Provide demos, such as published M-files, for access from the Help browser. • View the demos using the **Contents** pane in the Help browser. • Can be generated from M-files. Support graphics, images, stylized text, and page formatting. Suited for explaining how a file works, step-by-step.	"Providing Your Files in the Help Browser" on page 4-37

Creating Help for Your M-Files

Provide help for the M-files you create that is the same as the help for M-files provided with MATLAB.

Access the help using the `help` function.

Creating M-File Help for Your Files

To create the help, provide comments on contiguous lines of the M-file:

1. Open the M-file and go to the second line of a function or the first line of a script.
2. Enter the first help comment line, called the H1 line: enter a comment symbol, `%`, followed by the function name and the purpose of the function.

   MATLAB uses this first help comment line, the H1 line, in other ways, such as in the Current Folder browser description.
3. Continue entering help comments on the following lines.
4. Enter the See also line with the names of related functions.

   - To include an overloaded function, preface the function name with the partial path, using a forward slash.
   - If there are no related functions, do not include a See also line.

   When you run help for the function, MATLAB displays the functions in the See also line as links. The functions must have M-file help and be in the current folder or in a folder on the search path.
5. End the help section, use a blank line or an executable statement.

Be consistent in how you structure your help. For example, follow the style that MATLAB uses. For examples, look at the MATLAB M-files you are familiar with.

To help you create and manage M-file help for your own files, use the "Generating a Summary View of the Help Components in M-Files" on page 9-8.

Example of M-File Help for a User-Created File

For example, for `collatz.m`, enter:

```
function sequence=collatz(n)
% COLLATZ 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
%
%   See also COLLATZPLOT, LENGTH, MYFILES/LENGTH.

sequence = n
...
```

When you run `help collatz`, MATLAB displays

```
COLLATZ 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

See also collatzplot, myfiles/length.
```

Providing a Help Summary for Your M-Files

Provide a summary file for your own collection of M-files using the same method as MATLAB. In MATLAB, for each folder containing M-files, there is a `Contents.m` file that lists the functions in the folder and a brief description.

Running `help` *`foldername`* displays the information in the `Contents.m` file. It includes links to help for all the functions. Running `helpwin` *`foldername`* displays the same information in the Help browser.

Create your own `Contents.m` files:

- Refer to a `Contents.m` file provided with MATLAB as an example.
- Use the "Displaying and Updating a Report on the Contents of a Folder" on page 9-12 to help you create and maintain your `Contents.m` files.
- Provide version information in the first two lines of the `Contents.m file`:

  ```
  % Toolbox description
  % Version xxx dd-mmm-yyyy
  ```

Do not include any spaces in the date and use a two-character day, for example, `02-Sep-2008`. Use the `ver` function to get the information.

Providing Help for Classes You Create

If you create your own MATLAB classes, provide help (comments) in the class definition M-file.

Provide help in a comment line directly following the classdef statement, or in the comment line directly following the constructor method for the class. Optionally, provide additional help comments in the classdef file for other methods, and for properties and events.

To view the help, run:

```
doc classname
```

The resulting HTML page displays in the Help browser. It provides a convenient summary of the details, properties, methods, and events for the class. It includes links to descriptions and help for any super classes, and a link to view the code.

To go directly to help for a method, run:

```
doc classname.methodname
```

Similarly, to go directly to the help for properties and events, use `doc` with the dot notation, providing the property name or event name after the dot.

When you create an instance of a class and open the object in the Variable Editor, you can access the HTML help for the class. To view the help, click the class name link below the toolbar.

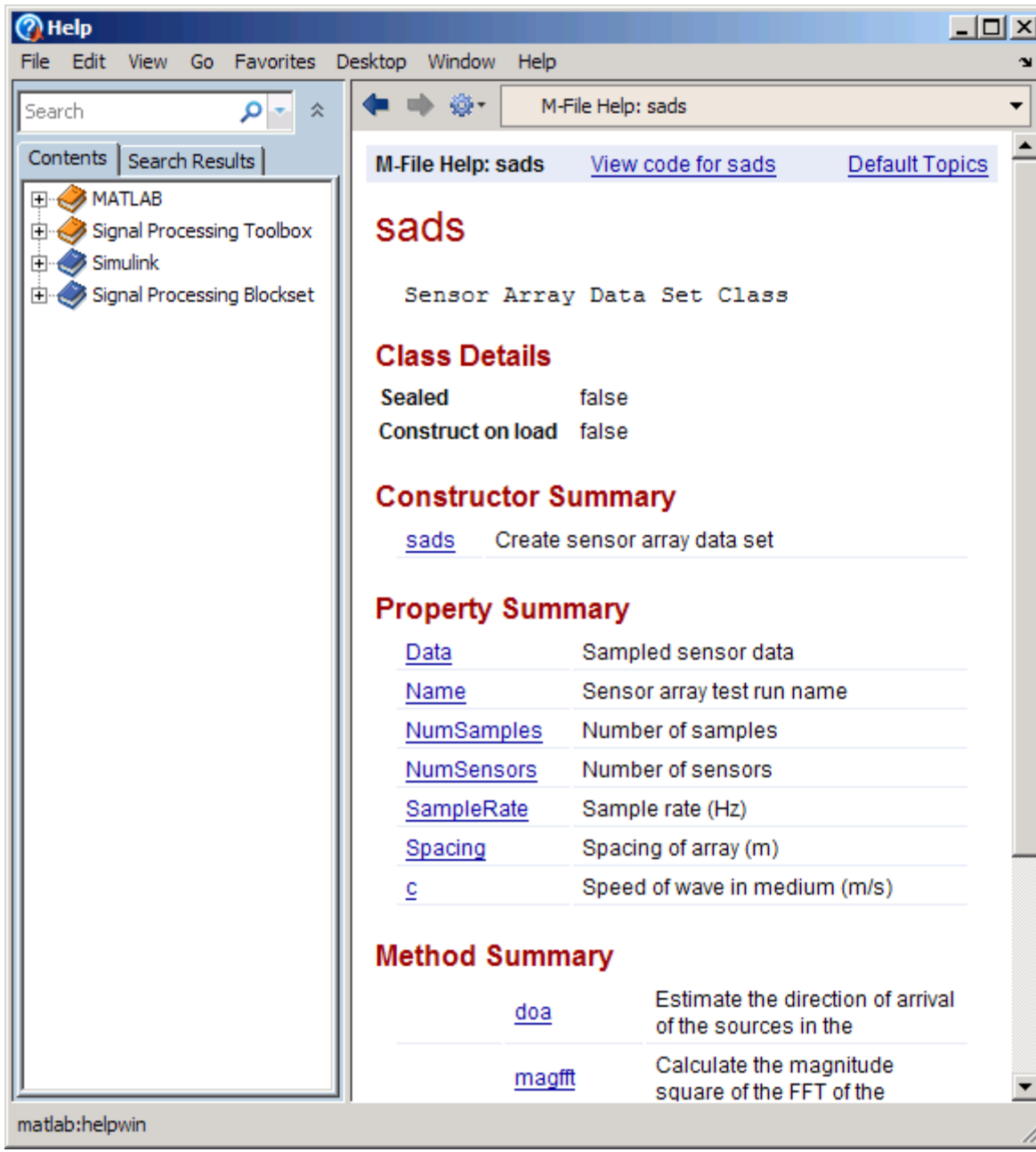
Example of Help for a User-Created Class

The example shows help for a user-created class M-file, `sads.m`, provided with MATLAB documentation. Run the following statements to use the example.

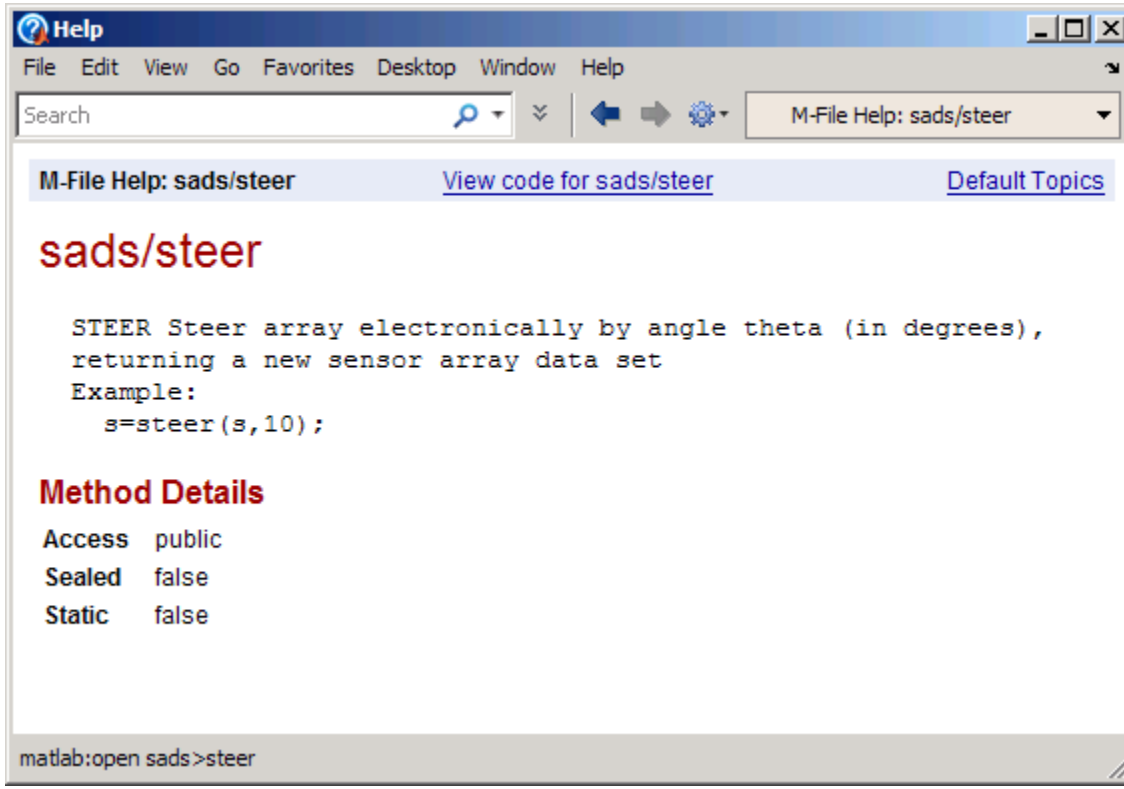
```
% Change the current directory to where the example file is located.
cd(fullfile(matlabroot,'help','techdoc','matlab_env', 'examples'))
% View help for the file in the Help browser.
doc sads
```

```
% For more information, follow links. Or go directly to help, e.g., for the steer method.
doc sads.steer
% To see the help comments in the class M-file, sads.m, click the link in help, or run:
open(fullfile(matlabroot,'help','techdoc','matlab_env', 'examples','sads.m'))
% Create an instance of the sads class.
loadparameters
sensorArray=sads(Data, Wavelength,SampleRate,Spacing,Name);
% Another way to see help for sads: click the classname link in the Variable Editor.
% Open sensorArray in the Variable Editor.
openvar sensorArray
```

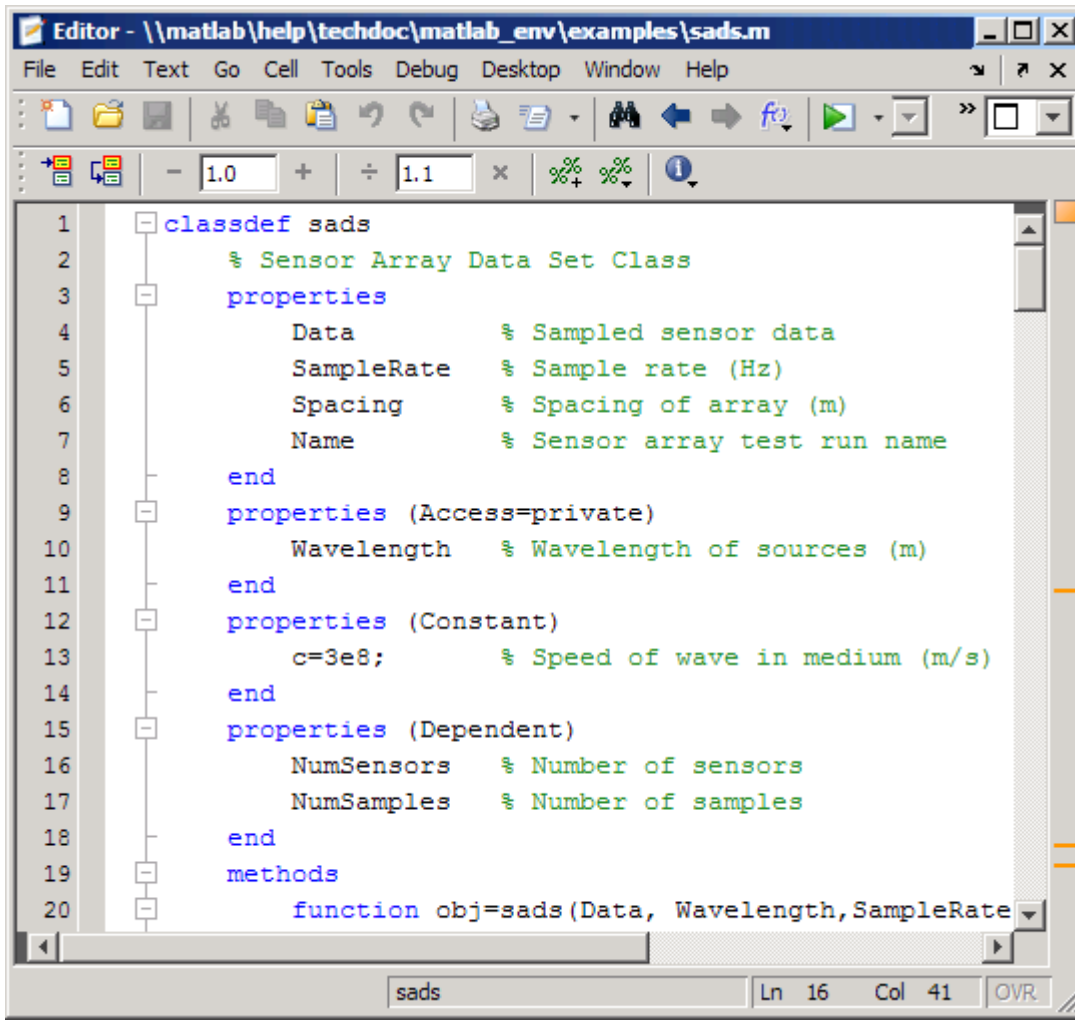
The following illustration shows the help for `sads`, displayed in the Help browser.


Help
File Edit View Go Favorites Desktop Window Help
Search
Contents
Search Results
MATLAB
Signal Processing Toolbox
Simulink
Signal Processing Blockset
M-File Help: sads
M-File Help: sads
View code for sads
Default Topics
sads
Sensor Array Data Set Class
Class Details
Sealed false
Construct on load false
Constructor Summary
sads Create sensor array data set
Property Summary
Data Sampled sensor data
Name Sensor array test run name
NumSamples Number of samples
NumSensors Number of sensors
SampleRate Sample rate (Hz)
Spacing Spacing of array (m)
c Speed of wave in medium (m/s)
Method Summary
doa Estimate the direction of arrival of the sources in the
magfft Calculate the magnitude square of the FFT of the
matlab:helpwin

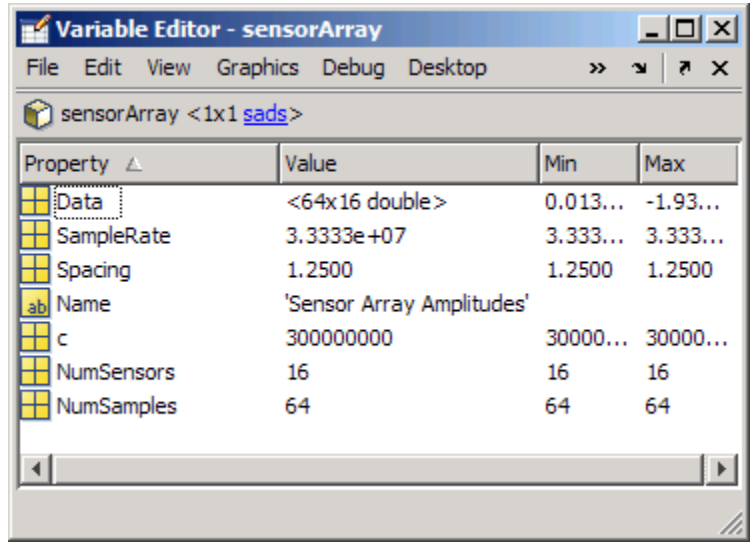
To see more information for the `steer` method, click the `steer` link under "Method Summary" in the `sads` help page, or run `doc sads.steer`.



Open the `sads` M-file by clicking the **View code for sads** link at the top of the `sads` help page.



Create an instance of the `sads` object, `sensorArray`, and view it in the Variable Editor. To view the `sads` help in the Help browser, click the **sads** link in the Variable Editor.


Variable Editor - sensorArray
File Edit View Graphics Debug Desktop
sensorArray <1x1 sads>
Property
Value
Min
Max
Data
<64x16 double>
0.013...
-1.93...
SampleRate
3.3333e+07
3.333...
3.333...
Spacing
1.2500
1.2500
1.2500
Name
'Sensor Array Amplitudes'
c
300000000
30000...
30000...
NumSensors
16
16
16
NumSamples
64
64
64

Providing Your Files in the Help Browser

In this section...

Overview of Providing Files for the Help Browser

A *toolbox* is a collection of files for use with MathWorks products that a user creates and provides to other users. For toolboxes you create, you can provide HTML help files and demos.

If you want your users to access the help files and demos in the Help browser, add them to the Help browser using special XML files.

You can also provide access to your help files and demos from the Start button

Providing HTML Help Files

Creating and Viewing HTML Help Files

Creating HTML help files for your toolbox allows you to include more than the text information that you can include with M-file help. For example, in HTML help files, you can use figures, illustrations, screen captures, equations, and formatting to make your help more usable.

To create HTML help files, use the MATLAB Editor, another text editor, or an HTML editing tool. If you use the Editor, use syntax highlighting and indenting features for HTML and related files. To customize the syntax highlighting and indenting in the Editor, select **File > Preferences > Editor/Debugger > Language**, and choose `HTML`.

To view an HTML help file that you created, use the `web` function. in MATLAB. For example, display `my_help_file.html` in the MATLAB Web browser:

```
web('I:\my_matlab_files\my_help_file.html')
```

To display the file in the Help browser, use the `web` function with the `-helpbrowser` option:

```
web('fullpath/filename.html', '-helpbrowser')
```

Include a link from M-file help for a function to an HTML help file. MATLAB M-file help includes a link to the HTML reference page for the function. To create this link in the M-file help, use a `matlabcolon` statement (`matlab:`) with the `web` statement.

Process for Adding Help Files to the Help Browser

Add HTML help files for your own toolbox to the Help browser so users can access them using the Help browser **Contents** pane. Optionally, provide a search database for your help files so users can find information in your help files using the Help browser search features.

The Help browser determines what to display using `info.xml` files on the search path. For the Help browser to display your HTML files in the Help browser:

- Create an `info.xml` file for your toolbox, which includes a pointer to your HTML help files.
- Provide the `info.xml` file to your toolbox users, along with the other help files for the toolbox.
- Instruct your toolbox users add the `info.xml` file to a folder on their search path so they can view the help files in the Help browser.

If you also want to include your toolbox on the **Start** button, put the Help browser and **Start** button information into a single `info.xml` file.

Instructions for Adding Help Files to the Help Browser

Follow these instructions to add your own HTML help files to the Help browser. Use the example files provided for the SpecSim (Example) Toolbox.

First, follow the procedure using the example files to see how it works. Then customize all the names and content for your files, or repeat the procedure for your files and delete the example files and folders.

1. Create or choose a folder for storing your help files. You must have write access to the folder. For this example, the folder name is `specsim_example`.

2. Add the folder to the search path. For the example, add `specsim_example` to the search path.

3. Create your `info.xml` file:

   a. Copy *`matlabroot`*`/help/techdoc/matlab_env/examples/helpinfo_specsim.xml` to the folder. For the example, copy the file to `specsim_example`.

   b. Rename `helpinfo_specsim.xml` to `info.xml`. In the example, rename `helpinfo_specsim.xml` in the `specsim_example` folder to `info.xml`.

4. Follow these steps to prepare the Help **Contents** pane and your HTML files:

   a. In the folder containing your `info.xml` file, create a subfolder for your help files. For the example, in `specsim_example`, create `specsim_example_helpfiles`.

   b. From *`matlabroot`*`/help/techdoc/matlab_env/examples`, copy `helptoc_specsim.xml` and `helpfuncbycat_specsim.xml` to the folder for your help files. For the example, copy the files to `specsim_example_helpfiles`.

   c. In the folder containing your files, rename `helptoc_specsim.xml` to `helptoc.xml`, and `helpfuncbycat_specsim.xml` to `helpfuncbycat.xml`. In the example, rename the files in `specsim_example_helpfiles`.

d Put your HTML help files in the same folder as your `helptoc.xml`. For the example, copy `specsimicon.gif` and the set of `specsim_...` HTML files from *`matlabroot`*`/help/techdoc/matlab_env/examples` to `specsim_example_helpfiles`.

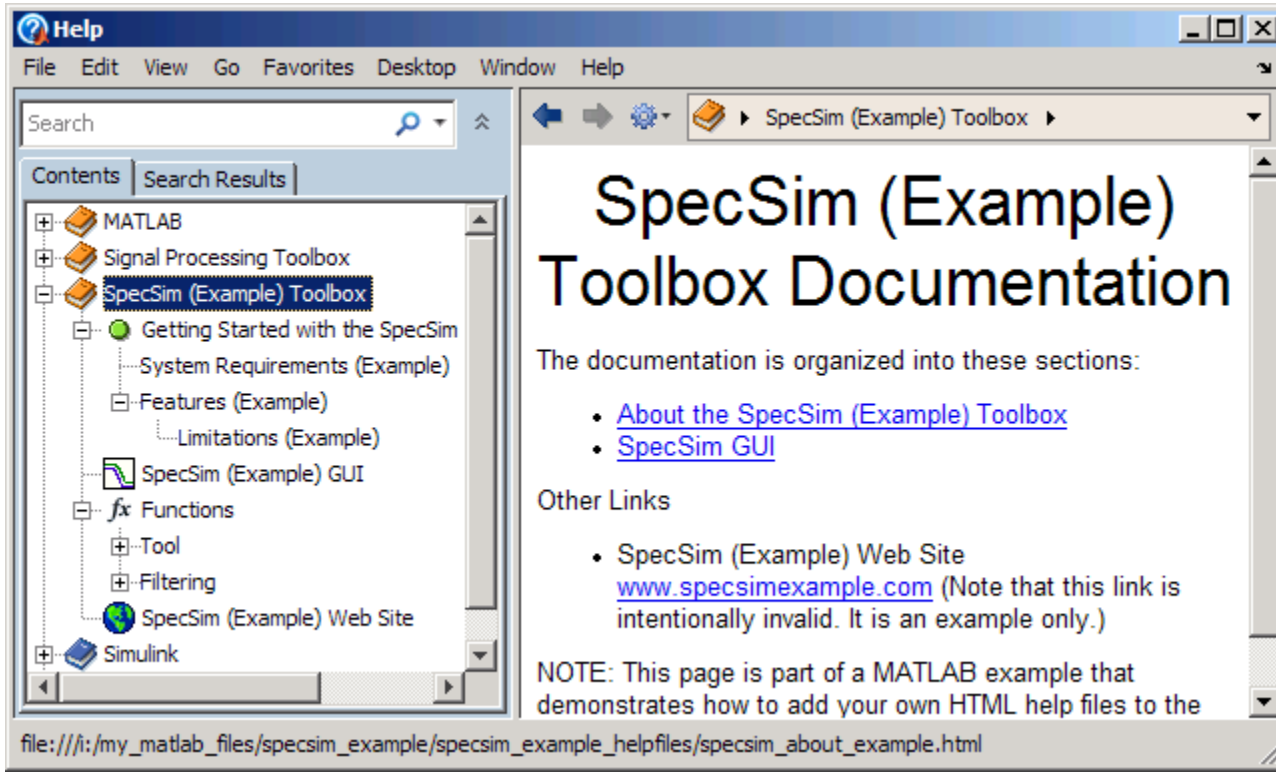
5 In the Editor, open, modify, and save your `info.xml`, `helptoc.xml`, and `helpfuncbycat.xml` files:

a In `helptoc.xml`, change the value in `<tocreference target=>` from `"helpfunctbycat_specsim.xml"` to `"helpfunctbycat.xml"`.

b Make other changes to `helptoc.xml`, `info.xml`, and `helpfuncbycat.xml` files for your toolbox. For details, see "File Descriptions for Adding Your Help Files" on page 4-41. For the example, make no other changes.

6 Select your toolbox in the Help browser product filter so that the Help browser displays it. Use **Preferences > Help** to access the product filter.

7 View your HTML help files in the Help browser **Contents** pane.



Note When navigating among the HTML help files you add, the Help browser **Contents** might not refresh to synchronize with the page displayed.

File Descriptions for Adding Your Help Files

Description of the info.xml File. The `info.xml` file adds the HTML help files to the Help browser. The table describes the example `info.xml` file provided for the SpecSim (Example) Toolbox, *`matlabroot`*`/help/techdoc/matlab_env/examples/helpinfo_specsim.xml`.

XML Tag	Description	Value in Example	Notes
`<matlabrelease>`	Release of MATLAB.	`R2009b`	Not currently used.
`<name>`	Title of toolbox.	`SpecSim (Example)`	The name of your toolbox that appears in the Help browser **Contents** pane.
`<type>`	Determines the toolbox location in the Help browser **Contents**.	`toolbox`	Allowable values: `matlab`, `toolbox`, `simulink`, `blockset`, `links_targets`, `other`. SpecSim (Example) Toolbox appears with other toolboxes. The entry has the icon for toolboxes, an orange book.
`<icon>`	Icon for your toolbox help in the **Start** button.	None	If you put your toolbox on the **Start** button and include a help entry there, specify an image to use for it. In this example, the **Start** button is not used.

XML Tag	Description	Value in Example	Notes
`<help_location>`	Location of help files	`specsim_example_helpfiles`	Location of `helptoc.xml` and HTML help files you provide for your toolbox. Specify the path to `help_location` relative to the location of the `info.xml` file. If you provide HTML help files for multiple toolboxes, each `help_location` must be unique.
`<list>` `<listitem> ...`	Entries for **Start** button	None	If you also want your toolbox to appear in the **Start** button, add at least one `listitem`. For details, see "Adding Your Own Toolboxes to the Start Button" on page 2-95.

Help Files to Include in <help_location> in the info.xml File. Put these help files into the folder you specified for `help_location` in the `info.xml` file.

File Name	Required or Optional	Description
`helptoc.xml`	Required	The file that the Help browser displays in the **Contents** pane. It shows your toolbox at the top level and defines the hierarchical entries within it. It references your HTML help files. A `helptoc.xml` file provided for the example, `helptoc_specsim.xml`, illustrates the structure. If you include icons, provide them in a folder within *`matlabroot`*`/help` and specify their path using `$help/...`, as shown in the example. You can also specify a full path.
HTML help files	Required	Help files you created for your toolbox. `helptoc.xml` references at least one of these files.
`helpfuncbycat.xml` `helpblockbycat.xml`	Optional	Referenced by `helptoc.xml`, provide a **Functions** category and **Blocks** category in the Help browser **Content** for your toolbox. A file provided for the example, `helpfuncbycat_specsim.xml`, illustrates the structure. Include this entry only if you provide the necessary files. See "Including Function and Block Entries in the Contents Pane" on page 4-44.
`helpsearch` folder containing files for search database	Optional	Allows the Help browser search to find your pages. Include this entry only if you provide the necessary files. See "Making Your Help Files Searchable" on page 4-46.

Including Function and Block Entries in the Contents Pane. To include a **Functions** and **Blocks** listing for your toolbox in the Help browser **Contents** pane:

1. Create an HTML help file for each function or block in your toolbox. Store them in the same location as the `helptoc.xml` file for the toolbox. For the example, copy the files *`matlabroot`*`/help/techdoc/matlab_env/examples/specsim_f1.html` through `specsim_f3.html` to `specsim_example_helpfiles`.

2. Create a function summary or block summary HTML file. In the file, provide helpful information, such as category names and functions in the category. You can include links to the function or block documentation like the MATLAB functions by category page does. For the example, copy the file *`matlabroot`*`/help/techdoc/matlab_env/specsim_function_list.html` to `specsim_example_helpfiles`.

3. Include a `helpfuncbycat.xml`, `helpblockbycat.xml`, or both in the same location as your `helptoc.xml` file. For the example, copy *`matlabroot`*`/help/techdoc/matlab_env/examples/helpfuncbycat_specsim.xml` to `specsim_example_helpfiles`.

4. In `helptoc.xml` for the toolbox, include a `tocreference` entry for functions, blocks, or both:

   ```
   <tocreference target="helpfuncbycat.xml"></tocreference>
   <tocreference target="helpblockbycat.xml"></tocreference>
   ```

   See this entry in the example file, `helptoc_specsim.xml`.

More Examples of info.xml Files. For additional examples, see the `info.xml` files for MathWorks products:

1. Select **Start > Desktop Tools > View Start Button Configuration Files** files.

2. From the resulting Start Button Configuration Files dialog box, select a product.

3. Click **Open** to view the `info.xml` file.

Note The `info.xml` files for MathWorks products contain some custom constructs. You might not be able to implement everything you see in `info.xml` files from The MathWorks.

Making Your Help Files Searchable

The `builddocsearchdb` function creates a searchable database that the Help browser search requires to find your help files. Build the database after creating the `info.xml` file for your toolbox, and after storing your HTML help files in the `help_location` specified in the `info.xml` file. This example uses the `info.xml` file for the SpecSim (Example) toolbox with the `help_location` specified as `I:\my_matlab_files\specsim_example_helpfiles`.

1. Add the folder containing your `info.xml` file to the search path.

   For the example, add `I:\my_matlab_files\specsim_example\specsim_example_helpfiles` to the path.

2. Create a searchable database by running `builddocsearchdb('`*`full_path to_help_location`*`')`. For the example, run:

   ```
   builddocsearchdb ('I:\my_matlab_files\specsim_example\specsim_example_helpfiles')
   ```

   Running `builddocsearchdb` creates a folder named `helpsearch` in the *`help_location`* folder. For the example, it creates `I:\my_matlab_files\specsim_example_helpfiles\helpsearch`.

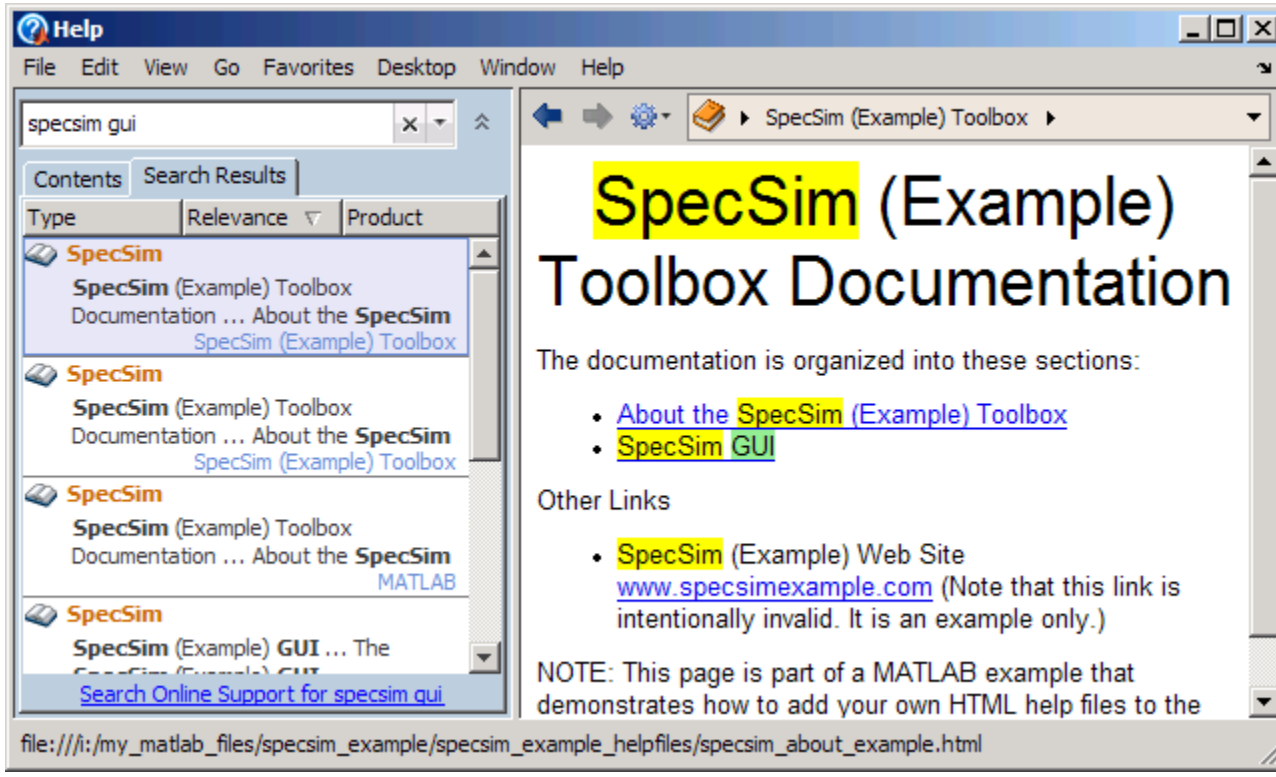
   The function also creates three files in `helpsearch`. It creates the `deletable` and `segments` file names each time you run `builddocsearchdb`. The name of the third file it creates varies, but always has the `cfs` extension.

3. Test the search features for your files.

   a. Set the product filter to only your toolbox. To access the product filter, select **File > Preferences > Help**.

   b. Use the search field in the Help browser to search for any words in the HTML help files that you provided in the *`help_location`* folder. This

example shows a search of SpecSim (Example) HTML help files for the term `specsim gui`.



Providing Demos

Creating Demos

You can provide demos and related files of any file type for your toolbox.

One way to produce demos is with the cell-publishing feature, available in the Editor. It creates a demo file to run in MATLAB and an HTML file that describes how the demo works. For more information, see "Overview of Publishing M-Files" on page 10-2.

Process for Adding Demos to the Help Browser

Add demos for your own toolbox to the Help browser so users can access them using the Help browser **Contents** pane. The Help browser determines the demos to display using `demos.xml` files on the search path. For the Help browser to display your demos in the Help browser:

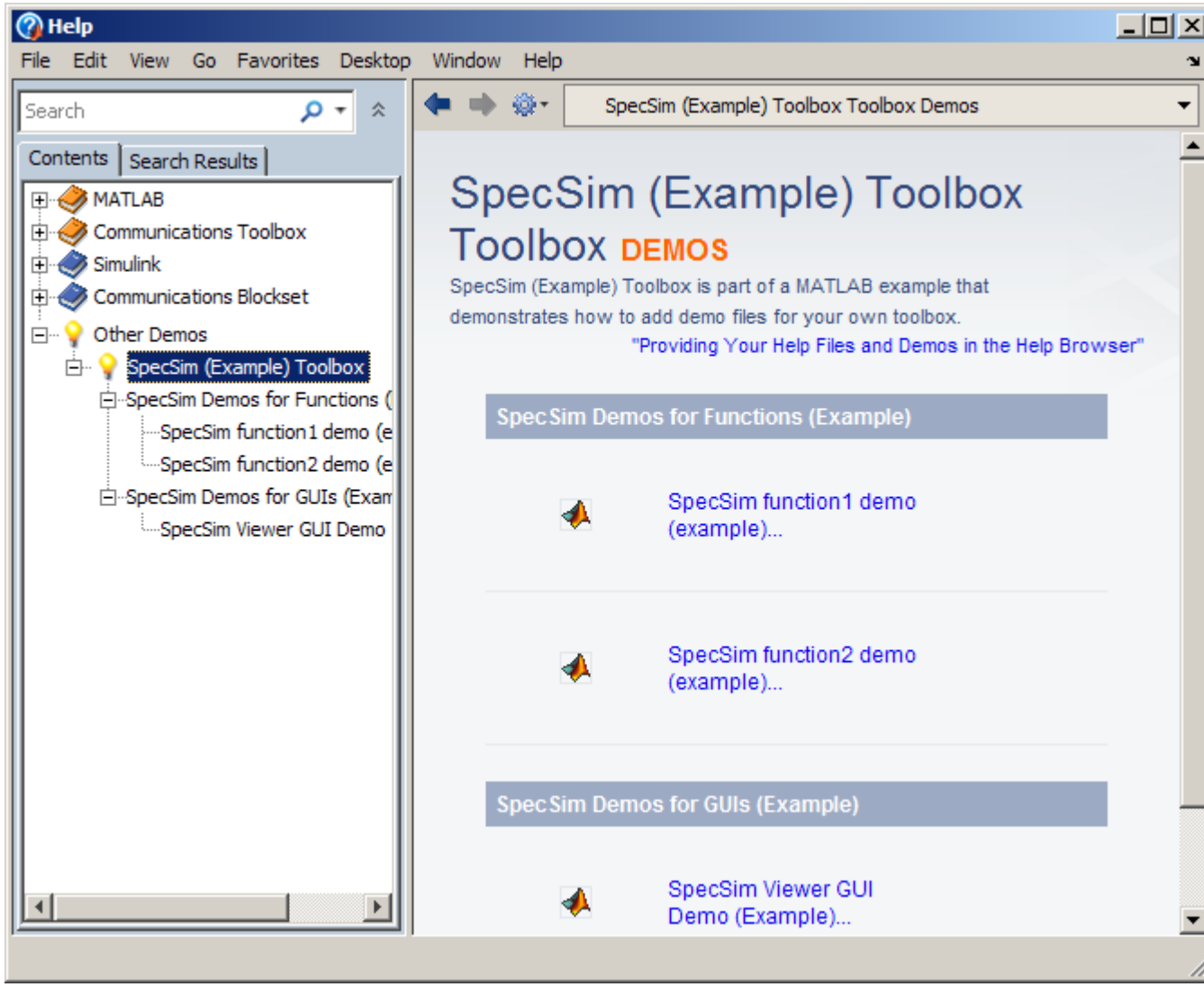
- Create a `demos.xml` file for your toolbox, which includes pointers to your demo files.
- Provide the `demos.xml` file to your toolbox users, along with the demos files for your toolbox.
- Instruct your toolbox users to add the `demos.xml` file to a folder on their MATLAB search path. Then, they can access your demos from the Help browser **Contents** pane.

Instructions for Adding Demos to the Help Browser

Follow these instructions to add your own demos files to the Help browser. Example files are provided for the SpecSim (Example) Toolbox.

1. Create or choose a folder for storing your demos files. You must have write access to the folder. For this example, the folder name is `I:/my_matlab_files/specsim_demo`
2. Add the folder for your demos files to the search path. The folder cannot be the current folder when you add it to the path.
3. Get an example `demos.xml` file to use as the basis for your own file. Copy *`matlabroot`*`/help/techdoc/matlab_env/examples/demos_specsim.xml` to the folder for your demos files. For the example, copy it to `I:/my_matlab_files/specsim_demo`.
4. Rename the copy of the file to `demos.xml`. For the example, rename the `demos_specsim.xml` file in `I:/my_matlab_files/specsim_demo` to `demos.xml`.

5 View the example `demos.xml` file in the Help browser. A new node, `Other Demos`, appears at the end of the Help browser **Contents** pane. Expand it to view the entry just added, `SpecSim (Example) Toolbox`.



6 In the Editor, open the `demos.xml` file. For the example, open `I:/my_matlab_files/specsim_demo/demos.xml`.

7 Change the `demos.xml` file to include the information for your toolbox, and save the changes.

   For details, see "File Description for demos.xml" on page 4-50.

8 Refresh the Help browser to see your demos in `Other Demos`. Remove the folder containing the `demos.xml` file from the search path, and then add it to the search path. For the example, remove `I:/my_matlab_files/specsim_demo` from the search path, and then add `I:/my_matlab_files/specsim_demo` to the search path.

Note The product filter does not provide an entry for `Other Demos` and the toolbox demos you add. The Help browser always shows them.

File Description for demos.xml. Within the `demos.xml` file, the root tag is `<demos>` and it includes one `<name>`, `<type>`, `<icon>` and `<description>` for your toolbox demos.

Include a `<demoitem>` for each demo you provide. Or, to provide categories of demos, include a `<demosection>` for each category, which contains `<demoitem>` entries. If you use any categories, then all demos must be in categories. In other words, if there is even one `<demosection>`, then all `demoitem` tags must be within `demosection` tags.

Line	XML Tag	Notes	Value for Example
2	`<demos>`	The root element for a `demos.xml` file.	No value
4	`<name>`	Name of your toolbox. It appears under `Other Demos` in the Help browser.	`SpecSim (Example) Toolbox`
5	`<type>`	The product type. Allowable values are `matlab`, `simulink`, `toolbox`, `blockset`, `links_targets`, or `other`. Not currently used.	`toolbox`

Line	XML Tag	Notes	Value for Example
6	`<icon>`	Use an icon for your demo. You can specify a relative path, that is, relative to the location of the `info.xml` file. This example uses `$toolbox/...` to reference an icon in a *`matlabroot`*`/toolbox` folder.	`$toolbox/matlab/icons/matlabicon.gif`
7	`<description>`	The description that appears in the Help browser display pane, on the main page for your demo.	`SpecSim (Example) Toolbox is ...`
11 to 12	`<website>` `<a href= "url"</a>` `</website>`	Links to a Web site. For example, MathWorks demos include a link near the top, on the right: `Product page at mathworks.com`. The `<website>` tag can appear anywhere before the `</demos>` tag. For the example, displays The MathWorks Web site.	`href = "http://www.mathworks.com/" For more information...`
14	`<demosection>`	Begins a category of demos. Each category includes a `<label>`, description, and at least one `<demoitem>`. Use for each demo category.	No value required
15	`<label>`	Title shown in Help browser for a `<demosection>`.	`SpecSim Demos for Functions (Example)`
16	`<demoitem>`	Contains `<label>` and at least one of these tags: `<callback>` or `<file>`. Use one `<demoitem>` per demo.	No value required
17	`<label>`	Title shown for `demoitem`.	`SpecSim function1 demo (example)...`
18	`<callback>`	The name of a file. When you double-click the demo icon or click the title text for the demo item in the Help browser, the file runs.	`demoone` (File not provided. The value is only an example name.)

Line	XML Tag	Notes	Value for Example
22	`<file>`	An HTML file. Specify a relative path from the location of `demos.xml`. When you click the demo name in the Help browser **Contents** pane, the HTML file displays in the Help browser .	`demotwo.html` (File not provided. The value is only an example name.)
None	`<dependency>`	Optional. Specifies other products required to run the demo, such as another toolbox. The text must match a product name specified in an `info.xml` file that is on the search path or in the current folder.	Not shown

Validating info.xml Files You Provide

When MATLAB finds an `info.xml` file on the search path or in the current folder, it tries to add information to the Help browser or **Start** button. MATLAB automatically validates the file against the supported schema. If there is an invalid construct in the `info.xml` file, MATLAB displays an error in the Command Window. The error is typically of the form:

```
XML-file failed validation against schema located in
...
XML-file name: full path to...\info.xml
```

An `info.xml` validation error can occur when you start MATLAB, press the **Start** button, or under certain other circumstances.

See these causes of the XML file validation error and what to do:

- “Outdated info.xml File for MATLAB Product” on page 4-53
- “info.xml File Unrelated to MATLAB Product” on page 4-53
- “info.xml File Intended for Use with MATLAB Software Contains Invalid Constructs” on page 4-53

Outdated info.xml File for MATLAB Product

If you have an `info.xml` file from a different version of MATLAB, it could contain constructs that are not valid with your version. To identify an `info.xml` file from another version, look at the full path names reported in the error message. The path usually includes a version number, for example, `...\MATLAB\R14\...` . The error is not actually causing any problems and you can safely ignore it. To ensure that the error does not occur, remove the offending `info.xml` file, or ensure that it is not on the search path or in the current folder.

info.xml File Unrelated to MATLAB Product

If the `info.xml` is an unrelated file, and not intended to add information to the **Start** button or Help browser, the validation error is irrelevant. In this case, the error is not actually causing any problems and you can safely ignore it. To ensure that the error does not occur, rename the offending `info.xml` file, or ensure that it is not on the search path or in the current folder.

info.xml File Intended for Use with MATLAB Software Contains Invalid Constructs

If the purpose of the `info.xml` file is to add information to the **Start** button or Help browser, correct the reported problem. Use the message in the error to isolate the problem or use any validator. One validator you can use is from the W3C® at `http://www.w3.org/2001/03/webdata/xsv`. For more information about the structure of the `info.xml` file, consult its schema, located at *`matlabroot`*`/sys/namespace/info/v1/info.xsd`.

Additional Resources

In this section...

Product Documentation at the MathWorks Web Site

The MathWorks Web site provides documentation for the latest version of all MathWorks products. To view the Web site documentation, you need an Internet connection and a Web browser. You can view the Web site documentation when MATLAB is *not* installed.

Note If you are not running the latest version of MATLAB, the documentation on the Web site could:

- Describe capabilities not available in your version.
- Describe capabilities that work differently in your version.

Ways to Access the Documentation at the MathWorks Web Site

- From MATLAB, select **Help > Web Resources > Support**.
- From a page you are viewing in the Help browser, you can go to the same page in the documentation on the Web site. See "Getting the Link to a Page" on page 4-15.
- From a Web browser, go to `http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml`.

Documentation for Products That Are Not Installed

To access documentation for products that are not installed, see "Product Documentation at the MathWorks Web Site" on page 4-54.

Documentation for the Latest Release

If you are running an older version of MATLAB, you can access documentation for the latest release. Consult the latest documentation to see if it has more information than in your version. See "Product Documentation at the MathWorks Web Site" on page 4-54.

Documentation in Other Languages

Some documentation is available in other languages. Contact your MathWorks sales and service office.

Technical Support

Technical support provides bug reports and workarounds, solutions to questions, published books and more.

Ways to access technical support:

- From MATLAB, select **Help > Web Resources > Support**.
- From a Web browser, go to `http://www.mathworks.com/support`.
- Search in the support solutions, technical notes, and bug reports from MATLAB:

  1. Perform a search in the Help browser.

2. At the bottom of the **Search Results** pane, click **Search Online Support for**.

The results display in your system Web browser.

Newsgroup

Access a user forum, the Usenet newsgroup `comp.soft-sys.matlab` (also known as `cssm`), to find, ask for, or provide help.

Select **Help > Web Resources > MATLAB Newsgroup Access**, or go to `http://www.mathworks.com/matlabcentral`.

Files Created By Other Users

Before you create your own files, see if someone has already provided a similar file. Use files created by other MATLAB users to save you time and provide you with new ideas for your own work. See Chapter 7, "File Exchange — Finding and Getting Files Created by Other Users".

Blogs

Read weekly commentary from the people who design and build MathWorks products.

Newsletters

Read News and Notes and the MATLAB Digest online. Select **Help > Web Resources > MATLAB Newsletters**, or go to `http://www.mathworks.com/company/newsletters/`.

Seminars and Webinars

See `http://www.mathworks.com/company/events/`.

Training

For details, select **Help > Web Resources > Training**, or go to `http://www.mathworks.com/services/training`.

5

Workspace Browser and Variable Editor

MATLAB Workspace

In this section...

About the Workspace

The workspace consists of the set of variables built up during a session of using the MATLAB software and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces. For example, if you run these statements,

```
A = magic(4)
R = randn(3,4,5)
```

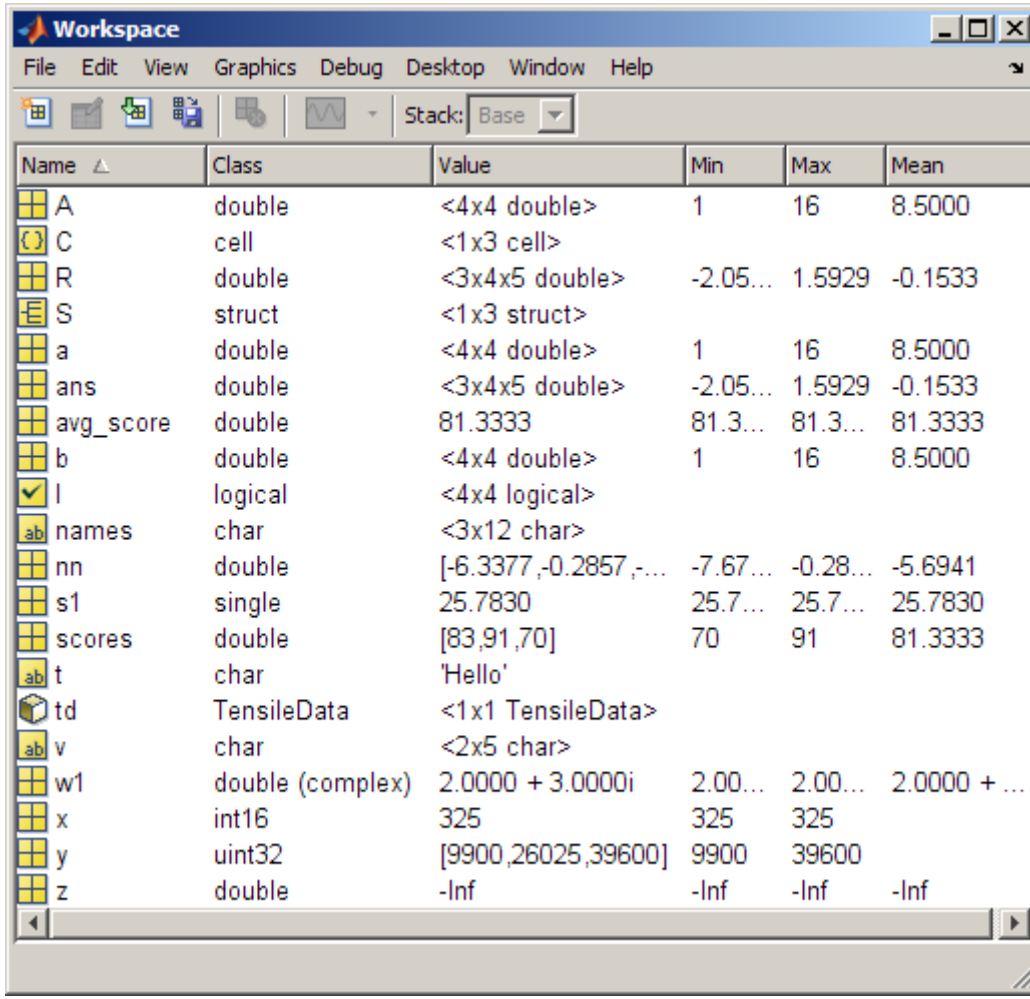
the workspace includes two variables, `A` and `R`.

You can perform workspace and related operations using the Workspace browser. When available, equivalent functions are documented with each feature of the Workspace browser.

Opening the Workspace Browser

To open the Workspace browser, select **Desktop > Workspace** in the MATLAB desktop, or type `workspace` at the Command Window prompt.

The Workspace browser opens.



You can specify which buttons you want to appear on the toolbar using preferences. Select **File > Preferences > Toolbars** to open the dialog box. Click **Help** for information using the dialog box.

Viewing and Editing Values in the Current Workspace

The Workspace browser shows the name of each variable or object, the class (also represented by the icon), its value, and where relevant, the **Min**, **Max**, and **Mean** calculations. MATLAB performs these calculations using the `min`, `max`, and `mean` functions, and updates the results automatically. These are other features of the Workspace browser:

- You can display additional columns, including size (dimensions), size in bytes, and other common statistical calculations such as mode and standard deviation. To show or hide columns, select **View > Choose Columns**. On Microsoft Windows systems, you can right-click any column header to hide it or to show or hide other columns. To specify the size limit for calculations and how NaNs are considered, use "Setting Workspace Browser Preferences" on page 5-21.
- To resize a column of information, drag the column header border. 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 header to sort on that column. Click the column header again to reverse the sort order in that column. For example, to sort on **Name**, click the column header once. To change from ascending to descending, click the header again. You cannot sort by the **Value** column in the Workspace browser.
- You can directly edit variable values in the Workspace browser **Value** column. To edit a value, position the pointer in the **Value** column at the row you want to edit, click, and type the new value.
- To view more of the data for a variable, as well as to more easily edit it, double-click a variable name and it opens in the Variable Editor. For more information, see "Viewing and Editing Workspace Variables with the Variable Editor" on page 5-24.

Function Alternative

Use `who` to list the current workspace variables. Use `whos` to list the variables and information about size and class. For example:

```
>> who

Your variables are:
```

```
A         S         avg_score  names      scores     v          y
C         a         b          nn         t          w1         z
R         ans       l          s1         td         x

>> whos
  Name           Size            Bytes  Class          Attributes

  A              4x4               128  double
  C              1x3               348  cell
  R              3x4x5             480  double
  S              1x3               826  struct
  a              4x4               128  double
  ans            3x4x5             480  double
  avg_score      1x1                 8  double
  b              4x4               128  double
  l              4x4                16  logical
  names          3x12               72  char
  nn             3x3                72  double
  s1             1x1                 4  single
  scores         1x3                24  double
  t              1x5                10  char
  td             1x1               152  TensileData
  v              2x5                20  char
  w1             1x1                16  double         complex
  x              1x1                 2  int16
  y              1x3                12  uint32
  z              1x1                 8  double
```

Use `exist` to see if the specified variable is in the workspace.

Saving the Current Workspace

The workspace is not maintained across sessions of MATLAB. 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 binary file specifically for use in MATLAB. 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 application. You can change the default file association in the Microsoft Windows operating system to associate MAT-files with either MATLAB or the Access application.

Saving All Variables

To save all of the workspace variables using the Workspace browser:

1. Select **File > Save Workspace As** from the Workspace browser, or click the Save button  in the Workspace browser toolbar.

   The Save to MAT-File 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.

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.

Another way to save selected variables from the Workspace browser to a MAT-file is by dragging the selected variables from the Workspace browser to the Current Folder browser. For more information, see “Creating and Updating MAT-Files” on page 6-31.

Specifying the Format When Saving MAT-Files

To specify preferences for saving MAT-files that pertain to compression, and compatibility between different versions of MATLAB, see “MAT-Files Preferences” on page 2-131.

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 folder. 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`. MATLAB provides additional functions for saving information — see “Exporting Data”.

Viewing and Loading a Saved Workspace and Importing Data

You can view saved variables and load the variables and other data into the workspace using the Current Folder browser, the Import Wizard, or the `load` function.

Viewing Variables in MAT-Files and Loading Them into the Workspace

Use the Current Folder browser to view the contents of a MAT-file without loading the file into MATLAB. You can also load selected variables by dragging them from a MAT-file in the Current Folder browser to the

Workspace browser. For details, see "Viewing Details About Files and Folders In the Current Folder Browser" on page 6-13.

Function Alternative. Use `whos` with the `-file` option.

Importing Data

MATLAB provides an Import Wizard to load MAT-files and other forms of data into the workspace. This procedure briefly describes how to use the Import Wizard from the Workspace browser to import data from MAT-files.

1. Click the Import Data button  on the toolbar in the Workspace browser.

   The Import Data 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. Any variables in the MAT-file that are not in the workspace are added to the workspace. If any variables being loaded have the same names as variables in the current workspace, MATLAB asks if you want to replace the values in the current workspace with the values in the MAT-file, or cancel.

You can also use the Workspace browser to import data you previously copied to the clipboard by selecting **Edit > Paste to workspace**, or use **Ctrl+V**. This imports the clipboard data using the Import Wizard.

For more information about the Import Wizard, see "Using the Import Wizard".

Function Alternative for Loading Variables

Use `load` to open a saved workspace. For example,

```
load('june10')
```

loads all workspace variables from the file `june10.mat`.

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**.

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:

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 2-133.

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
```

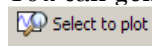
clears the variables `A` and `M` from the workspace.

Use the `clearvars` function with the `-except` option to keep the specified variables and clear all other variables.

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 base workspace in MATLAB. To access the base and function workspaces when debugging M-files, use the **Stack** field in the Workspace browser. The **Stack** field is only available in debug mode and otherwise appears dimmed. The **Stack** field is also accessible from the Editor and the Variable Editor. For more information, see "Finding Errors, Debugging, and Correcting M-Files" on page 8-116. See also the `dbstack` and `evalin` functions.

Creating Plots from the Workspace Browser

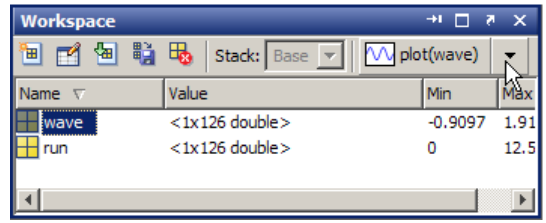
You can generate a plot of one or more variables with the *Plot Selector* tool 

on the Workspace browser toolbar. (The appearance of the tool varies, depending on the variables you select and your history of using it.) The Plot Selector contains menu items identifying plotting functions available to you and executes them when you click the graph icons. The following steps illustrate how the tool works:

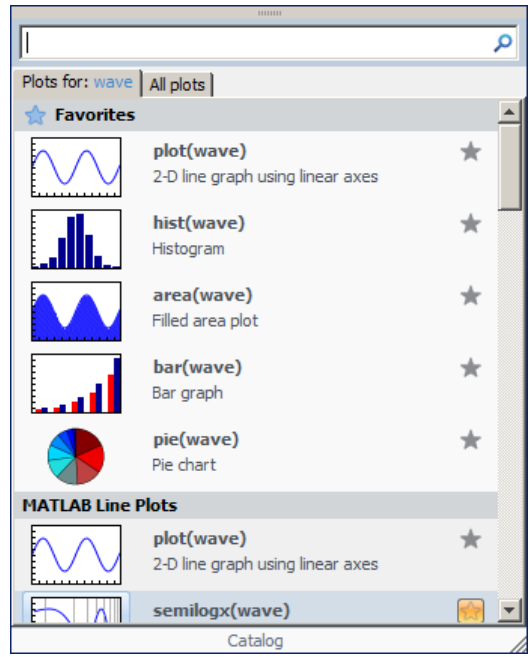
1. Create two vector variables:

```
run = 0:.1:4*pi;
wave = (sin(run.^2) + cos(run).^2);
```

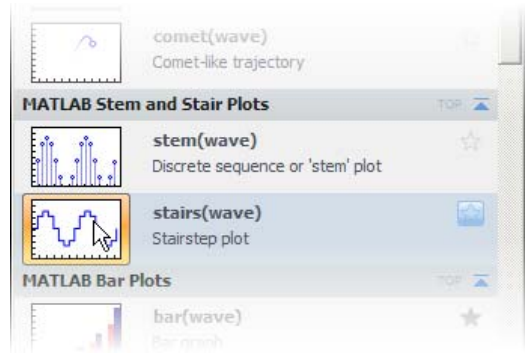
2. Select one or both variables in the Workspace Browser. You can **Shift**+click or **Ctrl**+click to select multiple variables to plot as *x*, *y*, or *z* components of a graph.



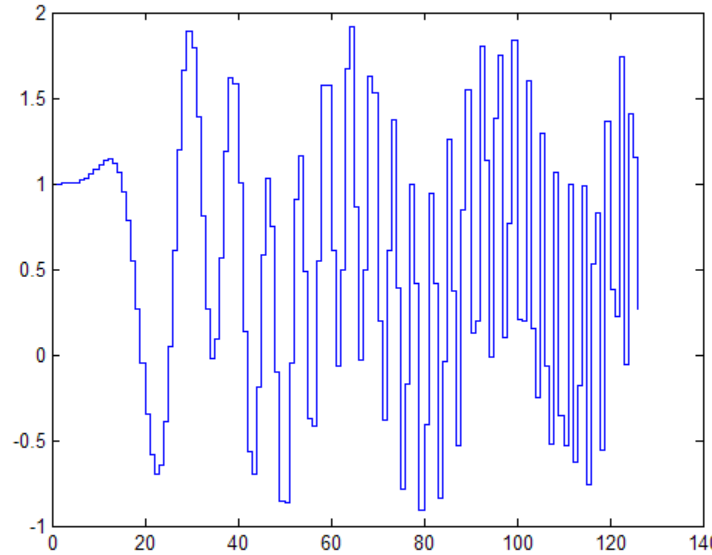
3. Click the down arrow icon in the Plot Selector on the toolbar. The Plot Selector menu opens.



4 Scroll to the graph type you want to display and click the icon on its left side. The icon highlights when you hover over it.



A figure window opens with a graph of the selected variable or variables, using the plotting function you just chose.

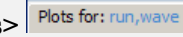


The Plot Selector GUI remains open until you click a desktop component or another window. This enables you to experiment with plot types without reopening the GUI each time.

Working with the Plot Selector GUI

You can interact with the Plot Selector in many ways. Learn about using it by viewing this video demo. Also, read the following table to better understand what you can accomplish with the Plot Selector and how to do it.

What You Can Do	How To Do It
Identify variables to graph.	Select a variable of numeric type in the Workspace Browser. To add variables to your selection, **Shift**+click or **Ctrl**+click.
Plot selected variables immediately using the default graph type.	Click the Plot Selector icon; a graph of the type it displays will plot using the selected variables.

What You Can Do	How To Do It
Plot selected variables after choosing a graphing function.	Click the arrow  to the right of the Plot Selector button. The GUI opens for you to choose a graph type. Create plot by clicking the chosen item's picture icon.
Interchange two variables before graphing them.	Click the Reverse Input Variable Order button  . The two function arguments the item displays interchange. The button appears only when you select two variables.
Identify the graphing functions you can currently use to create a plot.	With one or more selected variables, select the first tab, **Plots for:<variables>**  and do either of the following: • Scroll through the menu or navigate through it with up/down arrow keys. • Enter a search term in the search bar; only those menu items and categories containing the search string will display.
Identify available graphing functions.	With zero or more variables selected, select the second tab, **All plots**  and do either of the following: • Scroll through the menu or navigate through it with up/down arrow keys • Enter a search term in the search bar; only those menu items and categories containing the search string display

What You Can Do	How To Do It
Add a graph type to Favorites.	In either tab: • Click the star to the right of a menu item and do any of the following: • Right-click a menu item and select **Add to Favorites**. • Right-click a menu item and select **Copy**,and then right-click within **Favorites** and select **Paste**. • Drag a menu item from its category to **Favorites**. All of these actions create a copy the item in **Favorites** without removing it from its category.
Remove a graph type from Favorites.	In the **Favorites** category of either tab do either of the following: • Click the star at the right side of a menu item • Right-click a menu item and select **Remove Favorite** You cannot drag a menu item out of **Favorites**.
Change the order of menu items within a category.	Click a menu item outside the graph icon and drag it to a new position within its category.
Move a menu item to a different category.	Click a menu item outside the graph icon and drag it to a new category. Dragging an item away removes that item from the category it was in. This gesture does not apply to items within **Favorites**.

What You Can Do	How To Do It
Get help for a graphing function.	On either tab, hover mouse pointer over a menu item. A syntax description pops up next to the item. To see the complete function reference page, click the *More Help* link in the description header.
Enable a dimmed menu item.	Select one or more variables appropriate for calling the dimmed function. Then, open the Plot Selector tool again. See "Selecting Appropriate Variables" on page 5-15 and "Determining What Inputs a Graphing Function Needs" on page 5-16.
Manually execute or modify a plotting call.	Drag a menu item (even if it is dimmed) to the Command Window or the Editor; the code for that plot displays and you can edit it. See "Editing a Plot Selector Graphing Command" on page 5-17.
Display the Plot Selector as a window.	Open the Plot Selector, click the grab bar  at the center top, and drag the GUI to where you want it on the screen. That window closes if you click another MATLAB window. The window remains open if you focus on a window of another application.
Access the Plot Catalog.	Click the Catalog link at the bottom of the Plot Selector tool.

Selecting Appropriate Variables

If you select insufficient or inappropriate variables for a graph type, its menu item is dimmed on the **All plots** Plot Selector tab. The dimmed items do not appear at all on the **Plots for:** tab. Dimmed and missing Plot Selector menu items indicate that you cannot use the function for plotting the selected data via the tool. Reasons why a function is not available include:

- You selected a variable having the wrong class for graphing (it is not a numeric type).
- The graphing function requires additional inputs (for example, `scatter` requires two vectors).

- The graphing function requires fewer inputs than you selected (for example, `plot` cannot handle three vectors)
- Your selected variable has inappropriate type or dimensions (for example, matrices displayed by `feather` must be complex).
- You selected variables in the wrong order (for example, selecting scalar `n` and then matrix `Z` to `contour(n,Z)` instead of `contour(Z,n)`).

If you select variables that no function can display, the Plot Selector button label changes to **No valid plots for:<variable>** 

. If you click the button, you see the message "No plot available for selection. Change your variable selection or click the **All plots** tab to browse for plots."

Determining What Inputs a Graphing Function Needs

When a Plot Selector menu item is unavailable (dimmed) in the **All plots** tab, you can learn why by viewing the pop-up help for that function. Suppose you want to make a `semilogy` graph.

In the **All plots** tab, you can do several different things:

1. Create a nonnumeric variable, for example, a string.

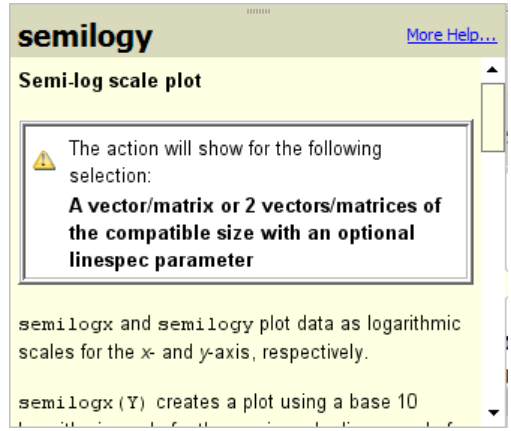
   ```
   name = 'abcd';
   ```

2. Select `name` in the Workspace Browser and open the Plot Selector tool by clicking its down arrow.

3. Click the **All plots** tab in the Plot Selector window.

4. Scroll to the **semilogy** menu item (all items are dimmed) and hover the mouse pointer over the item.

   A help message appears that includes a note specifying what inputs the `semilogy` function accepts. The pop-up help window contains a white box that describes the function's input requirements, as shown in the following figure.



The message helps you to understand what arguments the `semilogy` function supports.

A Help Window opens whenever your mouse pointer lingers over a menu item, but information set off in a white box only appears when the function is dimmed because it cannot generate a graph of the currently selected variables.

Editing a Plot Selector Graphing Command

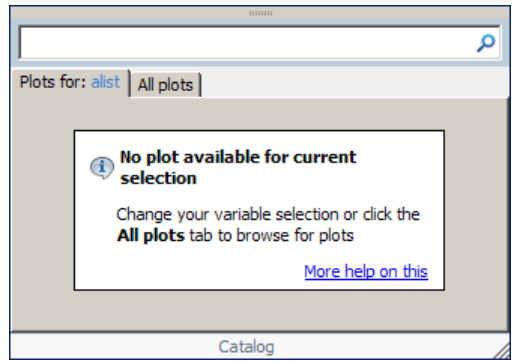
Even when the Plot Selector tool cannot successfully make a graph, it can still give you a starting point for doing so. You can drag any Plot Selector menu item into the Command Window to see the code it generates. Once the plotting command displays in the Command Window, it does not execute until you press **Enter**, enabling you to edit its arguments first. Dragging dimmed items works the same as dragging undimmed items. This enables you to fix problems due to selecting the wrong variables or selecting them in the wrong sequence. You can also add arguments to the plotting command, for example to specify a `linespec` or a `colorspec` for functions that can use them.

The following example illustrates how to drag a dimmed Plot Selector menu item, drop it into the Command Window, and edit the plotting command before executing it.

1 In the Command Window, enter

```
alist = '1 2 3 4 5 6 7 8 9 10';
ilist = [1 2 3 4 5 6 7 8 9 10];
```

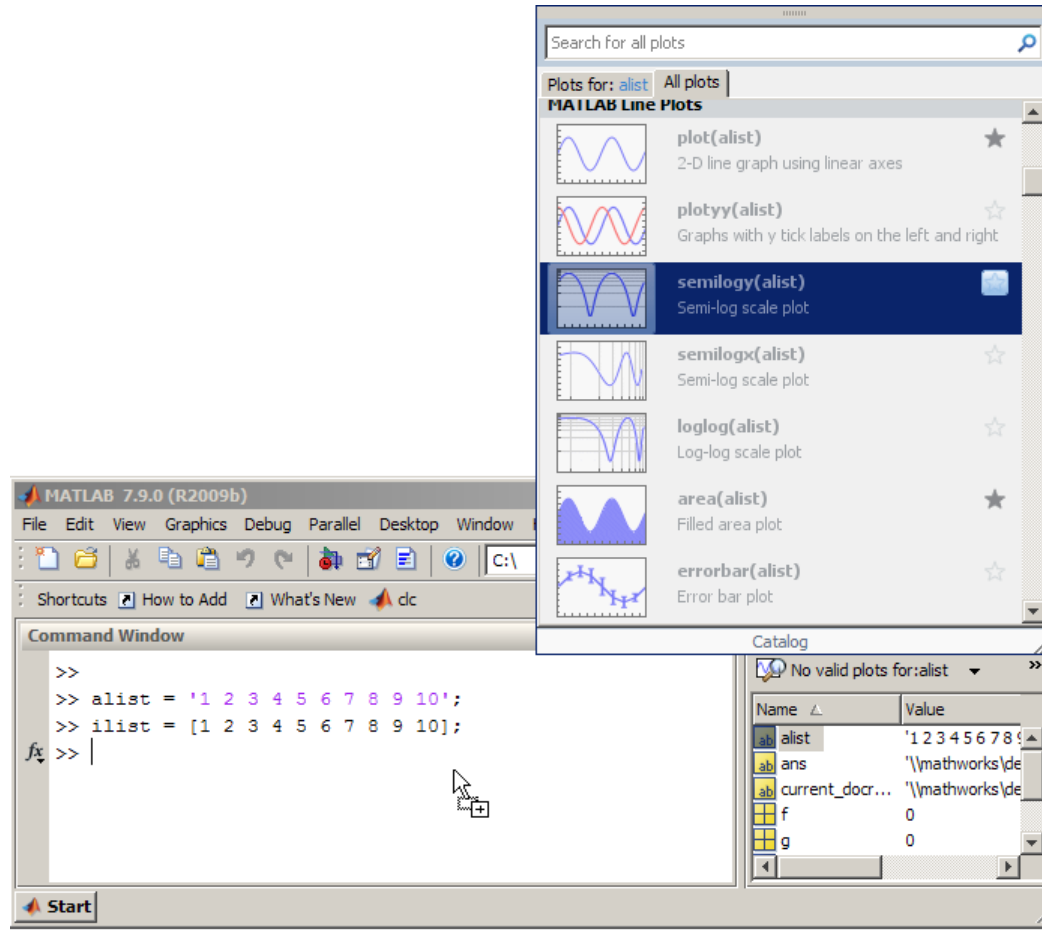
2 Assume that you want to create a graph of `ilist` but instead you select the string variable `alist` in the Workspace Browser. When you click the Plot Selector, it displays this error message.



You can go on to plot `ilist` by continuing as follows.

3 Click the **All Plots** tab and scroll to `semilogy`. The menu item is dimmed.

4 Press the mouse button over `semilogy` and drag it to the command window, as shown here.



5 Release the mouse button in the Command Window The following line of code appears there:

```
>> semilogy(alist,'DisplayName','alist');figure(gcf)
```

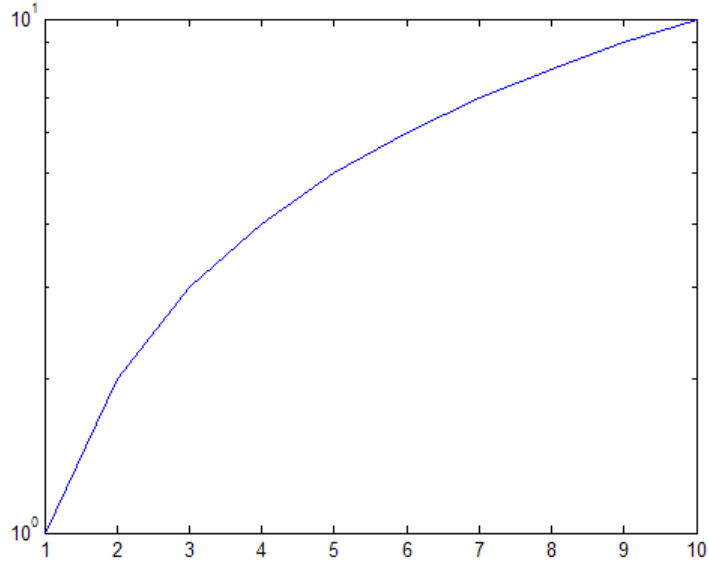
If you press **Enter**, an error displays because `alist` is not a proper calling argument:

```
??? Error using ==> semilogy
Invalid first data argument
```

6 Instead, change the code to the following, substituting `ilist` for `alist`:

```
semilogy(ilist,'DisplayName','ilist');figure(gcf)
```

This call works with the changed arguments and generates the following graph after you press **Enter**.



You can also edit the plotting command to add other arguments, including parameter/value pairs, to customize the graph.

For More Information

If you want to use the Plot Selector to create a graph for a subrange of a vector or a matrix, you can open the variable in the Variable Editor, select the range of data you want to graph, and open the Plot Selector from the Variable Editor toolbar. All data for the graph must all come from one variable and the subrange to plot has to be a contiguous selection.

You can add titles, axis labels, legends and other annotations to graphs that the Plot Selector makes. For more information about graphing data and customizing plots, see the following topics in the *MATLAB Graphics* documentation:

- “Figures, Plots, and Graphs”
- “Plotting Tools — Interactive Plotting”
- “Basic Plotting Commands”
- “Creating Specialized Plots”

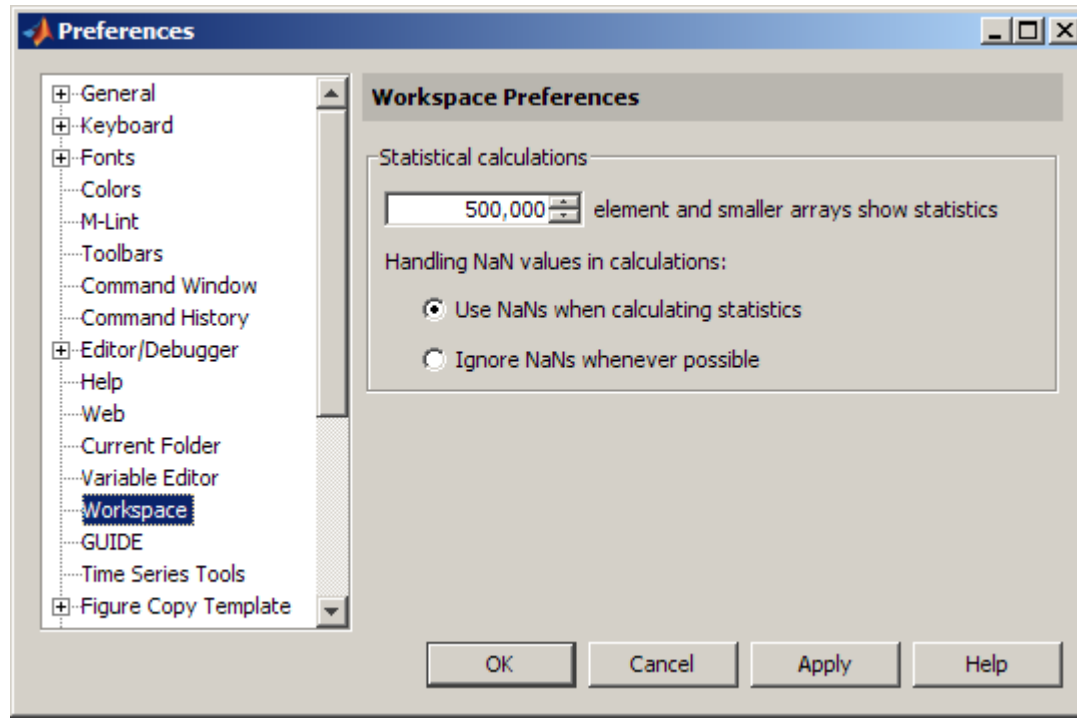
Opening Variables and Objects for Viewing and Editing

In the Workspace browser, double-click a variable and it opens in the Variable Editor where you can view and edit the contents of the variable. See “Viewing and Editing Workspace Variables with the Variable Editor” on page 5-24 for more information.

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.

Setting Workspace Browser Preferences

The Workspace browser displays statistical calculations for variables. Use preferences to restrict the size of arrays on which you perform calculations and to specify if you want those calculations to include or ignore NaNs. Select **File > Preferences > Workspace** to open the dialog box. Make your 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**; **Var**, however, 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 5-4.

Viewing and Editing Workspace Variables with the Variable Editor

In this section...

About the Variable Editor

The Variable Editor is a desktop component that lets you display variables in the current workspace. Use it to view and edit values of one or two-dimensional arrays, character strings, cell arrays, structures, and objects and their properties. You can also view the contents of multidimensional arrays. Edits you make in the Variable Editor immediately update the variable in the workspace. It also supports copying and pasting of data values.

Opening the Variable Editor

To open the Variable Editor from the Workspace browser, perform these steps:

1 If you do not have any, create some workspace variables, for example:

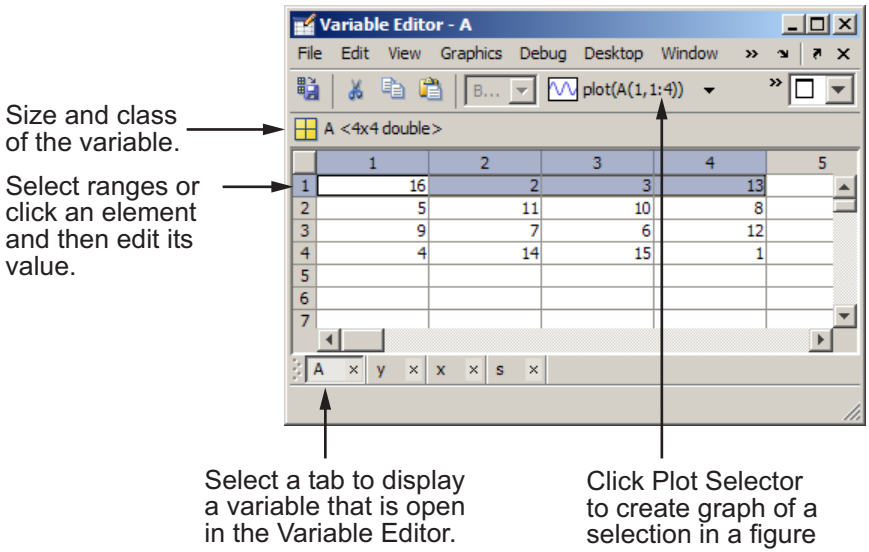
```
A = magic(4);
x = 0:.1:4*pi;
y = sin(x);
```

```
s = sprintf('This is yext\nwith two lines');
```

2. In the Workspace browser, select the variable you want to open. Use **Shift**+click or **Ctrl**+click to select multiple variables, or use **Ctrl+A** to select all variables to open.

3. Click the Open Selection button on the toolbar. For one variable, you can also open it by double-clicking it.

   The Variable Editor opens, displaying the values for the selected variable. The class and size of the value appear below the toolbar, and for some classes, include a link to the help for that class.



Repeat the steps to open additional variables in the Variable 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 Variable Editor.

Note MATLAB software does not limit the maximum number of elements in a variable that you can open in the Variable Editor. The limit is based on your operating system or the amount of physical memory installed on your system.

Keyboard Alternatives

To open a variable in the Variable Editor, use `openvar` with the name of the variable you want to open as the argument. For example, type

```
openvar('A')
```

You need to enclose the name of the variable name in single quotes, because the Variable Editor requires strings, not variable references. It needs the name of the variable so MATLAB can notify it when the variable changes value, disappears, or goes out of scope. If you were to type `openvar(A)` instead of `openvar('A')`, the Variable Editor would receive the value of `A` instead of its name. However, `openvar varname` and `openvar 'varname'` both work, as the function assumes string arguments when using *command syntax*. See "Command vs. Function Syntax" in the MATLAB Programming Fundamentals documentation for more information.

MATLAB opens `A` in the Variable Editor.

To see the contents of a variable in the workspace, type the variable name at the Command Window prompt. For example, type

```
A
```

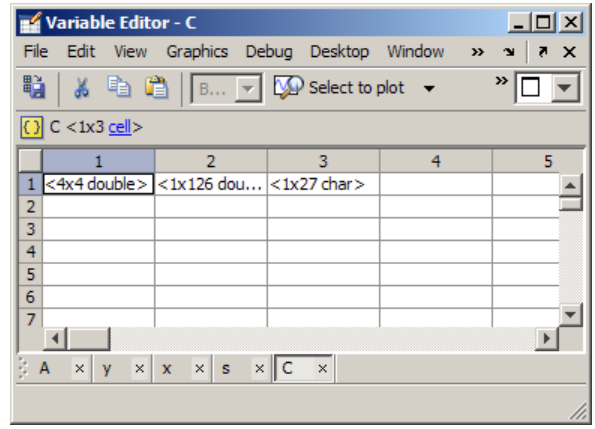
and MATLAB returns

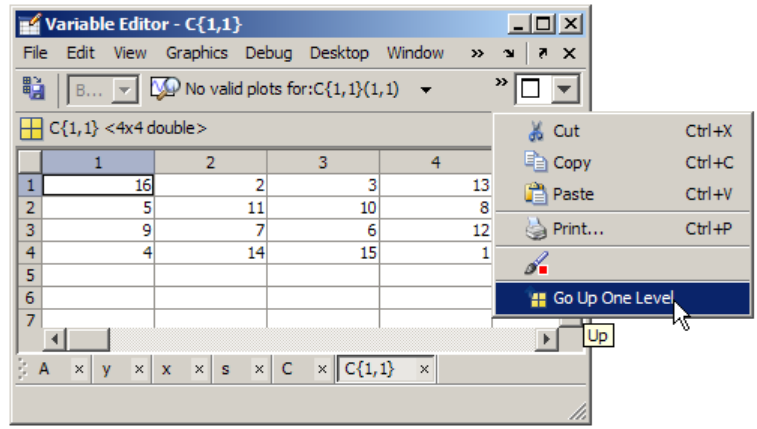
```
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Working with Different Types of Data in the Variable Editor

Cell Arrays — Viewing and Editing in the Variable Editor

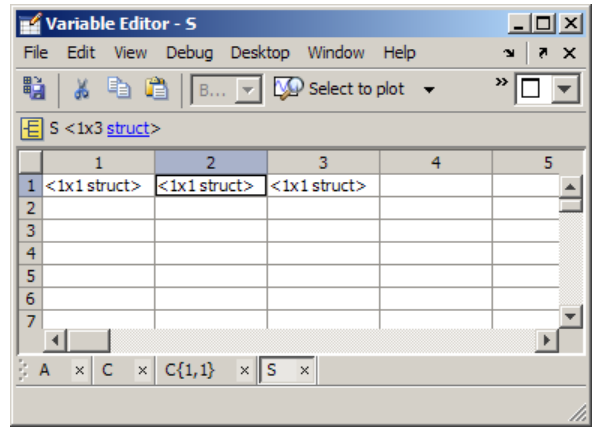
You can view and edit the content of cell arrays in the Variable Editor. In the Variable Editor, double-click an element of a cell array to open it as its own Variable Editor document. You can then view and edit the contents of that element. The following illustrations show a 1-by-3 cell array, `C`, and the contents of `C{1,1}`. When viewing an element in a cell array, for example, `C{1,1}`, use the Up button to go to its cell array, for this example, `C`.

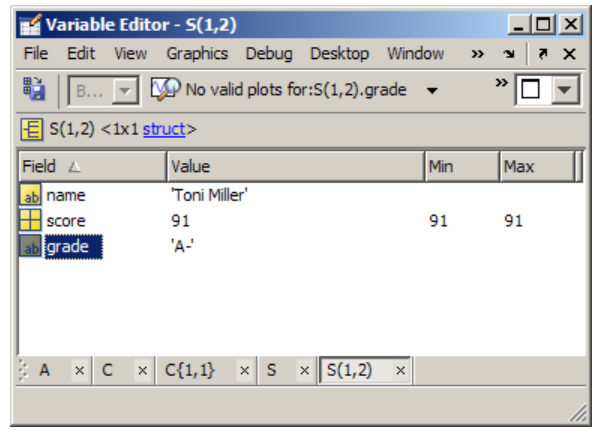




Structures — Viewing and Editing in the Variable Editor

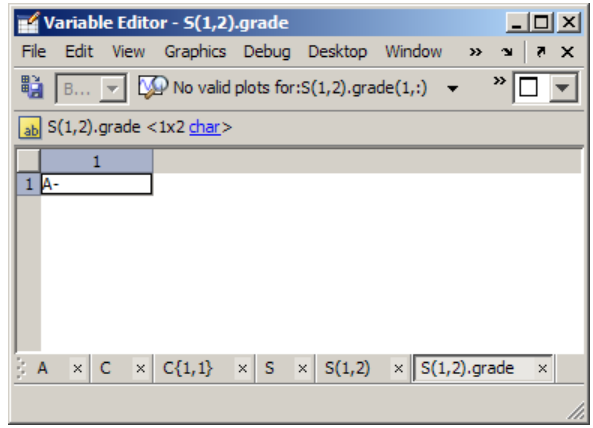
You can view and edit the content of structures in the Variable Editor. In the Variable Editor, double-click an element of a structure to open it as its own Variable Editor document. The following illustrations show a 1-by-3 structure, `S` and the result of double-clicking `S(1,2)`, which displays the contents in its own new document.





The information shown for the element of the structure is like what the Workspace browser displays: **Field**, **Value**, **Size** and other information. To show or hide columns, select **View > Choose Columns**. On Microsoft Windows systems, you can right-click any column header to hide it or to show or hide other columns. Click a column header to sort by that column, and click again to reverse the sort order. When viewing an element in a structure, for example, `S(1,2)`, use the Up button to view the structure, for this example, `S`. The button helps you navigate in the Variable Editor when there are many variables open.

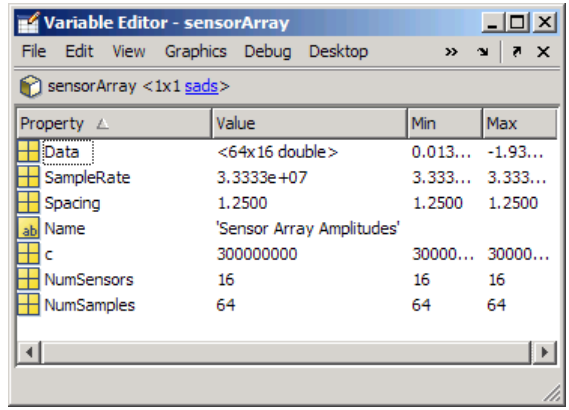
To edit an element, you can click the value and change its value. Or double click the element; a new Variable Editor document containing it opens. Click the value and then change it. The following illustration shows the result of double-clicking the `grade` field for `S(1,2)`, where you can change its value. You can use the Up button go up from the field to view the element. For example, when viewing `S(1,2).grade`, click the Up button to view `S(1,2)`.



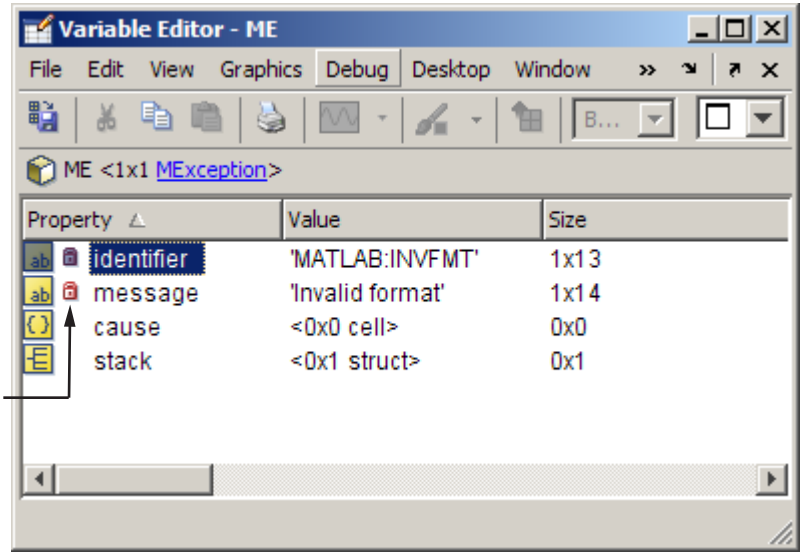
Objects and Their Properties — Viewing and Editing in the Variable Editor

Viewing Object Properties in the Variable Editor. In the Variable Editor, you can view and edit properties of many MATLAB objects you create. When you open an object in the Variable Editor, it displays **Property**, **Value**, **Size**, and other information. To show or hide a column, right-click the column header. To sort by a column, click that column header; to reverse the sort order, click the column header again. You can view help for the class by clicking the class name link. The Variable Editor has special attributes for `timeseries` objects; for more information, see "Viewing Time Series Objects" in the MATLAB Data Analysis documentation.

The following illustration shows the `sensorArray` object of the `sads` class, in the Variable Editor. For more information about this example, see "Example of Help for a User-Created Class" on page 4-31.

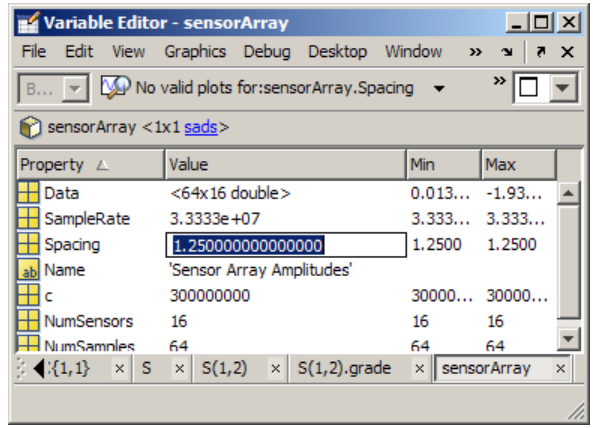


Additional icons, images of locks, denote protected and private properties of an object, indicating you cannot edit the values. The following illustration shows an `MExcception` object, `ME`, with the private properties `identifier` and `message`.

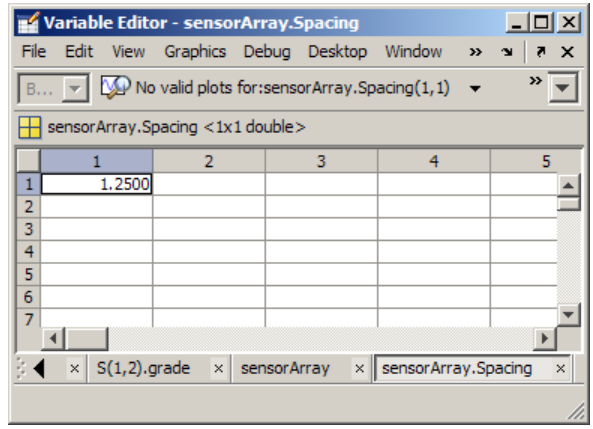


Icon denotes that `identifier` and `message` properties are private, and therefore you cannot edit their values.

Editing Property Values in the Variable Editor. To edit a property value while viewing the object, click the value field and edit its contents, as shown in the following illustration.



Alternatively, double-click the value, which displays the value in its own document where you can more easily view and edit it. In the following illustration, the `sensorArray.Spacing` property opens in its own document when you double-click it. When viewing a property, use the **Up** button to view the object, for this example, `sensorArray`. This button can help you navigate in the Variable Editor when there are many variables open.

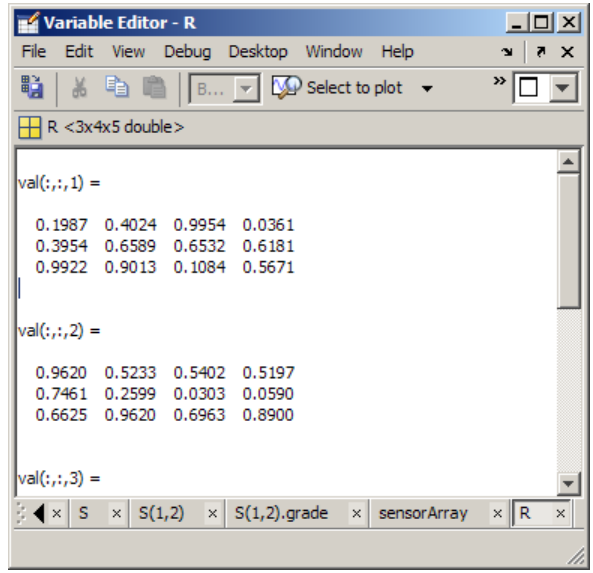


If the Variable Editor window is small, the **Up** button might not be visible. To get to it, click the >> More Menu button on the right side of the toolbar and choose **Go Up One Level** from the menu.

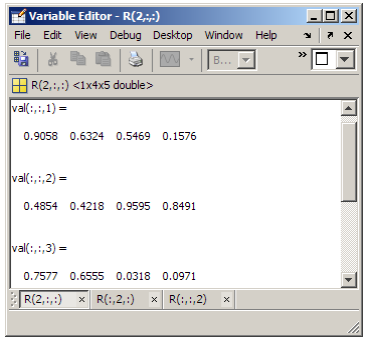
Getting Help for Objects and Properties from the Variable Editor. For most classes supplied by The MathWorks, when you click the link to the class name, for example, `char`, the reference page displays in the Help browser. For user-created classes, help comments supplied in the class definition file display in HTML format in the Help browser. For more information, see "Providing Help for Classes You Create" on page 4-31.

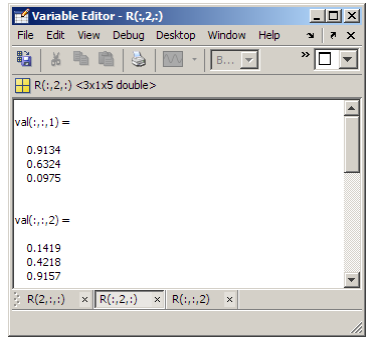
Multidimensional Arrays — Viewing in the Variable Editor

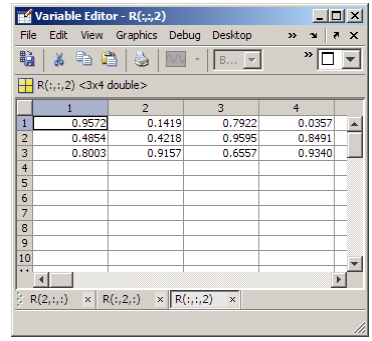
You can view the contents of multidimensional arrays in the Variable Editor. When you open a multidimensional array in the Variable Editor, it does not have usual grid structure, because multidimensional arrays do not fit that format. You cannot double-click an element in a multidimensional array to edit it. The following illustration shows `R = rand(3,4,5)` opened in the Variable Editor.



You can view subsets of multidimensional arrays as long as the indexing expression evaluate to either a 1-D vector or a 2-D matrix. For example, `R(2,:,:)` , `R(:,2,:)`, and `R(:,:,2)` display as follows.







You cannot edit subsets of multidimensional matrices. Because you can index into matrices in so many ways, the Variable Editor can incorrectly identify subscripts of variable elements that you might change. To avoid changing the wrong data elements, the Variable Editor prevents you from editing multidimensional matrices.

Navigating and Editing Shortcut Keys for the Variable Editor

Use the following shortcut keys (sometimes called hot keys) to move among elements in the Variable Editor. Navigating in the Variable Editor is much like navigating in the Microsoft Excel application.

Key	Result
Enter	Commit any changes to the element and move to next element. "Preferences for the Variable Editor" on page 5-46 let you specify what the next element is (the default is down)
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**

Key	Result
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 Apple Macintosh platforms)	Edit current element, positioning cursor at the end of the element

Changing Size, Content, and Format of Variables in the Variable Editor

To increase the size of an array, scroll to the desired element in the variable and enter a value. The array automatically expands to accommodate the new value. Empty elements fill 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. Clicking a header selects the entire row or column. Then right-click, and select **Delete** from the context menu. Similarly, you can update arrays in structure and objects.

To change the value of an element in the Variable Editor, click the element and type a new value. Press **Enter**, or click another element to effect the change. You can specify where the cursor moves to after you press **Enter** — see “Preferences for the Variable Editor” on page 5-46.

If you want to change the display format for the Variable 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 Variable Editor” on page 5-46.

If you open an existing MAT-file and change it using the Variable Editor, save that MAT-file if you want the changes to be permanent. For instructions, see “Saving the Current Workspace” on page 5-5.

Cut, Copy, Paste, and Clear Contents in the Variable 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 the row or column header (the element that shows the row or column number). Use **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 Variable 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 **Edit > Paste**. If the shape of the elements you cut differs from the shape of the elements into which you are pasting, the Variable Editor pastes all the elements. Either it expands the size of the selection you made, or it expands the array size to allow all the elements you are pasting. Pasting copied elements is the same as pasting cut elements, but the elements copied maintain their value rather than becoming `0` or `[]`.

To make the value of elements `0`, select elements, rows, or columns and then select **Edit > Clear Contents**. Clearing differs from cutting because the data from the selected elements does not move to the clipboard or modify it.

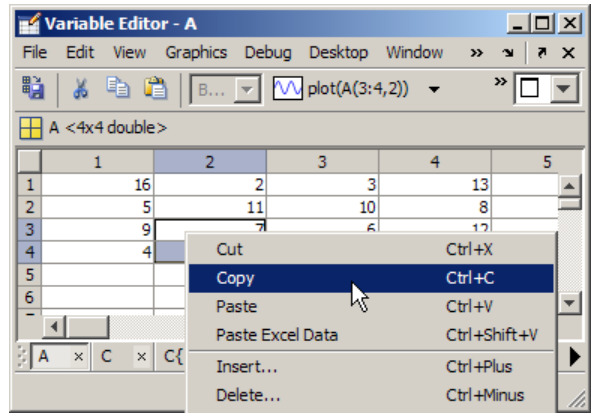
Example: Copying and Pasting Array Elements

In this example, you copy two elements. When you select one element for pasting, it replaces two elements.

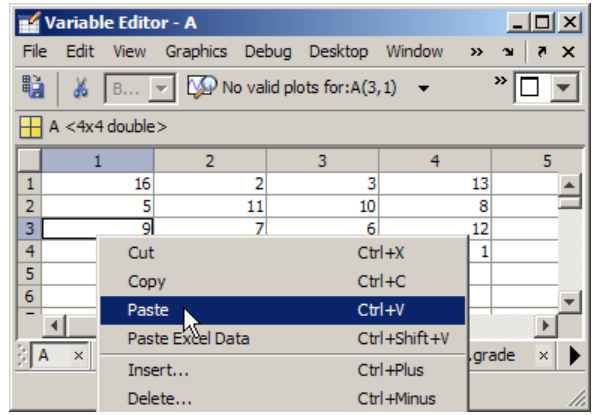
1. Create a matrix variable.

```
A = magic(4);
```

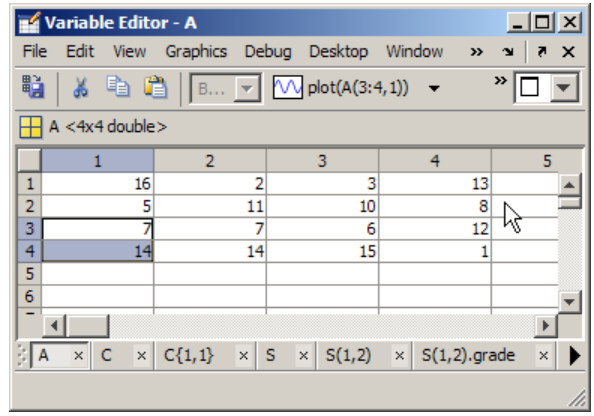
2. Select rows 3 and 4 or column 2 by clicking 7 (`A(3,2)`) and then **Shift**+clicking 14 (`A(4,2)`). Right-click the selection and select **Copy** from the context menu. You can also press **Ctrl+C** or chose **Copy** from the **Edit** menu to copy the values.



3. Select 9 (`A(3,1)`) and select **Paste** from the context menu or **Edit** menu, or type **Ctrl+V**.



The column vector you copied (`[7;14]`) replaces the contents of rows 3 and 4 in column 1 (which had been `[9,4]`), even though you only selected the element containing 9. That is, the shape of the copied elements determines which values get replaced, starting at the upper left element.



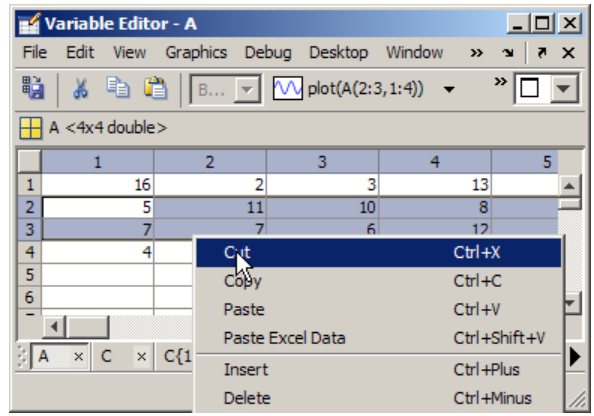
Example: Cutting and Pasting Array Elements

select two rows and cut their contents. Select one row for pasting. The Variable Editor expands the array size, adding a row to accommodate all cut elements. The values of the elements you cut becomes `0`.

1 Create a matrix variable.

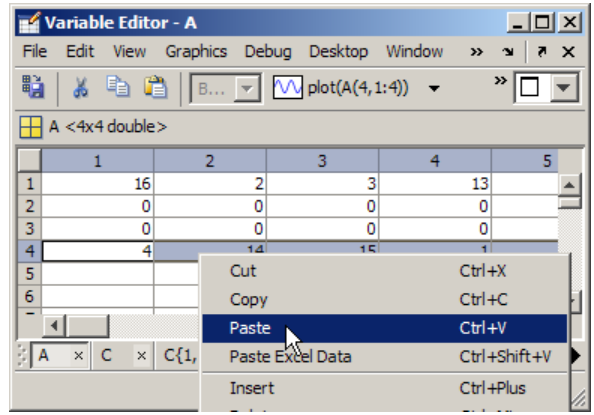
```
A = magic(4);
```

2 Select rows 2 by clicking its row number, then **Shift**+click the row number for row 3 to select both rows. Right-click the selection and select **Cut** from the context menu. You can also press **Ctrl+X** or chose **Cut** from the **Edit** menu to cut the values and copy them to the clipboard.

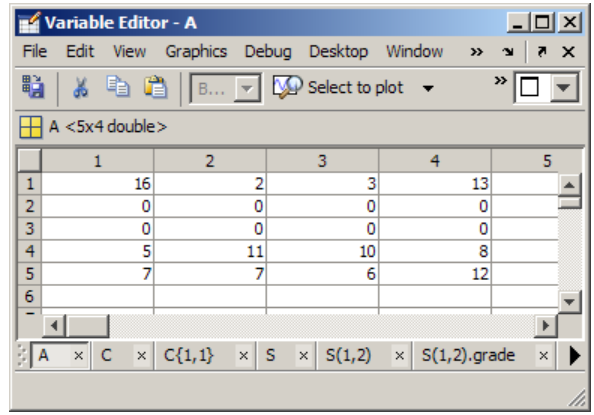


The values in the cut rows all become 0 as a result of the cut operation.

3 Select row 4 entirely and select **Paste** from the context menu or **Edit** menu, or type **Ctrl+V**.



The contents of the cut rows replace row 4 and extend the matrix to have an additional row.



Other Variable Editor Operations

Insert and Delete in the Variable Editor

You can insert and delete elements, rows, and columns in arrays in the Variable Editor. When you select **Edit > Insert**, or **Edit > Delete**, a dialog box appears in which you specify rows, columns, or elements. When you delete elements, the Variable Editor prompts you to provide, the direction for shifting existing elements.

Undo and Redo in the Variable Editor

You can undo the last action you performed in the Variable Editor, or redo a change after choosing undo. Select **Edit > Undo** or **Edit > Redo**. The actions supported are the following:

- A change to a value you make by editing it in the Variable Editor
- Cutting
- Pasting
- Inserting
- Deleting
- Clearing contents
- Pasting data from the Microsoft Excel application.

Exchanging Data with the Command Window

You can copy data from an array in the Variable 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 Variable Editor. Be sure that the data types are compatible. For example, you cannot paste text from the Command Window into a numeric array in the Variable Editor.

Creating New Workspace Variables from the Variable Editor

You can also create new variables from a selected element, data range, row, or column in an array in the Variable Editor. Right-click, and from the context menu, select **Create Variable from Selection**, or do the same from the **Edit** menu.

Exchanging Data with the Microsoft Excel Application

You can cut or copy cells from the Microsoft Excel application and paste them into the Variable Editor—use **Edit > Paste from Excel**. You can also cut or copy elements from an array in the Variable Editor and paste them into the Excel® application.

Be sure that the data types are compatible. For example, you cannot paste text from the Excel application into a numeric array in the Variable Editor.

Creating Graphs and Variables, and Data Brushing in the Variable Editor

The Variable Editor, like the Workspace Browser, provides several methods for creating graphs without typing plotting commands. Once a graph displays, you can "brush" either the graph or array elements in the Variable Editor to see which observations correspond in the other.

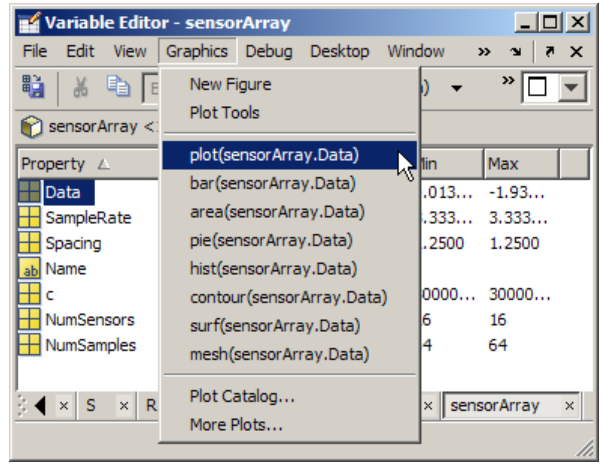
Generating Graphs Automatically

You can create graphs from selected variables in the Variable Editor. To create a graph, select a data range, row, or column in an array, and choose a graph type in one of the ways described in the following bullets. MATLAB examines the selected data and determines which kinds of graphs can display

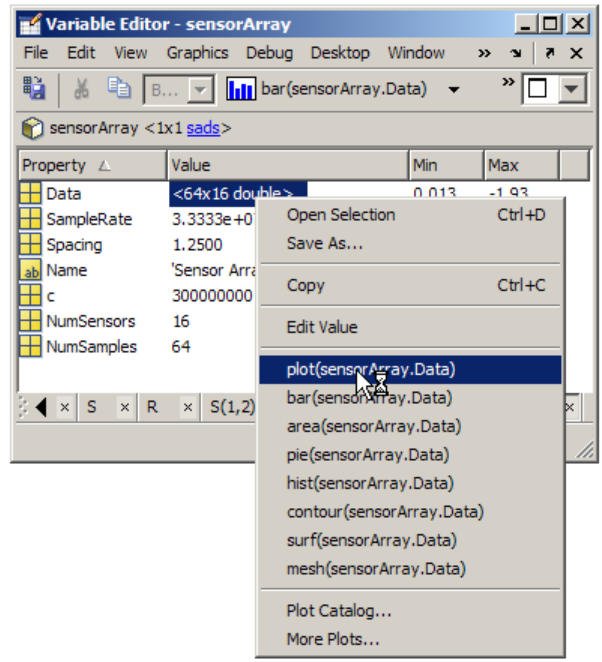
it. In some cases, MATLAB performs data conversion, such as using `cell2mat` to transform cell array data—which cannot be plotted directly—to matrix data. For more information, see "Plotting Process" in the MATLAB Getting Started Guide.

You can graph selections of numeric data and selected objects from the Variable Editor in three ways, illustrated here:

- Select data and choose from a list of graph types from the **Graphics** menu.

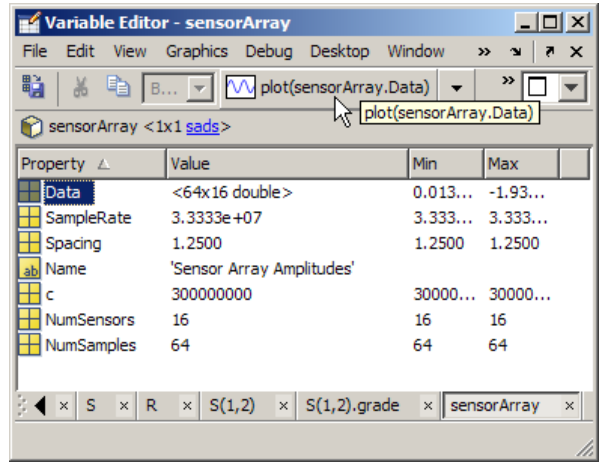


- Select data, right-click, and choose from a list of graph types from the context menu.

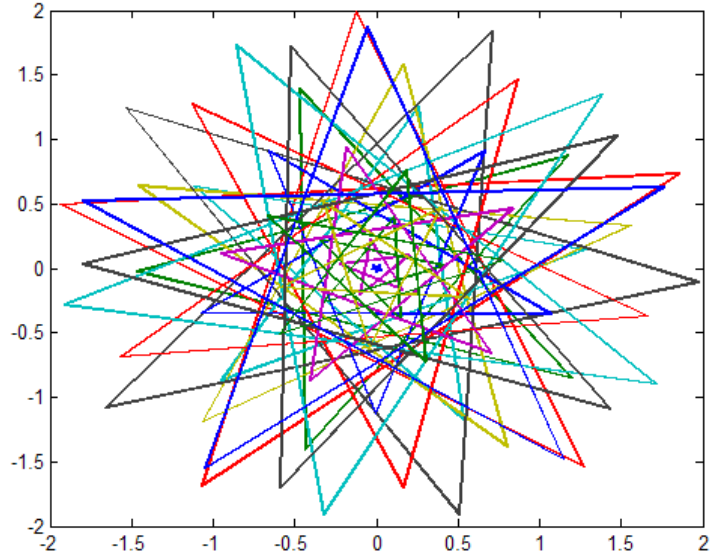


The types of graphs available on the context menu and the **Graphics** menu are the same.

- Select data and click the Plot Selector toolbar icon to generate the type of plot it displays.

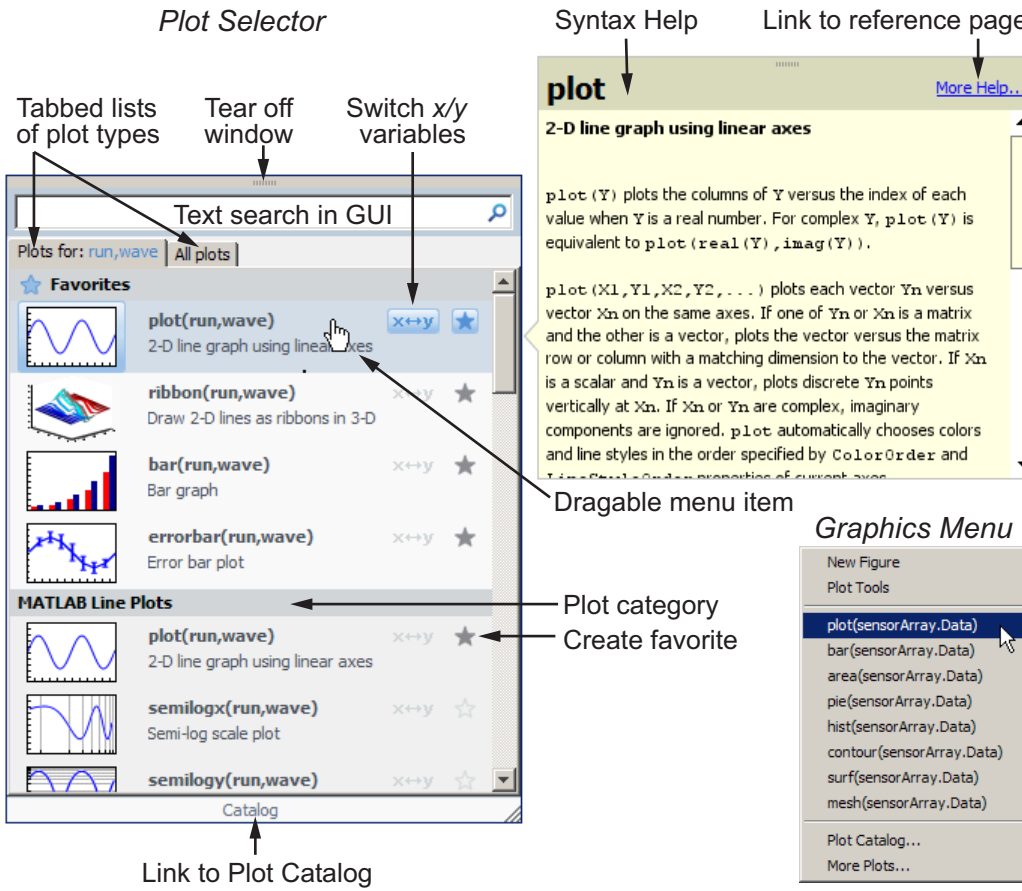


Assuming that you select the same graph type, all three methods generate identical plots of the selected data in the current or a new figure window.



The Plot Selector is the most flexible of the three methods. It lists more graph types you can currently make and, in a separate tab, all graph types available

to you. It also provides function help, and lets you prioritize graph types as a list of favorites. The following illustration compares it to the Graphics menu.



For more information about using the Plot Selector, see “Creating Plots from the Workspace Browser” on page 5-10.

Brushing Data in Linked Graphs

Data brushing is a technique for exploring where specific data observations fall in a set of graphs and tables. It helps you to visually identify relationships, outliers, trends, and noise that can be difficult to determine with numerical or statistical methods. Use the Data Brushing Tool  on the Variable Editor

and figure toolbars to mark specific observations (or ranges of them) in the Variable Editor and on graphs. You can remove brushed observations or save them to new variables.

If a variable you brush in the Variable Editor is plotted on a graph, selecting the Data Brushing tool and brushing array elements in the Variable Editor highlights those values in the graph displaying the variable you brush. Likewise, brushing observations on a linked plot highlights them in the Variable Editor. For data brush to communicate between the two windows, the figure must be in *Linked Plot*  mode. Linked Plot mode connects a graph's `XData`, `YData` and `ZData` to its data sources in the current workspace. For more information, see "Data Brushing with the Variable Editor" in the MATLAB Data Analysis documentation and the reference pages for `brush` and `linkdata`.

Preferences for the Variable Editor

To set preferences for the Variable Editor, select **File > Preferences > Variable Editor**. The Preferences dialog box opens showing **Variable Editor Preferences**.

Format

Specify the default array output format of numeric values displayed in the Variable Editor. This format preference affects only how numbers display, 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 an element and press **Enter**:

- If you want the cursor to remain at the element where you 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. Choose the **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 the decimal format of numbers you cut or copy from the Variable Editor when you paste them into text files or other applications. The **Decimal separator for exporting numeric data via system clipboard** 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**. This preference has no effect on numeric data copied from and pasted into MATLAB documents or into the Command Window. Within MATLAB, decimal separators are always periods.

6

Managing Files in MATLAB

How MATLAB Helps You Manage Files

MATLAB provides tools and functions to help you:

- Find a file you want to use with MATLAB
- Organize your files
- Ensure MATLAB can access a file so you can run or load it. Typically, any file you want to access must be in the MATLAB current folder or in a folder that is on the search path.

For a more complete introduction, see the “Managing Files in MATLAB” topic in the MATLAB Getting Started guide.

Understanding Important Folders and Path Names in MATLAB

In this section...

Important Folders MATLAB Uses

When you work with files and folders, be aware of key locations that MATLAB uses.

The Current Folder in MATLAB

- Is a reference location that MATLAB uses to find files.
- Is *not* the same location as the operating system current folder.
- Is sometimes referred to as the current directory, current working folder, or present working directory

matlabroot

`matlabroot` is the location where you installed . The location differs for each installation of MATLAB. Determine the location by running the `matlabroot` function.

Folders for MathWorks Products

Files and folders for products provided by The MathWorks are in *`matlabroot`*`/toolbox`:

- Do not change files, folders, and subfolders in *`matlabroot`*`/toolbox`.
- Do not store your files and folders in *`matlabroot`*`/toolbox`. The exception is the `pathdef.m` file, which you can change and save in its default location, *`matlabroot`*`/toolbox/local`.

To improve performance, at the beginning of each session, MATLAB loads and caches in memory the locations of files in *matlabroot*/toolbox. If you make changes to files and folders in *matlabroot*/toolbox, you can get unexpected results or see warnings related to the cache. See "Toolbox Path Caching in the MATLAB Program" on page 1-22.

To see a list of all toolbox folder names supplied with MathWorks products, run:

```
dir(fullfile(matlabroot, '/toolbox'))
```

Locations for Storing Your Files

For your convenience, MATLAB provides a MATLAB folder to store your files. At startup, MATLAB adds the folder to the search path, allowing MATLAB to access the files stored there.

The location of the MATLAB folder, referred to as the *userpath*, varies by platform and system configuration. To determine the location, run the userpath function.

On Microsoft Windows platforms, MATLAB sets the current folder to *userpath* at startup. On other platforms, instruct MATLAB to set the current folder to *userpath* at startup. For more information, see:

- "Startup Folder for the MATLAB Program" on page 1-11
- "Specifying the Current Folder at Startup" on page 6-45

If you create subfolders within MATLAB, make the subfolders accessible to MATLAB.

If you store files in locations other than MATLAB:

- Make the files accessible to MATLAB.
- Do not store the files in the folders provided for MathWorks products.

Path Names in MATLAB

When you work with files and folders, be aware of how MATLAB uses path names.

File Separator Characters, / and \

The file separator character is the symbol that distinguishes one folder level from another in a path name.

A forward slash (/) is a valid separator on any platform. A backward slash (\) is valid only on MicrosoftWindows platforms.

In the full path to a folder, the final slash is optional.

To use the file separator character when working with files programmatically, see `filesep`.

Path Names on Different Platforms

Use `fullfile` to construct path names in statements that work on all platforms. Use the `ismac`, `ispc`, and `isunix` functions when the path names differ depending on the platform.

Spaces in Path Names

When a function argument is a file or path name, and the name includes spaces, use the function syntax. For example:

```
delete('my file.m')   % Function syntax works for a file name containing a space
```

The command syntax does not work. For example:

```
delete my file.m   % Command syntax does NOT work for a file name containing a space
```

Absolute and Relative Path Names

MATLAB always accepts absolute path names, also called full path names, such as `I:/Document/My_Files`. An absolute path name can start with any of the following:

- UNC path `'\\'` string
- Drive letter, on Microsoft Windows platforms
- `'/'` character on UNIX[15] platforms

15. UNIX is a registered trademark of The Open Group in the United States and other countries.

Some MATLAB functions also support relative path names. The reference page for a function specifies the valid types of path name. Unless otherwise noted, the path name is relative to the current folder. For example:

- `/myfolder` refers to `myfolder` in the current folder and `myfile.m` refers to `myfile.m` in the current folder.
- `../myfolder/myfile.m` refers to `myfile.m` in `myfolder`, where `myfolder` is at same level as the current folder. Each repetition of `../` moves up an additional folder level.

Partial Path Names in MATLAB

A partial path name is the last portion of a full path name for a location on the MATLAB search path.

Some functions accept partial path names. The reference page for a function typically specifies the valid types of path names.

Examples of partial path names are: `matfun/trace`, `private/cancel`, and `demos/clown.mat`.

Use a partial path name to:

- Specify a location more conveniently than by using the full path name.
- Specify a location independent of where MATLAB is installed.
- Locate a function in a specific toolbox when there is more than one function with that name in multiple toolboxes. For example, get help for the `set` function in the Database Toolbox™ product:

  ```
  help database/set
  ```

- Locate private and method files, which sometimes are hidden.

Specify enough of the path name to make the partial path name unique. Specifying the `@` in method folder names is optional.

Maximum Length of Path Names in MATLAB

The maximum length allowed for a path name depends on your platform.

For example, on Microsoft Windows platforms:

- The maximum length is known as `MAX_PATH`.
- You cannot use an absolute path name that exceeds 255 characters.
- For a relative path name, you may need to use fewer than 255 characters. When the Windows operating system processes a relative path name, it can produce a longer absolute path name, and therefore exceed the maximum.

If you get unexpected results when working with long path names, use absolute instead of relative path names. Alternatively, use shorter names for folders and files.

See Also

- "Slash and Backslash — / \"
- "Naming M-files"
- `ismac`, `ispc`, and `isunix` functions, for MATLAB statements that require different path names for different platforms

Using the Current Folder Browser to Manage Files

In this section...

What Is the Current Folder Browser?

The Current Folder browser:

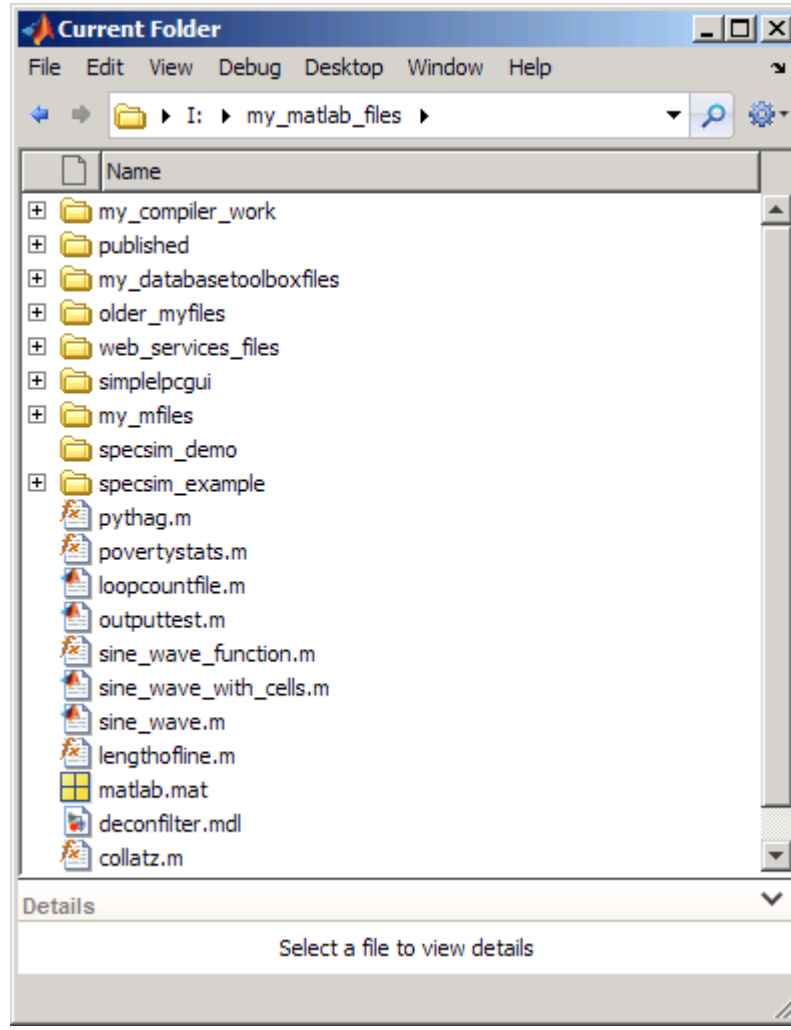
- Is the main tool for working with files in MATLAB
- Lets you access operating system file management features from within MATLAB
- Is like file browsers provided with operating systems, but also includes features unique to MATLAB. For example, you can add folders to the search path from the Current Folder browser.

Opening the Current Folder Browser

Open the Current Folder browser by selecting **Desktop > Current Folder** from the MATLAB desktop.

The Current Folder browser shows the full path to the current folder in the navigation bar, and shows the contents of the current folder.

Change the size, location, or other characteristics of the Current Folder browser as you would for any tool in the MATLAB desktop. See Chapter 2, "Desktop".



Preferences for the Current Folder Browser

Access the preferences by selecting **File > Preferences > Current Folder**. See:

- Refresh — How the Current Folder browser reflects changes made outside of MATLAB.

- History — The number of recently used current folders maintained in the Current Folder browser drop-down list.
- Hidden files (not on MicrosoftWindows platforms)

Viewing Files and Folders

In this section...

Viewing the Contents of a Folder in the Current Folder Browser

The Current Folder browser displays the contents of the current folder.

Double-clicking a subfolder displays the folder contents, and makes that folder become the MATLAB current folder.

View the contents of subfolders by clicking the + (expand) button. MATLAB might not be able to run files in the subfolders. See "Making Files and Folders Accessible to MATLAB" on page 6-38.

Viewing Hidden Files and Folders

The operating system, by default, hides certain files and folders. The Current Folder browser can display hidden files and folders. How to instruct the Current Folder browser to display hidden files depends on the operating system.

On Microsoft Windows platforms, the Current Folder browser follows the Windows preference for showing hidden files. To set or change the Windows preference:

1 Open Windows Explorer.

2 Select **Tools > Folder Options**.

3 Click the **View** tab.

4 Under **Advanced** settings, select **Show hidden files and folders**.

On other platforms, specify the behavior using Current Folder preferences:

1 Select **File > Preferences > Current Folder**.

2 Specify the setting for **Hidden files**.

Refreshing the View

When files and folders change outside of MATLAB, the Current Folder browser automatically reflects the changes. When you access files on a network, frequent refreshing of the Current Folder browser can slow performance in MATLAB. Try improving the performance by changing how frequently refreshing occurs using the Current Folder **Refresh** preference:

1 Select **File > Preferences > Current Folder**.

   By default, the **Auto-refresh view from file system** option is on, with an update time of 3 seconds. Every 3 seconds, the Current Folder browser checks for and reflects changes made from programs and tools other than MATLAB.

2 Try to alleviate slowness by either:

   - Increasing the **Number of seconds between auto-refresh**.
   - Clearing the **Auto-refresh view from file system** check box to turn off the feature.

3 Click **OK**.

To refresh the view at any time:

1 Right-click in the list area of the Current Folder browser.

2 Select **Refresh** from the context menu.

Viewing the Contents of a Folder Using Functions

Use the `dir` or `ls` functions.

Viewing Details About Files and Folders In the Current Folder Browser

In the Current Folder browser, details about files and folders are in columns, beneath file and folder names, and in the **Details** panel.

Customizing the Column View

To specify the columns that the Current Folder browser displays (size, date modified, type, and description):

1. Select **View > Show**.
2. Select the columns to show. Clear the columns to hide.

To show fewer columns:

- Show the **Type** column only if the icon column does not provide enough information.
- Sort or group by a column without showing the column.

To modify the columns:

- Change the order — Drag a column header to a new position.
- Change the width — Drag the edge of the column header.

Viewing Descriptions

Show or hide descriptions in the Current Folder browser by selecting **View > ShowDescription**.

Descriptions appear in gray text beneath the name of the file or folder. The Current Folder browser shows descriptions only for files and folders that are relevant to products from The MathWorks. How the Current Folder browser gets the description depends on the type of item:

- **M-files** — The description is the first line of the help comments, known as the H1 line.
- **Simulink Models** — The description is from the Description pane of the Model Properties dialog box. Use the Current Folder browser to view model descriptions without starting the Simulink software.
- **Folders** — The description is the first comment line of the `Contents.m` file for the folder.

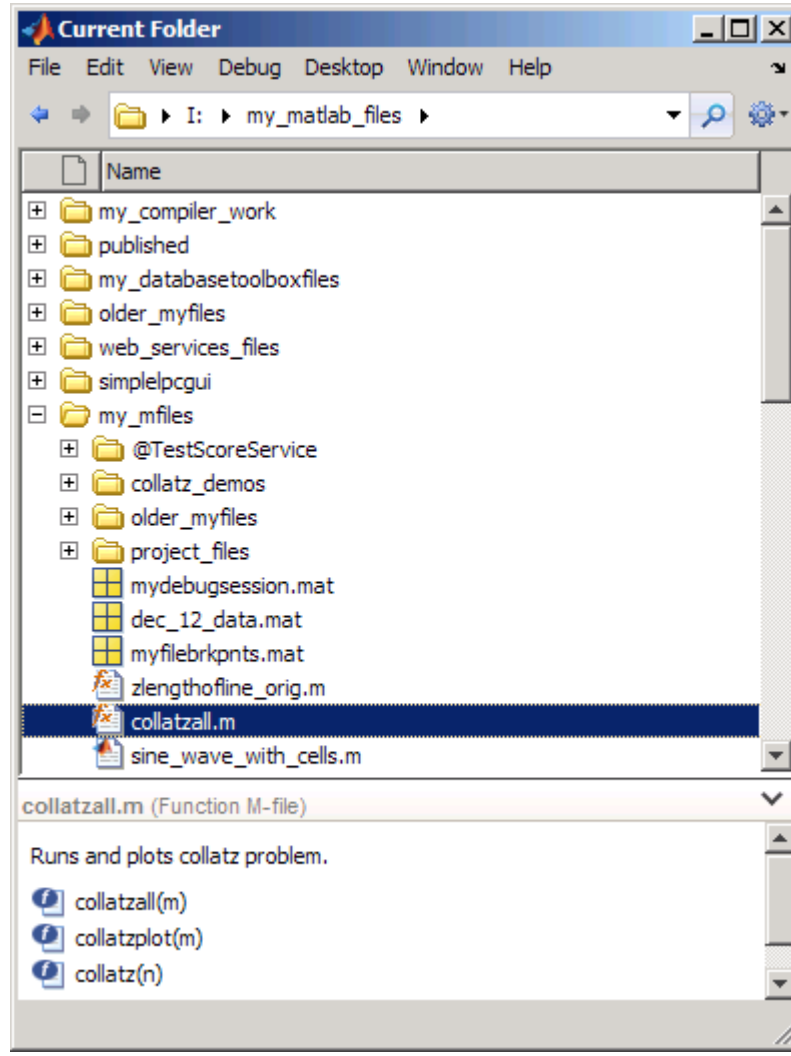
To provide descriptions for your own files and folders, see "Providing Your Own Help and Demos" on page 4-27.

Using the Details Panel

View more information about files and folders that are relevant to MathWorks products by selecting the file or folder. Information appears in the **Details** panel.

To customize the **Details** panel:

- Change the height by dragging the separator bar up or down.
- Collapse or expand by clicking the down arrow in the title bar of the panel. Or, press **Tab** until the pointer is at the **Details** panel, and press **Enter**



Viewing and Going To Elements within an M-File. The **Details** panel lists these elements in the selected M-file:

- Subfunctions
- Cells

- Properties 
- Methods 

To open the M-file in the Editor, scrolled to the start of the element, double-click the element in the **Details** panel.

Viewing and Loading MAT-File Variables. Use the **Details** panel to view the name, class, and value of all variables in the selected MAT-file. To load a variable into the workspace, select it in the **Details** panel and drag it to the Workspace browser.

Viewing Help for an M-File

From the current folder browser, you can view help for an M-file that is in the current folder or in a folder on the search path:

1. Right-click the M-file.
2. Select **View Help** from the context menu.

   The reference page, if it exists, opens in the Help browser. Otherwise M-file help, if it exists, displays in the Help browser.

Getting Details About Files and Folders Using Functions

To...	Use This Function
Get the name, date, and size for a file or folder	`dir`
Get and set the read-write status and other attributes for a file or folder	`fileattrib`
Separate a path name into folder and file name	`fileparts`
Build a file or folder name from parts	`fullfile`
Determine if a string corresponds to a folder	`isdir`

Sorting and Grouping Files and Folders in the Current Folder Browser

Organize, find, and manage the files and folders you use with MATLAB by sorting and grouping items.

By default, sorting is by **Name** and grouping is off.

Regardless of the sorting and grouping options selected, the Current Folder browser lists folders and files separately.

Sorting Items

To change the order of items listed, sort by column:

1. Select **View > Sort By**.
2. Select the name of the column to sort by.

Grouping Items

To see related items listed together, group them:

1. Select **View > Group By**.
2. Select **Type**, **Size**, or **Date Modified**.

Each group has a label. To hide the items in a group, click the collapse button (–) next to the label.

To turn off grouping, select **View > Group By > Stop Grouping**.

Using Sorting and Grouping Together

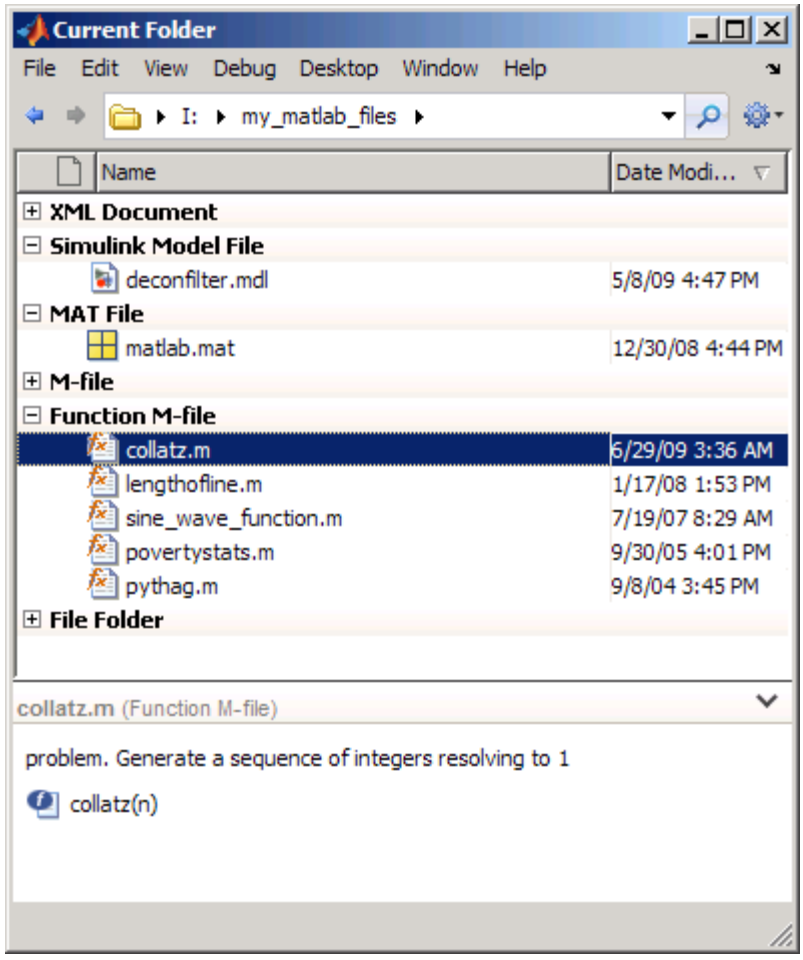
You can sort and then group, or group and then sort.

After grouping items, sort using different criteria. The sort applies to the groups and to items within each group.

Example of Sorting and Grouping

The following figure illustrates grouping by type, with some groups collapsed. The sort order is descending by date

- Because the most recently modified item was an **XML Document**, the group appears at the top.
- Within the **Function M-file** group, the most recently modified file, `collatz.m`, appears at the top.



Viewing Only One Type of File

To view *only* files of a certain type, for example, only M-files, use a simple search. See "Simple Search for File and Folder Names in the Current Folder Browser" on page 6-20.

Finding Files and Folders

In this section...

Finding Files and Folders by Name in the Current Folder

In the Current Folder browser, use the typeahead feature to find a file or folder by name in the current folder:

1. Position the pointer in the list of files and folders in the current folder.
2. Type the first characters of the name you want to find.

   As you type, the Current Folder browser searches downward from the top of the window, looking through all expanded folders. It selects the first entry in the current folder whose name begins with the characters you typed. It does not find

Typeahead and *find as you type* are other names for this feature.

Simple Search for File and Folder Names in the Current Folder Browser

Find names in the current folder and subfolders that contain a specified series of characters by using the search field in the Current Folder browser. *Instant search* and *filtering* are other names for this feature.

Steps for Using the Search Field

1 Change the current folder to the folder you want to look in.

See "Viewing and Changing the Current Folder Using the Current Folder Browser" on page 6-43.

2 Click the search button  in the address bar.

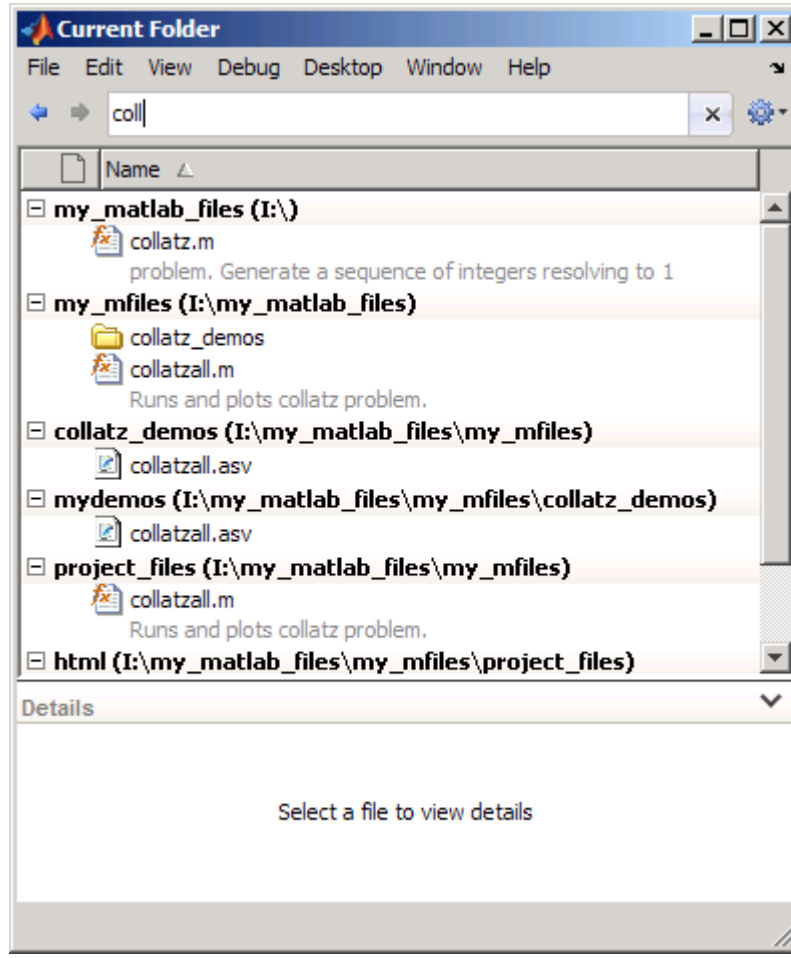
The path in the address bar becomes a field where you enter text, displaying the message `Type search text (ex:  *.m)`.

(If the address bar is not in the Current Folder browser toolbar, see "Using Toolbar Features" on page 2-112.)

3 In the search field, begin typing the string to you want to find. What you type replaces the message in the field. Ignore irrelevant characters in the string by using `*` (an asterisk) as the wildcard character.

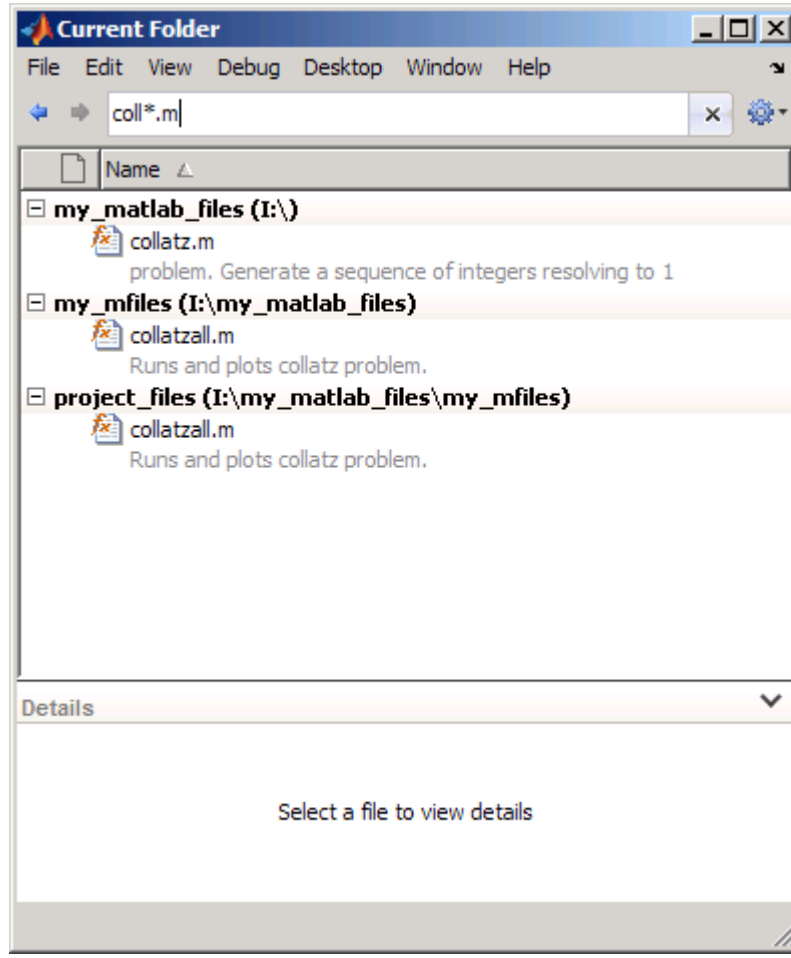
As you type, the Current Folder browser lists only the names of files and folders that include the string you typed.

The following is an example of the results when you search for `coll`. The example shows results arranged by location, with the full path to the location in parenthesis.



4 Further filter the list by typing additional characters or removing characters you already typed.

Continuing the example, append `*.m` to show only file names that being with `coll` and have a `.m` extension.

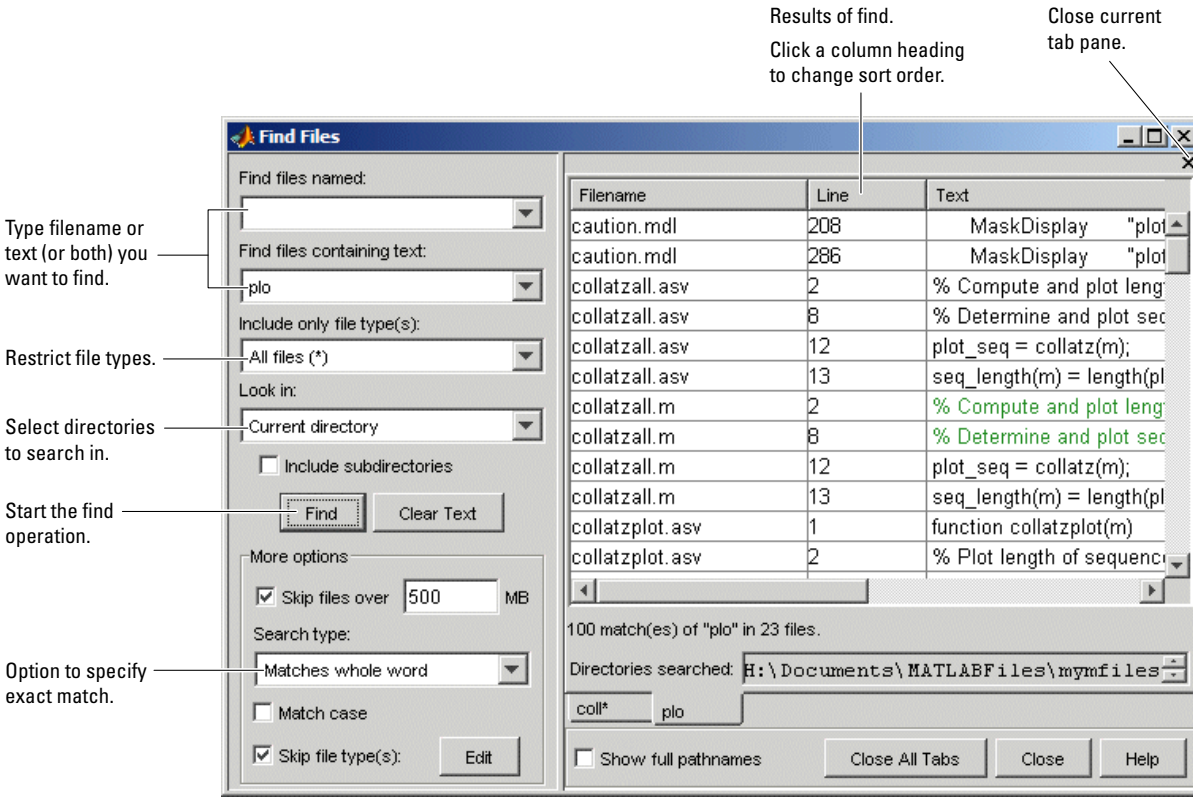


5 Customize the way search results appear by using the **View** menu options: **Show**, **Sort**, and **Group**. See "Viewing Files and Folders" on page 6-11.

6 Clear the filter results and show all items in the current folder by clicking the Close box in the filter field. Alternatively, press the **Esc** key.

Advanced Search for Files — Find Files Tool

To look for a specified string in file names and within files located in multiple folders, use the Find Files tool.



Steps for Using the Find Files Tool

1. Open the Find Files tool by selecting **Edit > Find Files** from any desktop tool.

2. Search for file names containing a specified string by typing the string in the **Find files named** field. Ignore irrelevant characters in the string by using `*` (an asterisk) as the wildcard character. For example, type `coll*` to search for file names that start with `coll`.

   Reuse a search string you previously entered in this MATLAB session by clicking the down arrow in the search field and selecting it from the list.

3. Search for a specified string in the content of files by typing the string in the **Find files containing text** field. For example, search for `plot`. Alternatively, select text in the Command Window or Editor and that text appears in the field.

   - For partial word searching in file contents, under the **More options Search type**, select `Contains text`.
   - Find an exact full-string match by selecting `Matches whole word`.

4. Specify file types to search for by selecting one of the options listed in the table.

One type	For **Include only file type(s)**, select the file type you are looking for. For example, select `*.m` to limit the search to M-files.
All types	**a** For **Include only file type(s)**, select **All files (*)**. **b** Clear the **Skip file type(s)** check box, under **More options**.
Other variations	**a** For **Include only file type(s)**, select **All files (*)**. **b** Select the **Skip file type(s)** check box, under **More options**.

	c Select **Edit** to specify the file types. See "Skipping File Types" on page 6-27.

5 Specify the folders to search using one of the **Look in** options:

- Select an option listed.
- Enter the full path for one or more folders. Separate each path by a semicolon (`;`).
- Include subfolders by selecting the **Include subdirectories** check box.

6 Further restrict the search using **More options**. For example, use the **Skip files over** option. It ignores large files that could take a long time to look through.

7 Perform the search by clicking **Find**.

The Find Files tool presents the search results in the right pane of the dialog box, with a summary at the bottom. For text searches, results include the line number and line of code.

8 Customize the display of results:

- To see file locations, select **Show full pathnames**.
- To sort results by a column, click the column heading. For example, click **Line** to sort results by line number.

Opening Files from the Results List

1 Select the files you want to open.

2 Right-click and select one of the **Open** options from the context menu.

Accessing Previous Results

View the results of a previous search by selecting its tab at the bottom of the results pane. Find Files shows up to 10 tabs for previous search results

while the tool is open. File Files does not maintain the results after you close the tool.

Skipping File Types

Use the Find Files tool to look in all file types *except* file types you specify:

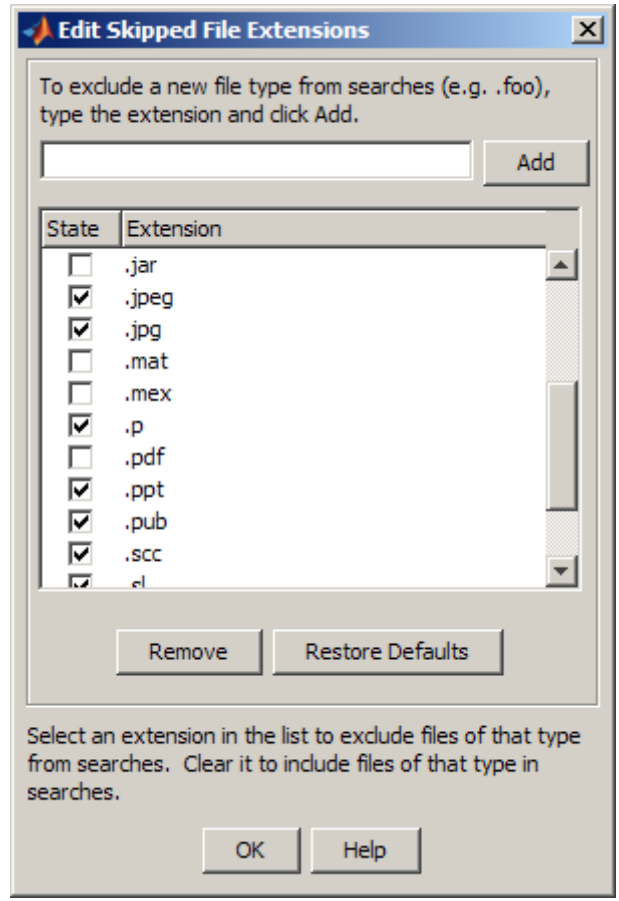
1. For **Include only file type(s)**, select **All files (*)**.
2. Specify the file types you want the search to ignore:
   a. Select the **Skip file type(s)** check box.
   b. Click **Edit**.
3. In the resulting Edit Skipped File Extension dialog box, specify which file types to look in and which to ignore:
   - Ignore a file type by selecting its **State** check box.
   - Look for a file type by clearing its **State** check box.
4. Add any file types not listed, if you want to skip them or look for them:
   a. Enter the file extension in the field at the top of the dialog box.
   b. Click **Add**.

      The file type appears in the list.
   c. Verify that the **State** check box has the setting you want.

   The example at the end of this procedure shows the `scc` file type added.
5. Reduce the size of the list by removing any file extensions not relevant to your search:
   a. Select the name of the extension.
   b. Click **Remove**.
6. Click **OK** to accept your changes.

   The **Edit Skipped File Extensions** dialog box closes.

After making the changes, when you use the Find Files tool, search ignores the selected file types.



Locating a File or Folder in the Operating System Browser

From the Current Folder browser, you can go to a file or folder location in the Windows Explorer or the Apple Macintosh Finder:

1 In the Current Folder browser, right-click the file or folder.

2 From the context menu, select **Locate on Disk**.

The Windows Explorer or Macintosh Finder opens to the folder containing the selected item.

Finding Files and Folders Using Functions

To...	Use This Function
Determine if a variable, function, or folder exists.	`exist`
Search for the specified string in the first line of M-file help	`lookfor`
See files and folders that are relevant to MATLAB	`what`
See the full path to a file	`which`

Additional Ways to Find Files

- "Viewing Files and Folders" on page 6-11
- "Finding Functions Using the Function Browser" on page 3-37
- "Getting Better Search Results" on page 4-6
- "Finding Files in File Exchange — Searching and Using Tags" on page 7-13

Creating, Changing, and Deleting Files and Folders

In this section...

Creating Folders Using the Current Folder Browser

1. Right-click at the location for the new folder. See "Locations for Storing Your Files" on page 6-4.
2. From the context menu, select **New Folder**.
3. Type the name for the new folder.
4. Press **Enter**.

Creating Files Using the Current Folder Browser

Creating M-Files and Model Files

1. Right-click at the location for the new file. See "Locations for Storing Your Files" on page 6-4.
2. From the context menu, select **New File**.
3. Select the type of file you want to create: **Blank**, **Function**, **Class**, or **Model**.

The function or class M-file contains template information that prompts you to provide fundamental elements for the file.

4 Type the name for the new file.

For naming conventions, see "Function Name" and "Naming M-files".

5 Press **Enter**.

Creating and Updating MAT-Files

To create or update a MAT-file using variables in the workspace:

1 In the Current Folder browser, change the current folder to the folder where you want to save the variables. See "Locations for Storing Your Files" on page 6-4.

2 In the Workspace browser, select the variables to save.

3 Drag the selected variables from the Workspace browser to the Current Folder browser.

4 Drop the variables in the Current Folder browser:

- Create a MAT-file by dropping the variables onto any empty location in the Current Folder browser. Then name the file.
- Update an existing MAT-file by dropping the variables onto the file name.

  MATLAB warns you when the MAT-file contains variables of the same name. To update the existing variables, click **Continue**. Otherwise, click **Cancel**.

  To suppress the warning, select **File > Preferences > General > Confirmation Dialogs**, and clear the preference, **Confirm when overwriting variables in MAT-files**.

See also "Opening Files and Importing Data Using the Current Folder Browser" on page 6-36.

Creating Files and Folders Using Functions

To...	Use This Function
Create a folder	`mkdir`
Create an M-file	`edit`
Create a MAT-file	`save`

See also "Locations for Storing Your Files" on page 6-4.

Renaming Files and Folders Using the Current Folder Browser

1. Select the item to rename.
2. Right-click and select **Rename** from the context menu.
3. Type over the existing name with the new name. Warnings appear when:
   - The new name is invalid. Change the name to make it valid. See "Naming M-files".
   - The folder is on the search path. See "Making Changes to Folders on the Search Path" on page 6-55.
4. Press **Enter**.

Renaming Files and Folders Using Functions

Use `movefile`.

Deleting Files and Folders Using the Current Folder Browser

To remove items:

1. Select the items to delete.

   You cannot delete a folder while it is on the search path. See "Making Changes to Folders on the Search Path" on page 6-55.

2 Right-click and select **Delete** from the context menu.

When you delete a file or folder using the Current Folder browser, MATLAB permanently removes it or moves it to another location, based on your platform.

Platform	Behavior Deleting Files and Folders Using the Current Folder Browser
Windows platforms	Follows the Windows system preference for sending files to the Recycle Bin. Some systems only allow recycling of local files and not files accessed on a network. To delete a selection permanently when the system preference is set to recycle, press **Shift+Delete**.
Linux platforms	Specify the behavior: 1 Select **File > Preferences > General**. 2 Set the **Deleting files** option you want. If you select to move files to a temporary folder, determine the location by running `tempdir`. To delete a selection permanently when the preference is set to recycle, press **Shift+Delete**.
Macintosh platforms	Follows your Macintosh system preference for sending files to the Trash.

Deleting Files and Folders Using Functions

To...	Use This Function
Delete a file	`delete`
Delete a folder	`rmdir`

There is no opportunity to recover folders deleted using `rmdir`.

By default, the `delete` function permanently deletes files. To move them to a different location instead, use the **Deleting files** preference:

1. From any desktop tool, select **File > Preferences > General**.
2. Set the **Deleting files** option you want.

With the preference set to delete files permanently, `delete` runs faster.

To override the preference when using the `delete` function, use the `recycle` function.

The location for deleted files varies by platform.

Platform	Location for Files Not Permanently Deleted Using the delete Function
Windows platforms	Recycle Bin. Some systems only allow recycling of local files and not files accessed on a network.
Linux platforms	`MATLAB_Files_<day>-<mo>-<yr>_<hr>_<min>_<sec>` folder in the location returned by the `tempdir` function. For example, when `tempdir` returns `/tmp`, files deleted at 2:09:28 in the afternoon of November 9, 2009 move to `/tmp/MATLAB_Files_09-Nov-2009_14_09_28`.
Macintosh platforms	Trash

Deleted files remain in these locations until you remove them. To remove deleted files, use operating system features, such as **Empty Recycle Bin** on Windows platforms.

Copying and Moving Files and Folders

Copy and move files and folders using the Current Folder browser using standard GUI practices. For example, drag a file from one folder to another or to another application, such as Windows Explorer.

Using the Current Folder browser, you cannot move a folder that is on the search path. See “Making Changes to Folders on the Search Path” on page 6-55

To copy and move files and folders using functions, use `copyfile` and `movefile`.

Opening and Running Files Using the Current Folder Browser

In this section...

Opening Files and Importing Data Using the Current Folder Browser

1 In the Current Folder browser, right-click the file you want to open or load.

2 From the context menu, select an option for opening or importing the file:

- **Open** — Uses the appropriate MATLAB tool for the file type. For example, loads a MAT-file into the Workspace browser.
- **Open in GUIDE** — Opens a FIG-file in GUIDE instead of a figure window.
- **Open as Text** — Uses the Editor.
- **Open Outside MATLAB** — Uses the application or tool that the operating system associates with the file type.

  For example, `.mat` is the extension for MATLAB data files and Microsoft Access files. Whereas **Open** loads the file into the MATLAB workspace, **Open Outside MATLAB** opens the file into Microsoft Access. See "Changing File Associations for the MATLAB Program from the Windows Environment" on page 1-6.
- **Load** — Uses the Import Wizard. For more information, click **Help** in the wizard.

See Also

- "Using the Details Panel" on page 6-14.
- "Opening Files from the Results List" on page 6-26

Running M-File Scripts from the Current Folder Browser

For convenience, you can run M-file scripts (M-files that do not require input arguments) from the Current Folder browser:

1. In the Current Folder browser, change the current folder to the folder containing the file to run.
2. Right-click the file.
3. From the context menu, select **Run File**.

Making Files and Folders Accessible to MATLAB

In this section...

Files and Folders MATLAB Can Access

For performance reasons, MATLAB limits where it looks for files. To run or get help for an M-file, or to load a MAT-file, the file must be in either:

- The current folder in MATLAB
- A folder that is on the search path. See "What Is on the Search Path?" on page 6-46

Also make accessible:

- Folders containing files that you and others create.
- Folders containing files that *called by* files you run.
- Subfolders containing files you run. Making a folder accessible does not make its subfolders accessible

For files in `private`, `@` (class), and `+` (package) folders, instead make the parent folder accessible. When the current folder is a `private` folder, subfolders and files in `private` are also accessible. See "Organizing Classes in Directories".

How to Make Files Accessible

For files that you and other users create, see "Basic Options for Making Files Accessible" on page 6-39.

To understand the differences in the basic options, and for other approaches, see "All Options for Making Files Accessible" on page 6-39.

Basic Options for Making Files Accessible

- Store the files you and other users create in the `MATLAB` folder, which is on the search path. See "Locations for Storing Your Files" on page 6-4.
- Change the current folder to the folder that contains the files.
- Add the folders that contain the files to the search path.

All Options for Making Files Accessible

Usage	Recommendation
You *seldom* run the file	Change the current folder to the folder that contains the file. See "Determining and Changing the Current Folder" on page 6-43.
The file is an M-file *script* (takes no input or output arguments)	Use the `run` function.
Files are in *one* folder	Put the files in the *`userpath`* folder. See "Locations for Storing Your Files" on page 6-4
Files are in *multiple* folders	Add the folders to the search path. See "Adding Folders to the Search Path" on page 6-49. If you regularly use the files, save the changes—see "Saving Changes to the Search Path" on page 6-52.
Files call other files that are in multiple folders	**1** Determine the location of all the called files. See "Displaying Dependencies Among M-Files" on page 9-16. **2** Add the folders to the search path. See "Adding Folders to the Search Path" on page 6-49.
Some files in multiple folders have the same name	See "Detecting and Addressing Name Conflicts" on page 6-42.

Usage	Recommendation
Use files in different versions of MATLAB or on different platforms	Modify the search path in a `startup.m` file. See "Using the Search Path with Different MATLAB Installations" on page 6-53.
Work with the search path content programmatically	See functions in the Search Path category.

Determining If MATLAB Can Access a File

The following table lists ways to determine if MATLAB has access to a file.

Option	When MATLAB Can Access the File	When MATLAB Cannot Access the File
Use the file.	Works successfully.	Produces an error. Typical error notifications include: • Dialog box • Message: `??? Undefined function or method 'file'Name'` message • Message: `Cannot find function 'fileName'`
View file in the Current Folder browser.	File is in the current folder.	File is in a subfolder of the current folder, unless the subfolder is on the search path.
Select **File > Set Path**.	Set Path dialog box list includes the file location.	List in the Set Path dialog box does not include the file location.
Run `dir` with no arguments.	Result includes the file, indicating file is in current folder.	Result does not include the file.

Option	When MATLAB Can Access the File	When MATLAB Cannot Access the File
Run `path`.	Result includes the file location, indicating file is in a folder on search path.	Result does not include the file location.
Run `which` *`fileName`*.	Result is the full path to file.	Result is an error or a file with the same name in another location.

Ensuring MATLAB Uses the File You Want

About Name Conflicts and Shadowed Files

When MATLAB has access to multiple files with the same name, these precedence rules determine the file MATLAB uses:

- MATLAB uses the file in the current folder instead of a file on the search path.
- MATLAB uses the file closest to the top of the search path instead of a file further down.

The file MATLAB does *not* use is called a *shadowed* file. In some cases, MATLAB warns you that a shadowed file exists.

To resolve other name conflicts, for example, when:

- A file has the same name as a variable in the base workspace
- A file has the same name as a built-in function for a MathWorks product

MATLAB follows these precedence rules:

- "Precedence Rules" and "File Precedence" in the MATLAB Programming Tips documentation.
- "Class Precedence and MATLAB Path"

Detecting and Addressing Name Conflicts

MATLAB might not be accessing the file you want it to when:

- You use a file and get a warning about a potential name conflict.
- You get unexpected results.

To identify a name conflict, try using the `which` function.

To address a name conflict, try one of the following:

- Change the current folder.
- Move or remove folders on the search path.
- Rename or move files.
- Specify the full path or partial path to the file you want.
- Maintain a single version of a file instead of multiple versions.

Name conflicts can arise from using files that you create. They also can arise from using:

- Files that other users create, such as from File Exchange
- A different system that has additional MathWorks products installed
- A different version of MATLAB, which could include new functions that have the same names as your existing files

See Also

- "Built-In Functions" and "Overloaded MATLAB Functions"
- `rehash` and "Toolbox Path Caching in the MATLAB Program" on page 1-22

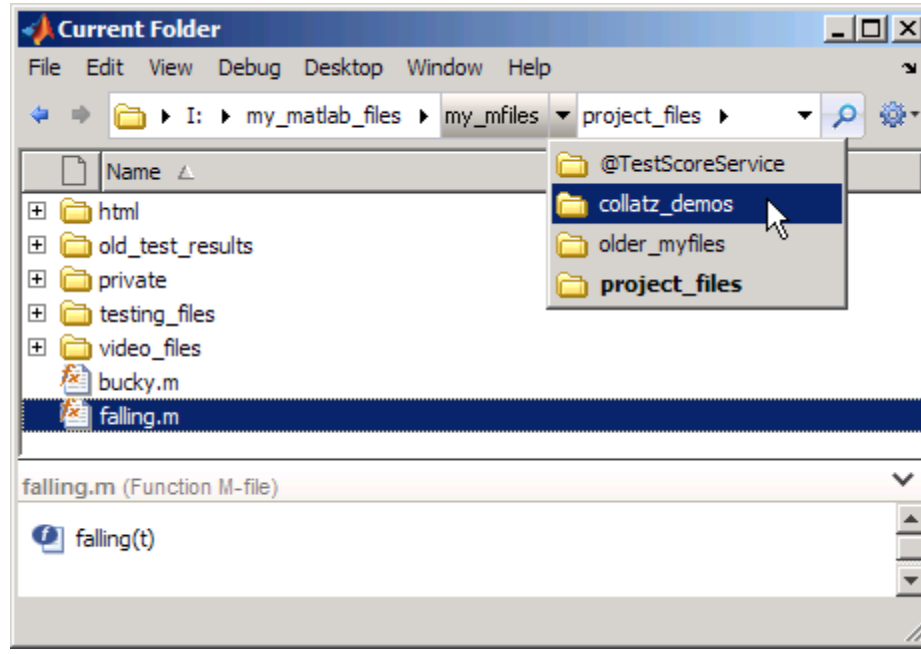
Determining and Changing the Current Folder

In this section...

Viewing and Changing the Current Folder Using the Current Folder Browser

To view and change the current folder:

- View the full path to the current folder in the address bar.
  - If the full path does not fit in the address bar, hover over the folder icon to see it.
  - To view the full path as a string, suitable for copying or pasting, click the empty area near the right edge of the address bar.
- View subfolders and change the current folder using the address bar. For more information, right-click the address bar, and from the context menu, select **Help Using Address Bar**.
- Make a subfolder become the current folder by right-clicking the subfolder and selecting **Open** from the context menu.



Viewing and Changing the Current Folder from the Desktop Toolbar

To view or change the current folder in the desktop toolbar, use the current folder field. For example, select a previously used current folder from the history.

Determining and Changing the Current Folder Using Functions

Use the `cd` function.

Specifying the Current Folder at Startup

To specify the current folder programmatically when MATLAB starts, see "Startup Folder for the MATLAB Program" on page 1-11.

See Also

- "The Current Folder in MATLAB" on page 6-3
- "Making Files and Folders Accessible to MATLAB" on page 6-38

Using the Search Path

In this section...

What Is the Search Path?

The search path is:

- A subset of all the folders in the file system.
- One way that MATLAB efficiently locates files used with MathWorks products.
- Also referred to as the *path*.

MATLAB can access all files in the folders on the search path.

What Is on the Search Path?

- By default, folders provided with MATLAB and other MathWorks products. Relevant folders under *`matlabroot`*`/toolbox` are:
- By default, `MATLAB`, the *`userpath`* folder. See "Locations for Storing Your Files" on page 6-4.
- Folders you add to the search path, for the files you and other users create.

Adding folders to the search path is like performing an include or import operation in other applications.

Class, package, and `private` folders are *not* on the search path. See "Files and Folders MATLAB Can Access" on page 6-38.

Order of Folders on the Search Path

The *order* of folders on the search path is relevant when two files with the same name are in folders on the search path. MATLAB uses the file nearer to the top of the search path. See "Ensuring MATLAB Uses the File You Want" on page 6-41.

How MATLAB Stores the Search Path

When you change the search path, MATLAB uses it in the current session. To use it in future sessions, save the changes.

MATLAB saves the search path information in the `pathdef.m` file. The `pathdef.m` file is a series of full path names, one for each folder on the search path, separated by a semicolon (`;`).

By default, `pathdef.m` is in *`matlabroot`*`/toolbox/local`.

Ways to View and Change the Search Path

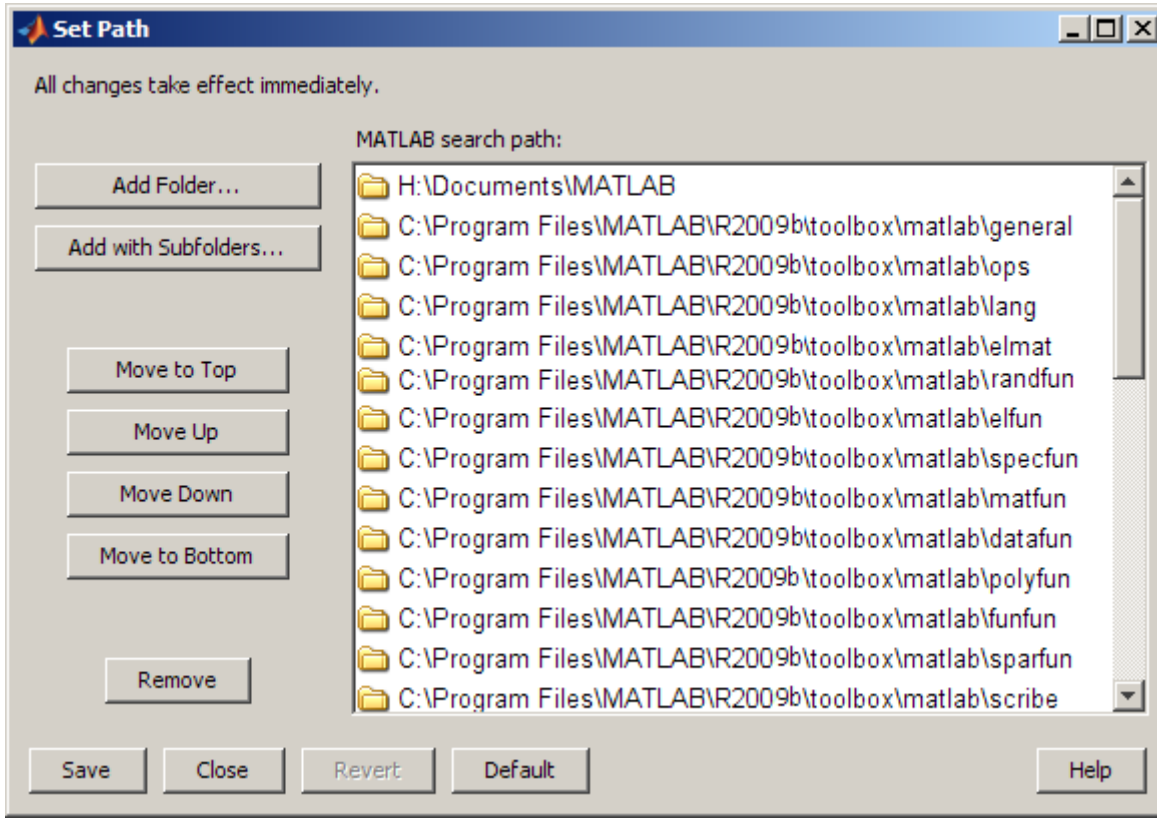
- Use the `path` and other functions in the Search Path category.
- "Using the Current Folder Browser to Manage Files" on page 6-8.
- "Using the Set Path Dialog Box" on page 6-47
- "Specifying Startup Options Using the Startup File for the MATLAB Program, startup.m" on page 1-19

Using the Set Path Dialog Box

To view and change the MATLAB search path using the Set Path dialog box:

1. Open the dialog box by selecting **File > Set Path**.

   The Set Path dialog box opens, listing all folders on the search path.



2 Add, remove, and change the order of folders on the search path as described in the following topics:

- "Adding Folders to the Search Path Using the Set Path Dialog Box" on page 6-49
- "Removing Folders from the Search Path Using the Set Path Dialog Box" on page 6-50
- "Changing the Order of Folders on the Search Path" on page 6-51

3 Commit or cancel changes:

- Keep the changes for use in the current session by clicking **Close**.

- Keep changes for use in the current and future sessions by clicking **Save**. See "Saving Changes to the Search Path" on page 6-52.
- Undo changes by clicking **Revert**.
- Restore the default search path by clicking **Default**. See "Restoring the Default Search Path" on page 6-52.

See also "Using the Search Path" on page 6-46.

Adding Folders to the Search Path

To add folders to the search path, you can use the Current Folder browser, the Set Path dialog box, or functions.

Adding Folders to the Search Path from the Current Folder Browser

1. Right-click the folders you want to add.
2. From the context menu, select **Add to Path**, and select the option you want:
   - **Containing Folder** (parent folder of the selected files or folders)
   - **Selected Folders**
   - **Selected Folders and Subfolders**

MATLAB adds the specified folders to the top of the search path.

Adding Folders to the Search Path Using the Set Path Dialog Box

1. Open the dialog box by selecting **File > Set Path**.
2. Select an option:
   - **Add Folder**
   - **Add with Subfolders**
3. In the resulting Browse for Folder dialog box:

a Select the folder you want to add to the search path.

b Click **OK**.

MATLAB adds the specified folder to the top of the search path.

4 Apply the changes:

- To use the newly modified search path only in the current session, click **Close**.
- To reuse the newly modified search path in the current session and future sessions, click **Save**.

5 Click **Close**.

Adding Folders to the Search Path Using Functions

Use the `addpath` function.

Removing Folders from the Search Path

To remove folders from the search path, you can use the Set Path dialog box or functions.

Removing Folders from the Search Path Using the Set Path Dialog Box

1 Select **File > Set Path**.

2 In the resulting dialog box, select the folders you want to remove from the search path.

3 Click **Remove**.

4 Apply the changes:

- To use the newly modified search path only in the current session, click **Close**.
- To reuse the newly modified search path in the current session and future sessions, click **Save**.

5 Click **Close**.

Removing Folders from the Search Path Using Functions

Use the `rmpath` function.

Changing the Order of Folders on the Search Path

To change the order of folders on the search path, you can use the Set Path dialog box.

To move folders to the top or bottom of the search path, you can use functions.

Changing the Order of Folders on the Search Path Using the Set Path Dialog Box

1 Select **File > Set Path**.

2 In the resulting dialog box, select the folders you want to move.

3 Click one of the **Move** buttons, such as **Move to Top**. The order of the folders changes.

4 To use the newly modified search path in future sessions, click **Save**.

   If you do not save the changes, the newly modified search path remains in effect until you end the current session of MATLAB.

5 Click **Close**.

Note The `MATLAB` (*`userpath`*) folder automatically moves to the top of search path the next time you start MATLAB. See "Locations for Storing Your Files" on page 6-4.

Moving a Folder to the Top or Bottom of the Search Path Using Functions

Use the `path` function.

Saving Changes to the Search Path

Changes you make to the search path remain in effect during the current session of MATLAB. For MATLAB to use the changed search path in subsequent sessions, save the search path, which updates the `pathdef.m` file.

Ways to Save Changes to the Search Path

- Click **Save** in the Set Path dialog box. See "Using the Set Path Dialog Box" on page 6-47.
- Use `savepath`.

Where to Save the Search Path File

Save the search path to the *default* location, *`matlabroot`*`/toolbox/local`, so MATLAB can locate it.

If you do not have write access to the default location, MATLAB prompts you for a different location. Choose the MATLAB startup folder.

Restoring the Default Search Path

The default search path contains only folders provided by The MathWorks.

Ways to restore the default search path:

- Click **Default** in the Set Path dialog box. See "Using the Set Path Dialog Box" on page 6-47. This method also adds the *`userpath`* folder—see "Locations for Storing Your Files" on page 6-4.
- Use the `restoredefaultpath` function.

See also "Recovering from Problems with the Search Path" on page 6-53.

Using the Search Path with Different MATLAB Installations

Using the Search Path with Different Versions

The default search path changes for each release of MATLAB because the default folders that come with the products change. Different versions of MATLAB cannot use the same `pathdef.m` file.

To use your files with a new version of MATLAB or with multiple versions, do one of the following:

- For each version, add the folders containing your files to the search path. Save the search path where that version of MATLAB can access it.
- Instead of changing the `pathdef.m` file, include `addpath` statements in the `startup.m` file. Use the same `startup.m` file with the multiple versions of MATLAB.

Using the Search Path with Different Platforms

To use your files with MATLAB on different platforms, include `addpath` statements in the `startup.m` file. See "Specifying Startup Options Using the Startup File for the MATLAB Program, startup.m" on page 1-19

Recovering from Problems with the Search Path

When there is a problem with the search path, you cannot use MATLAB successfully.

You could experience search path problems when:

- You save the search path on a Windows platform, and then try to use the same `pathdef.m` file on a Linux platform.
- The `pathdef.m` file becomes corrupt, invalid, renamed, or deleted.
- MATLAB cannot locate the `pathdef.m` file.

For example, when you start MATLAB, if a message like the following appears, it indicates a problem with the search path:

```
Warning: MATLAB did not appear to successfully set the search
path...
```

To recover from problems with the search path, try the following steps, in order, proceeding to the next step only as necessary:

1. Ensure MATLAB is using the `pathdef.m` file you expect:
   a. Run
      ```
      which pathdef
      ```
   b. If you want MATLAB to use the `pathdef.m` file at a different location, make corrections. For example, delete the incorrect `pathdef.m` and ensure the correct `pathdef.m` is in a location that MATLAB can access. See "Where to Save the Search Path File" on page 6-52.
2. Look for and correct problems with the `pathdef.m` and `startup.m` files:
   a. Open `pathdef.m` and `startup.m` in a text editor. Depending on the problem, you might not be able to open the `pathdef.m` file.
   b. Look for obvious problems, such as invalid characters or path names.
   c. Make corrections and save the files.
   d. Start MATLAB again to ensure that the problem does not recur.
3. Try to correct the problem using the Set Path dialog box:
   a. Restore the default search path and save it. See "Using the Set Path Dialog Box" on page 6-47. Depending on the problem, you might not be able to open the dialog box.
   b. Start MATLAB again to ensure that the problem does not recur.
4. Restore the default search path using functions:
   a. Run `restoredefaultpath`, which sets the search path to the default and stores it in *`matlabroot`*`/toolbox/local`.
   b. If `restoredefaultpath` seems to correct the problem, run `savepath`.
   c. Start MATLAB again to ensure that the problem does not recur.

   Depending on the problem, a message such as the following could appear:

```
The path may be bad. Please save your work (if desired), and quit.
```

5. Correct the search path problems encountered during startup:

   a. Run

      ```
      restoredefaultpath; matlabrc
      ```

      Wait a few minutes until it completes.

   b. If there is a `pathdef.m` in the startup folder, it caused the problem. Either remove the bad `pathdef.m` or replace it with a good `pathdef.m` file. For example, run:

      ```
      savepath('path_to_your_startup_folder/pathdef.m')
      ```

      See "Startup Folder for the MATLAB Program" on page 1-11.

   c. Start MATLAB again to ensure that the problem does not recur.

After correcting problems with the search path, make any changes to run your files. For example, add the *`userpath`* folder or other folders to the search path.

Making Changes to Folders on the Search Path

You could encounter errors or unexpected behavior when you try to delete, rename, or move folders that:

- Are on the search path
- Contain subfolders that are on the search path

The behavior varies by platform because it depends on the behavior of similar features in the operating system.

If your task fails and the error message indicates it is because the folder is on the search path, do one of the following:

1. Remove the folder from the search path.
2. Delete, rename, or move the folder.
3. Add the folder to the search path.

Related Topics for Managing Files

- “Comparing Files and Folders” on page 8-81
- Chapter 9, “Tuning and Managing M-Files”
- Chapter 12, “Source Control Interface”
- Chapter 7, “File Exchange — Finding and Getting Files Created by Other Users”

7

File Exchange — Finding and Getting Files Created by Other Users

Before Using File Exchange

In this section...

What Is File Exchange?

File Exchange lets you use files that were created by other users. Users have submitted thousands of files to a repository located at the MathWorks Web site, that include:

- M-files
- Simulink models
- Video demos

Use File Exchange to find a file you want and download it for use in MATLAB. Using the files saves you time, provides new ideas for your own work, and extends the set of features provided with MathWorks products.

What You Need to Use File Exchange

To access the repository, you need:

- An Internet connection

  If your network uses a proxy server to access the Internet, specify the proxy server settings. For more information, see "Specifying Proxy Server Settings" on page 2-106.

- A MathWorks Account. If you do not have an account, create one when you open the File Exchange desktop tool. Use the **Create an account** link in the login window.

There is no cost to create an account or to use files in the File Exchange repository.

Ways to Access the File Exchange Repository

There are two ways to access the repository:

- File Exchange tool in the MATLAB desktop. See "How To Use the File Exchange Desktop Tool" on page 7-5.
- Web interface. Select **Help > Web Resources > MATLAB File Exchange**, or go to `http://www.mathworks.com/matlabcentral/fileexchange/`

Both the desktop tool and Web interface provide similar functionality. Use either to find, view details for, download, and provide feedback about files in the repository. There are some differences.

When to Use the Desktop Tool

Use the desktop tool when you want to:

- Work within the MATLAB desktop, as a natural part of your workflow
- Use multiple tags to find files
- Use search words and tags together to find files

When you use the File Exchange desktop tool, you can access only those files that are licensed under the BSD license. As a result:

- The desktop tool could report fewer matches than the Web interface reports.
- You might not find a file using the desktop tool that you can find using the Web interface.

When to Use the Web Interface

Use the Web interface when you want to:

- Submit files to the repository.
- Use File Exchange in a Web browser, without starting MATLAB.
- View a list of all authors, monitor changes to files with a watch list, and use other related features.
- View the complete list of files that match your criteria.

For performance reasons, the desktop tool does not list all matches at once. The desktop tool lists a maximum of 50 matches at once. You can view the other files in the desktop tool by changing your criteria.

How To Use the File Exchange Desktop Tool

In this section...

Steps for Using File Exchange

1 Open the tool by selecting **Desktop > File Exchange**.

2 In the login window, provide the e-mail address and password for your MathWorks Account.

For more information, see "What You Need to Use File Exchange" on page 7-2.

After logging in, File Exchange displays:

- The most popular tags for all files in the repository. *Tags* are keywords that users associate with files. The more frequently users apply a tag, the more popular it is.
- The 50 most recently submitted files

3 Find files. In general, the most efficient way to begin is by entering search words.

For more information, see "Finding Files in File Exchange — Searching and Using Tags" on page 7-13.

4 Refine results by selecting relevant tags, which you can see in cloud or list view. In cloud view, the font size of the tag indicates its popularity.

For more information, see "Using Tags to Find Files in File Exchange" on page 7-14.

5 Look for files you want to use. If you do not see any files you want, see other results by changing the sort order, the search words, or the selected tags. File Exchange lists up to 50 different files.

For more information, see "Viewing and Sorting the List of Files in File Exchange" on page 7-28.

6 View more details about a file by clicking the file name in the list of files. The file details page opens.

For more information, see "Viewing Details About a File" on page 7-30.

7 Get a file you want to use by clicking the **Download** button at the top of the file details page.

For more information, see "Downloading Files from the File Exchange Repository" on page 7-32

8 Use a downloaded file with MATLAB software.

If you have problems with the file, see "Best Practices for Using Files Provided by Other Users" on page 7-37.

9 Provide your rating and comments, and add tags to the file using the **Submit** area at the bottom of the file details page.

For more information, see "Contributing to the File Exchange Repository" on page 7-39.

10 When you finish using the tool, close it or log out.

If you have questions while you work, see the Frequently Asked Questions (FAQ) by clicking the Help button  .

Example — Finding and Downloading a File in File Exchange

This example looks for files to help you analyze voice signals.

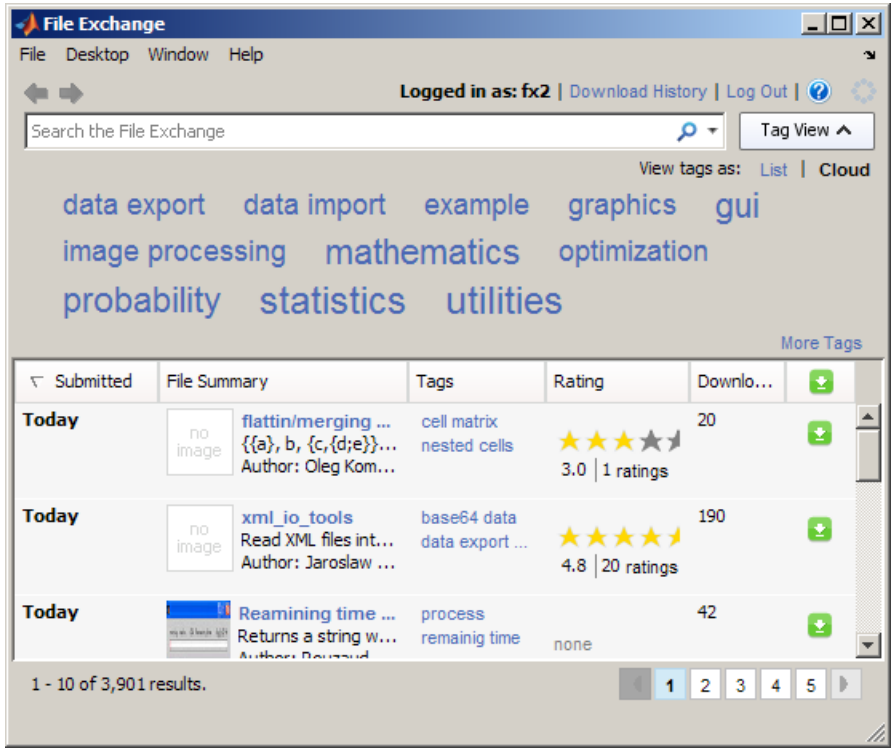
Note The File Exchange repository changes daily. Your results could differ from this example.

---

1 Select **Desktop > File Exchange**, and log in to your MathWorks Account.

File Exchange:

- Shows the most popular tags for *all* files in the repository. For the example, the most popular tags are `data export`, `data import`, etc.
- Lists the *50 most recently submitted* files, with 10 files per page. For the example, the most recently submitted file is `flattin/merging nested cells`.
- Reports the total number of files in the repository accessible to File Exchange in the desktop (have a BSD license). For the example, there are 3,901 files.



2 Type `signal` in the search field.

File Exchange:

- Reports that 277 files contain `signal` or variations of it in the title, tag, or description.
- Lists 50 of the 277 matches, with the most recently submitted, `Find and Replace`, at the top of page 1.
- Shows the most popular tags associated with the 50 files listed: `aerospace`, `automotive`, etc.

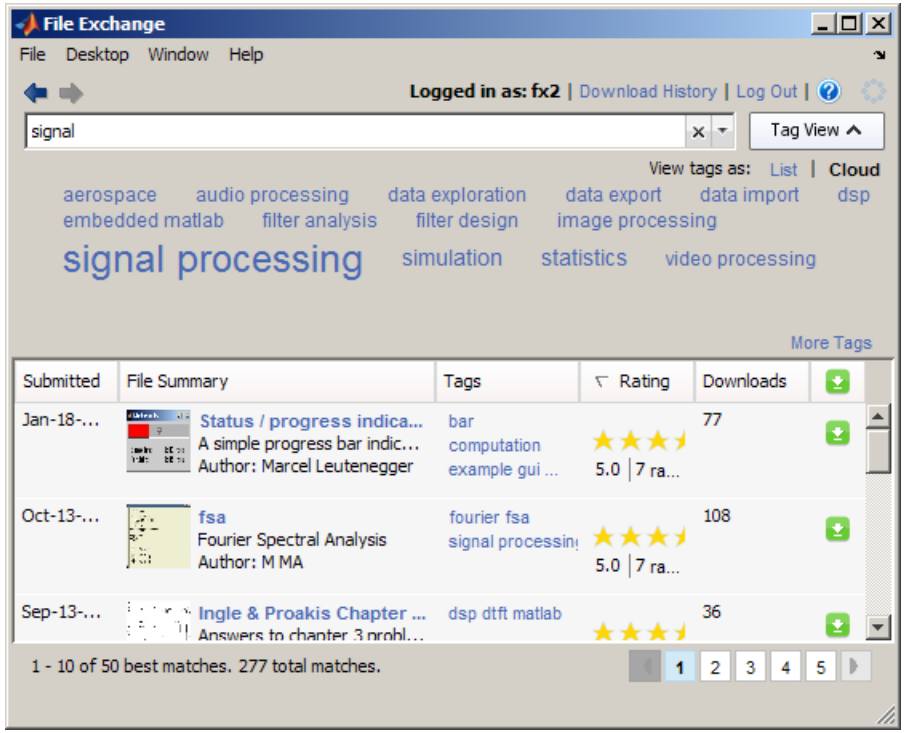
For more information, see "Using Search to Find Files in File Exchange" on page 7-13.

3 Because there are no obvious results or tags of interest, change the criteria. Change the sort order to show results that have the highest ratings first by clicking the **Rating** column header.

File Exchange:

- Lists the 50 highest rated files among the 277 matches. The most highly rated result is `Status / progress indicator`.
- Shows the most popular tags associated with the 50 files listed.
  - The `audio processing` tag is now among the most popular.
  - The `automotive` tag is no longer among the most popular tags for the 50 files listed.

For more information, see “Sorting the List of Files in File Exchange” on page 7-29.

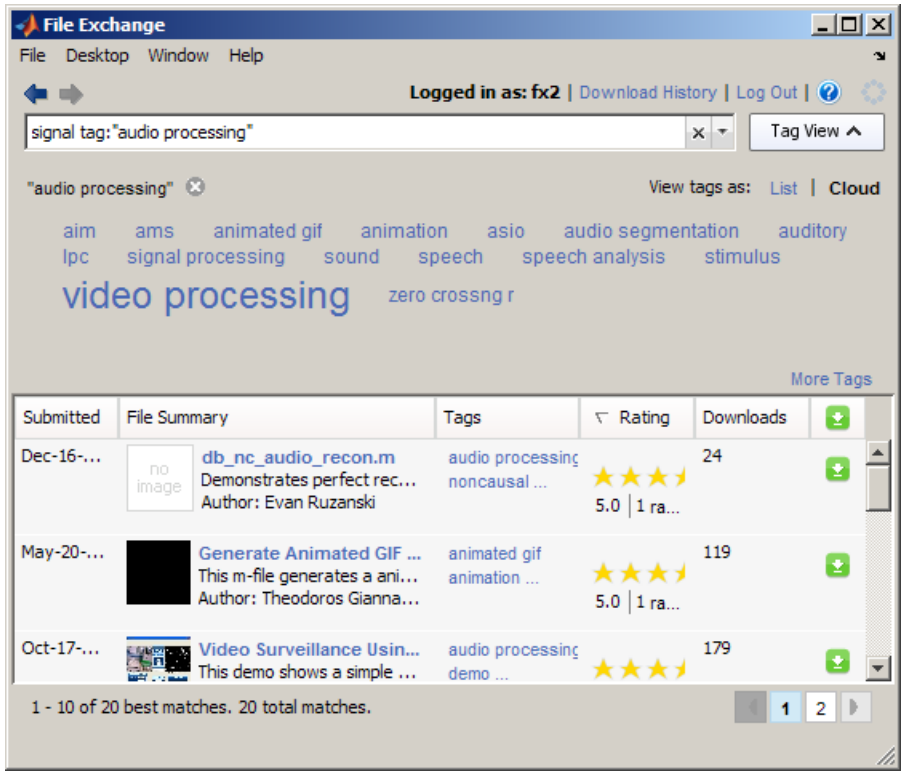


4 Narrow the results by selecting the `audio processing` tag.

File Exchange:

- Reports that 20 files are associated with the `audio processing` tag or variations of it, and have `signal` in the title, tag, or description.
- Shows the popular tags associated with the 20 files listed.

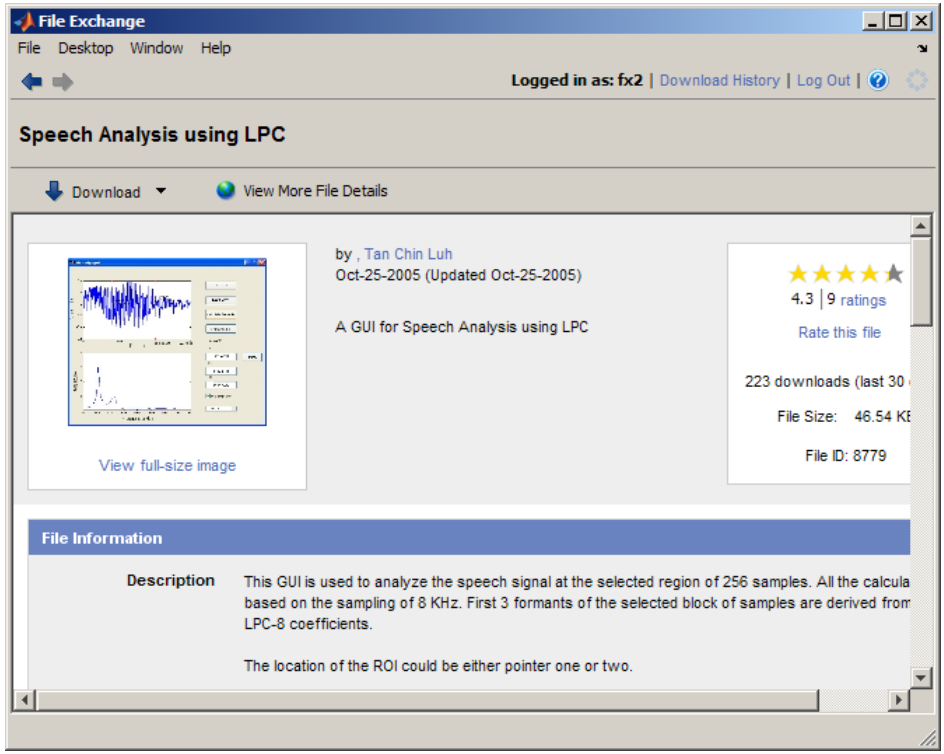
For more information, see “Finding Files Using Tags” on page 7-16.



5 Scroll through the list of files on the first page. `Speech Analysis using LPC` looks like it could be useful. Select the file name in the list of files.

File Exchange replaces the list of files with the details page for `Speech Analysis using LPC`. On the details page, view additional information to decide if you want to use the file. For example, the details page reports that the file requires the Data Acquisition Toolbox™ and Signal Processing

Toolbox products. For more information, see “Viewing Details About a File” on page 7-30.



6 Download the file to the current folder:

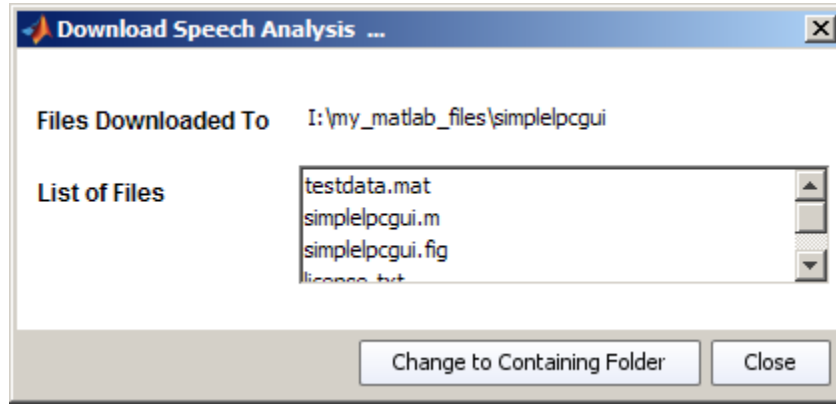
   a In the details page, open the menu on the **Download** button.

   b From the button menu, select **Download to Current Folder**.

   For more information, see “Downloading Files from the File Exchange Repository” on page 7-32.

7 In the resulting confirmation dialog box, click **Download**.

8 When the download completes, File Exchange reports the list of files and download location. Click **Change to Containing Folder** to access the files.



9 Go to the Current Folder browser. The current folder is `simplelpcgui` and contains the files you downloaded. Open the `simplelpcgui.m` file to review it. You can run the file. For more information, see "Best Practices for Using Files Provided by Other Users" on page 7-37.

For more options to find files, see "Example — Using Tags to Find Files in File Exchange" on page 7-18.

Finding Files in File Exchange — Searching and Using Tags

In this section...

About Finding Files in File Exchange

- Find files by using search words, searching for attributes, selecting tags, and sorting.
- You can use all the methods at the same time.
- It is more efficient to search first, and then refine the search results by selecting tags and changing the sort order.
- Because the repository changes daily, criteria you use to find files now could show different results when you use the same criteria later.

Using Search to Find Files in File Exchange

1. Go to the list of files.
2. Type search words in the search field and press **Enter**.

File Exchange finds files in the repository whose titles, descriptions, or tags contain the search words you entered.

For an example, see "Example — Finding and Downloading a File in File Exchange" on page 7-6 .

Syntax for Search Words

To view guidelines for entering search words:

1. Click the arrow in the search field.
2. From the menu, select **Using the Search Field to Find Files**.
3. View the relevant help topic.

Finding Files by Product, Author, and Other Attributes in File Exchange

You can search for files by their attributes, for example, files whose author is Jones. For more information, click the arrow in the search field and select **Advanced Search Help**.

Using Tags to Find Files in File Exchange

What Are Tags?

In File Exchange:

- Tags are keywords associated with a file.
- Users create tags and apply tags to files.
- Typically, a file has more than one tag applied to it.
- There is no relationship among the tags applied to a file.

Use tags to:

- Find files about a topic.
- Label files that pertain to a topic.

Ways to View Tags

Viewing Popular Tags for a List of Files. Popular tags are those tags users applied most often. Multiple users can apply a tag to a file. The popularity of a tag reflects the number of times the tag was applied by all users.

File Exchange shows popular tags above the list of files:

- When the search field is empty, File Exchange shows popular tags for the entire repository.
- After you select a tag or perform a search, File Exchange shows the popular tags associated with the resulting list of up to 50 files.

You can change the way that popular tags display:

- To show or hide popular tags, click **Tag View**.
- To see additional popular tags, make the File Exchange tool wider.
- To change the view of tags, click **Cloud** or **List**:
  - **Cloud** view — The font size of a tag indicates its popularity.
  - **List** view — All tags have the same font size. You can see the popularity of a tag by the number in parentheses.

Viewing More Tags for a List of Files. When you want to see more than just the popular tags for a list of files, click **More Tags**. The resulting window:

- Allows you to choose the view, as a cloud or a list

- Shows the 250 most popular tags in the entire repository when the search field is empty
- Shows all tags associated with the list of files when the search field is not empty
- Does not show tags you already selected
- Automatically closes when you click anywhere outside of it
- Remains open if you click its top edge or drag the window by its top edge to another location

To See Different Tags. If you do not see tags of interest in popular tags or **More Tags**:

- Change the search words, remove tags, or select different tags.
- When the search field is not empty, and there are more than 50 results, you can change the sort order to see different tags.

Viewing Tags for a File. You can view tags for a file:

- In the list of files, in the **Tags** column
- On the file details page, in **Everyone's Tags**

For more information about **Everyone's Tags** and **Your Tags**, click the information button in **Tags for This File**.

Finding Files Using Tags

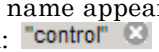
- "Selecting Tags to Find Files" on page 7-16
- "Removing Tags You Already Selected to Expand Results" on page 7-17
- "Directly Entering a Tag Name" on page 7-17

Selecting Tags to Find Files. You can select tags before or after entering search words. In general, it is more efficient to search first, and then refine the search results by selecting tags.

Select one or more tags from any of the following locations:

- Popular tags shown above the list of files
- The window that opens when you click **More Tags**
- The **Tags** column in the list of files
- **Tags for This File** on the file details page

After you select a tag:

- The tag name appears below the search field. Example of `control` tag selected: 

- `tag:"tagname"` appears in the search field.
- File Exchange looks for files that have variations of the selected tag associated with them.
- File Exchange reports the files that best match all your criteria.

For example, when you select the tag `control`, File Exchange finds files that have the tag `controls design`, `pid controller`, and other variations. The reverse is not true: if you select the tag `pid controller`, File Exchange does not find files with the tag `control`.

Removing Tags You Already Selected to Expand Results. When selecting a tag produces too few results, remove it to see more results. To remove a selected tag, do one of the following:

- Click the close button  for the tag.
- Delete `tag:"tagname"` from the search field.

Directly Entering a Tag Name. When you do not see a tag but want to use it to find files, you can directly enter the tag in the search field.

To enter the tag directly, type `tag:"partial_tagname"`, and press **Enter**.

For example, if you do not see the tag `control` in popular tags or **More Tags**, enter `tag:"control"` in the search field.

For more information, click the arrow in the search field and select **Advanced Search Help**.

Example — Using Tags to Find Files in File Exchange

The example illustrates advanced aspects of using tags to find files.

Note The File Exchange repository changes daily, so your results could differ from this example.

---

1. Open File Exchange and view the tags as a list: for **View tags as**, select **List**. See "Ways to View Tags" on page 7-15.

   You want to find files to help you compare your data, which has small sample sizes, to known distributions.

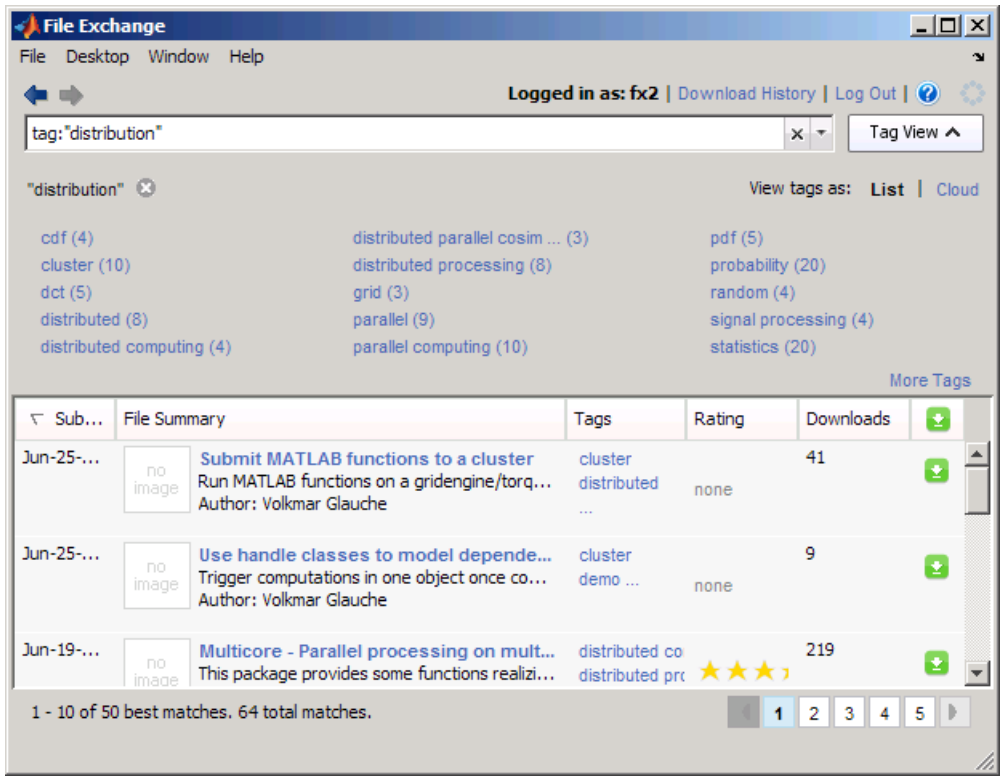
   The `statistics` tag looks relevant. But it could be too general because it was applied 712 times.

   You think a `distribution` tag could help, but do not see it.

2 Type `tag:"distribution"` in the search field, and press **Enter**. For more information, see “Directly Entering a Tag Name” on page 7-17.

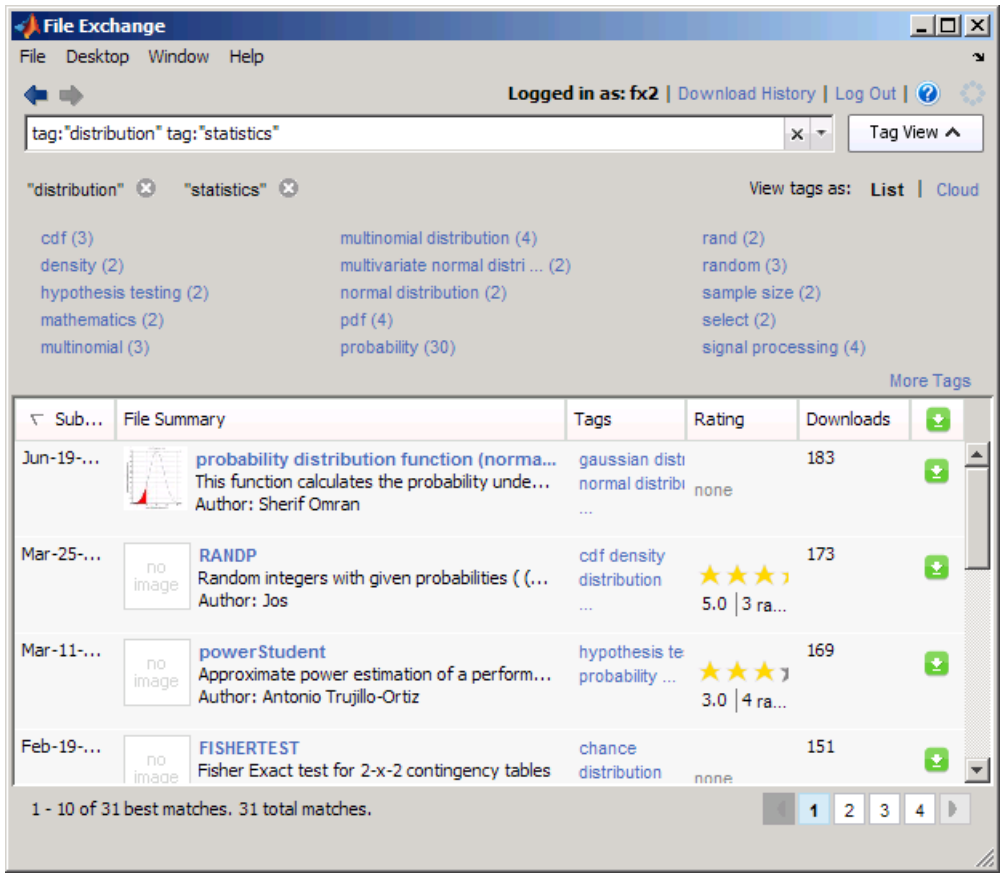
File Exchange reports 64 files that have the `distribution` tag or a variation of it applied to them.

The `statistics` tag was applied 20 times among the 50 files listed. The tag could be applied additional times among the 14 matches not listed.



3 Narrow your results by selecting the `statistics` tag. For more information, see "Selecting Tags to Find Files" on page 7-16.

File Exchange reports 31 files that have both the `distribution` and `statistics` tags applied to them. The number of times the `statistics` tag was applied increased from 20 to 31. Before you selected the `statistics` tag, it was applied only 20 times among the 50 files listed, as described in step 2. But there were 64 results in step 2, so the `statistics` tag was also applied 11 times among the 14 results that were not listed then.



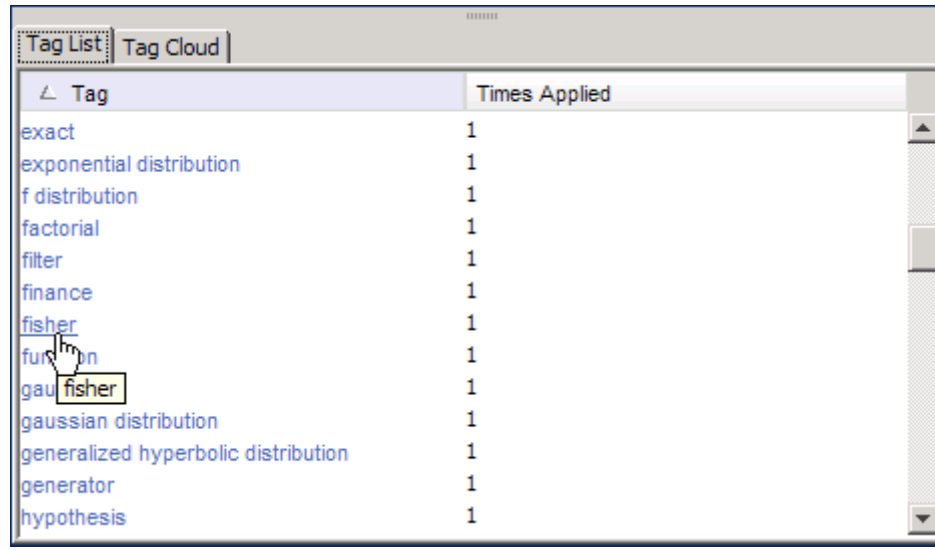
4 None of the popular tags are of interest. The `FISHERTEST` file is close to what you want. But the description shows it as being for a 2-by-2 contingency table and you have a 2-by-3 table.

   Look for a `fisher` tag to see if there are other files for the Fisher test:

   a Select **More Tags**.

   b From the resulting window, select the `fisher` tag to add it to your criteria.

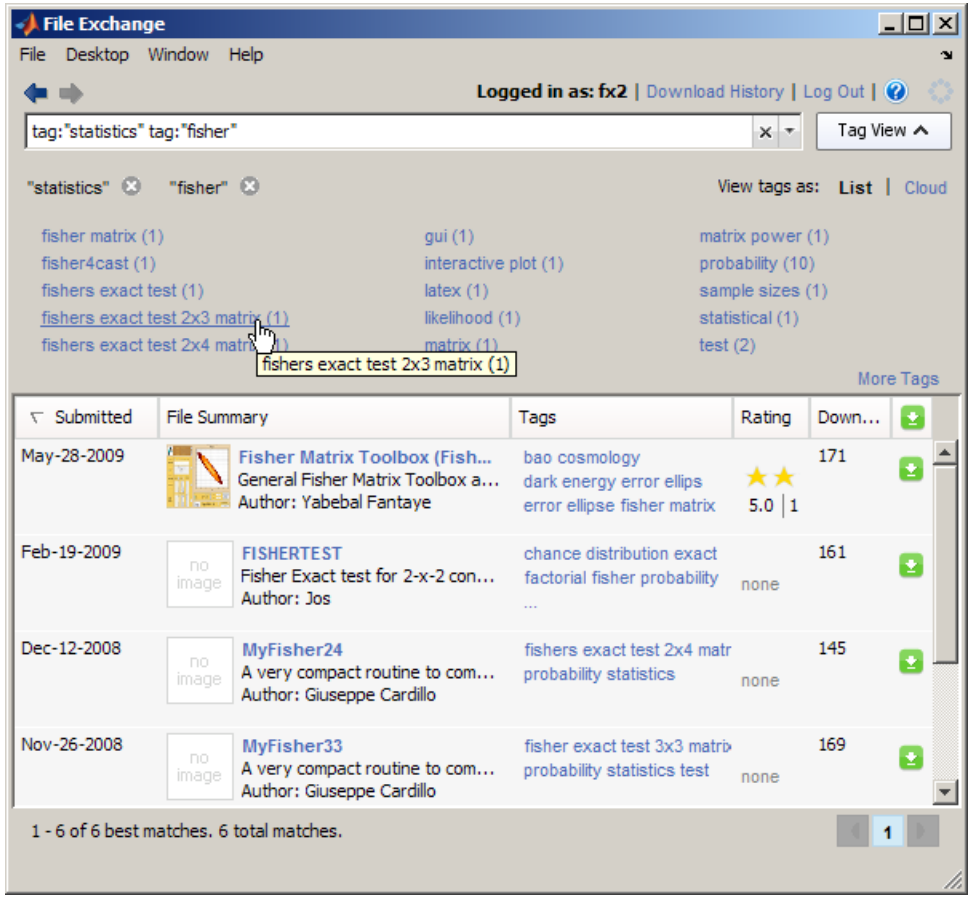
   For more information, see “Viewing More Tags for a List of Files” on page 7-15.



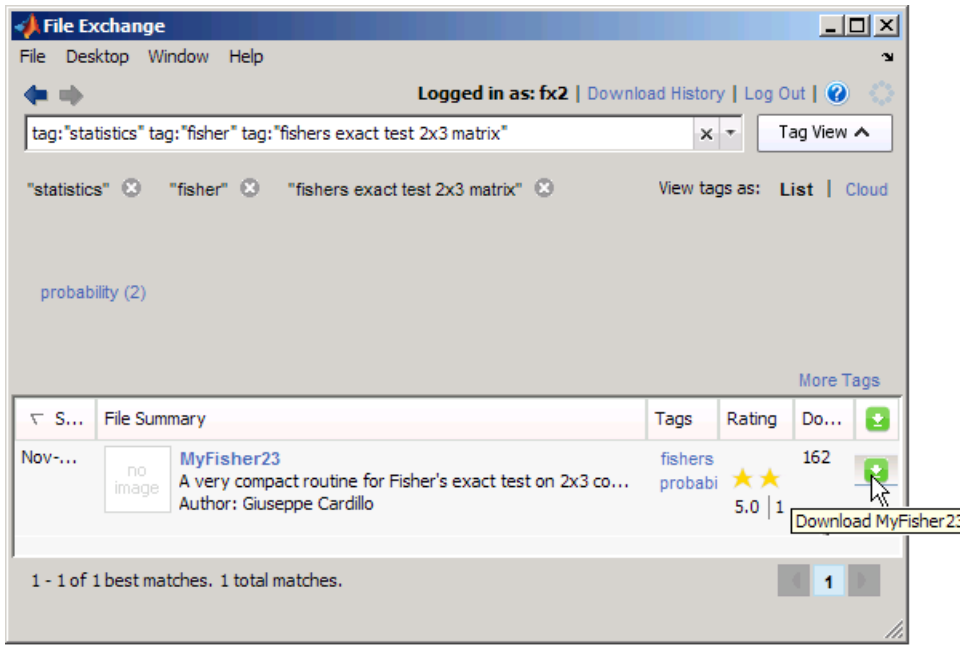
5 File Exchange reports one file that uses all three specified tags. The file is not of interest.

Expand the results by removing a tag from your criteria. Remove the least relevant tag, `distribution`, by clicking the close button for it. For more information, see "Removing Tags You Already Selected to Expand Results" on page 7-17.

6 File Exchange reports six files with the tags `statistics` and `fisher` applied to them. The tags include `fishers exact test 2x3 matrix`, which is what you need. Select the tag.



7 One file has the three selected tags applied to it. The file is relevant to you, has a good rating, and was downloaded often. Download it by clicking the Download button.



Applying a Tag to a File

When a tag is associated with a file, you can apply the tag to the file to identify it as relevant to you.

To apply a tag to a file, go to the details page for the file and use **Everyone's Tags**.

For more information, including an example, click the **Tags for This File** information button on the file details page.

Adding a New Tag to a File

You can add a new tag to a file. See "Adding Tags to a File" on page 7-39.

Clearing Your Criteria

Clear the search field. The default list of files displays, using the sort order you last specified. For more information, see "Viewing the Default List of Files" on page 7-28.

Getting Better Results Using Search and Tags

When too few or too many files match your criteria, try the following suggestions to get better results.

To Get More Matches	To Get Fewer Matches	Explanation
Remove tags.	Add tags.	Because results must have *all* the tags you chose, specifying fewer tags results in more matches.
Directly enter partial tag names instead of selecting tags.	Select tags or enter full tag names.	You can get more results when you type `tag:"partial_tagname"` in the search field than when you select a tag. For example, typing `tag:"control"` in the search field instead of selecting the tag `design controller` results in more matches.
Use search words instead of tags.	Use tags instead of search words.	You get more results by searching for a word than by selecting a tag of the same name. For example, typing `control` in the search field instead of selecting the tag `control` results in more matches.
Remove " " from around search words.	Add " " from around search words.	Without quotation marks, you find files that have all the search words, but not necessarily in sequence.
Try again in about 15 minutes.	N/A	When you add a new tag and use it to find files, File Exchange could report no files found. It could take up to 15 minutes after adding a new tag before you can use it.

To Get More Matches	To Get Fewer Matches	Explanation
Check the search field for mistakes.		You could have inadvertently removed a meaningful space or quotation mark. For guidelines, click the arrow in the search field and select **Using the Search Field to Find Files**
Be sure that search attributes and values are valid.		For correct names and values, click the arrow in the search field and select **Advanced Search Help**.
Start over by clearing the search field.		If you want to start over instead of changing criteria you already provided, clear the search field.

If you have too many results, try the opposite of the suggestions:

- Add tags
- Use full tag names
- Use tags instead of search words

Viewing and Sorting the List of Files in File Exchange

In this section...

Viewing the List of Files in File Exchange

Viewing the Default List of Files

The default list of files:

- Is what you see when you open the File Exchange tool. The search field is empty.
- Shows the 50 files most recently submitted to the repository.

To view to the default list of files at any time:

1. Click the clear button ⊗ in the search field.
2. Sort the files to show the most recently submitted files by clicking the **Submitted** column header once or twice.

Viewing the List of Files that Match Your Criteria

When you perform a search, select a tag, or change the sort order, File Exchange:

- Lists up to 50 files that best match your criteria
- Reports the total number of files in the repository that match your criteria

File Exchange does not list more than 50 results at once for performance reasons.

To see up to 50 different files, change the search words, tags, or sort order.

Sorting the List of Files in File Exchange

Use the sort features to help you find files you want:

- To sort files, select the **Submitted**, **Rating**, or **Downloads** column header. File Exchange sorts all files in the repository that match your search words and selected tags.
- To reverse the sort order, click the column header again.

Sorting applies to all files matching your criteria, not just to the files listed:

- When there are more than 50 reported matches, performing a sort usually changes the list of files. It sorts not only the files listed, but all files in the reported number of matches. It does *not* merely change the order of the files listed.
- When there are 50 or fewer matches reported, sorting merely changes the order of the files listed.

The new sort order remains in effect until you do one of the following:

- Change the sort order again
- Log out of File Exchange
- Exit MATLAB

Sorting by Number of Downloads

The File Exchange desktop tool shows the number of downloads for the last 30 days.

Viewing Details About a File

In this section...

Viewing the File Details Page

To get more information about a file than what you see in the list of files, click the file name.

The file details page opens. It includes:

- Files included in the submission
- Required products
- File size
- Comments from users
- A full description, which can contain algorithms and code snippets

To return to the list of files from the file details page, click the back button .

Viewing the Contents of a File

To view the contents of a file, download it and open it.

To view the contents of a file without first downloading it:

1. Go to the file details page.
2. Click **View More File Details**.

   The file details page opens on the Web interface to File Exchange.
3. Click **Download Now**.
4. Choose the option to open the file.

For submissions that include more than one file, choose the file you want to view from the unzip tool.

Downloading Files from the File Exchange Repository

In this section...

About Downloading Files

After finding a file of interest, get the file from the repository to use it in MATLAB.

Choose from these options when downloading files:

- Download from the list of files or from the details page
- Download to the default folder, the current folder, or another folder

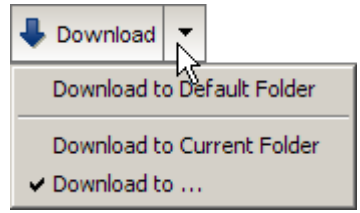
Downloading from the List of Files

The file downloads to the last download folder used. If you did not previously specify a folder, the file downloads to the default folder.

1. Go to the list of files.
2. For the file you want to download, click the download button  .
3. Confirm or cancel the download in the resulting dialog boxes.

Downloading from the File Details Page to a Location You Choose

1. Go to the file details page.
2. Click the arrow on the **Download** button.



3. From the drop-down menu, select a location.
4. Confirm or cancel the download in the resulting dialog boxes.

To download a file to the last download location used, click **Download** on the file details. page.

The Default Folder for Downloaded Files

The default location for downloads is `Documents/MATLAB/Downloads`.

On your system, `Documents` could be `My Documents` or something similar.

Which Location Should You Choose When Downloading Files?

To...	Use this Location
Easily find your downloaded files	Default folder
Easily run a file immediately after downloading it	Current folder in MATLAB
Use a hierarchy of folders for managing files	A folder you specify

Downloading a Submission that Consists of Multiple Files

A "file" in the File Exchange repository is a submission that can be one file or can consist of multiple related files.

For a submission containing multiple files, File Exchange:

- Downloads a single zip file containing the files.
- Creates a folder within the download folder you specified.
- Unzips the files into the folder.

Why You Might See Multiple Folders for One Download

You could see a folder within a folder for one download.

If the author submitted the files to the repository in a folder, File Exchange maintains the files in the folder. When you download, the standard zip and unzip process creates a second folder level.

You can remove one of the extra folders. If you remove an extra folder, the download history becomes inaccurate for that file.

Viewing and Locating Files You Downloaded

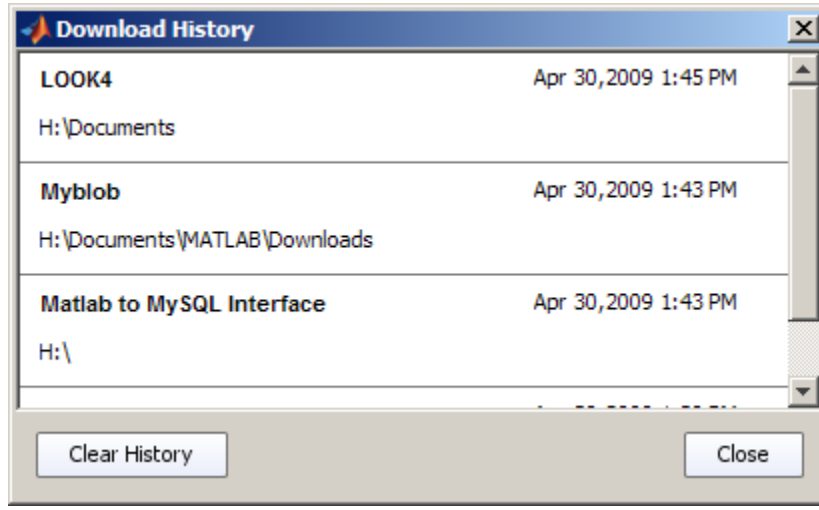
Use the download history to view and locate files you downloaded.

- "Viewing a Log of Files You Downloaded" on page 7-34
- "Locating a File You Downloaded" on page 7-35
- "Viewing Details for a File You Downloaded" on page 7-35
- "Clearing the Download History" on page 7-36

Viewing a Log of Files You Downloaded

1. Click the **Download History** button.
2. In the resulting dialog box, view the files you downloaded.

Example of download history window:



If you changed the name or location of a downloaded file, the download history does not reflect your changes.

Locating a File You Downloaded

To find a file you downloaded:

1. Click the **Download History** button.
2. In the resulting dialog box, select a file.
3. From the context menu, select **Change to Containing Folder**.

The folder that contains the file becomes the current folder in MATLAB.

To find files easily without using the download history, always download to a single location, such as the default folder, `Downloads`.

Viewing Details for a File You Downloaded

To display the file details page for a file you downloaded:

1 Click the **Download History** button.

2 In the resulting dialog box, select a file.

3 From the context menu, select **Find in File Exchange**.

Clearing the Download History

To remove all entries in the download history dialog box:

1 Click the **Download History** button.

2 In the resulting dialog box , click the **Clear History** button.

Best Practices for Using Files Provided by Other Users

In this section...

Ensure MATLAB Can Access the File

To ensure that MATLAB can access the file, the file must be in the current folder or its folder must be on the MATLAB search path. Here is one method to use:

1. Always choose the default `Downloads` folder as the download location
2. Add the `Downloads` folder and all its subfolders to the search path.

   For more information, see "Adding Folders to the Search Path" on page 6-49.

Consult the File Details Page

Review relevant information on the file details page, including:

- The description, which can include advice.
- Required products, as reported by the author.
- The release number (version) for MATLAB, as reported by the author. If you use a different release, the file might not run as expected.
- Comments provided by other users, which can include tips and workarounds.

Look for Updates to the File

A newer version of the file could address issues you have. Here are ways to find out about changes:

- View the last date the file changed on the file details page.
- View a history of changes made to the file. To see the changes, go to the file details page and click **View More File Details**. The file details page at MATLAB Central opens, and includes the log of updates.
- Get notification of changes to a file by adding the file to your watch list. To use the watch list, go to the file details page and click **View More File Details**. The file details page at MATLAB Central opens. On the page, select **Watch this File**.

When you download the new version of the file to the same location as the existing version, File Exchange:

- Replaces the existing version, if you have not changed the file.
- Changes the extension for the existing version to `.bak`, if you had changed the file.

Read the File

You can learn more about how a file works by reviewing it:

- Open the file and skim the comments and code. Look for potential function or variable name conflicts between your files and the downloaded files. For more information, see "About Name Conflicts and Shadowed Files" on page 6-41.
- Look for a `readme` or related file, which could have been provided in the download folder.

Ask Questions

If you still need help to use the file, try either of the following:

- Present your question in a comment about the file. For more information, see "Providing Comments About a File" on page 7-41.
- Contact the author. Select the author name on the file details page to display the author profile, which includes contact information.

Contributing to the File Exchange Repository

In this section...

How You Can Contribute to the Repository

File Exchange is a valuable resource because of the contributions of users like you.

You can contribute by:

- Adding tags for existing files
- Providing feedback for files you use by rating the files and adding comments
- Submitting your own files

Adding Tags to a File

1. Go to the file details page.
2. In the **Add New Tags** field near the bottom of the page, type the name of the tag to add:
   - To add multiple tags at once, separate each tag with a space.
   - To add a tag that has multiple words, put the words inside quotation marks. For example `"pid controller"`.
   - You can enter up to 64 characters for a tag name, including spaces. File Exchange truncates characters after the 64th.
3. Click **Submit**:

- **Your Tags** includes the tag.
- **Everyone's Tags** includes the tag. If the tag had already been applied to the file, the number of times the tag was applied to the file increases by one. The tag becomes more popular.
- It could take up to 15 minutes until you can use the newly added tag in File Exchange.

For more information, including an example, click the **Tags for This File** information button on the file details page.

Removing Tags from a File

You have permission to remove tags you added to a file. You cannot remove tags added by other users.

To remove a tag:

1. Go to the file details page.
2. In the **Tags for This File** area, under **Your Tags**, click - (the remove button) for the tag you want to remove.

After removing a tag:

- **Your Tags** no longer includes the tag.
- If other users applied the tag to the file, the tag remains in **Everyone's Tags**. The number of times that tag was applied to the file decreases by one.

Rating a File

Assign a number of stars to rate a file:

1. Go to the file details page.
2. Go to **Rate This Submission** at the bottom of the page.
3. Click a star. For example, click the third star from the left to assign a rating of three stars.
4. Click **Submit**.

If you enter a rating but do not want to submit it, use the back button to leave the file details page.

To change a rating you already submitted, rate the file again. It would be helpful to add a comment about why you changed your rating. File Exchange maintains only your most recent rating. File Exchange reports the number of times users rated a file. When you rate a file more than once, the number of ratings includes all the times you rated a file.

It could take up to 15 minutes until your rating appears in the file list in File Exchange.

Providing Comments About a File

Provide feedback about a file you downloaded. Use comments to let the author and other users know:

- What was useful
- What was problematic
- Ways of using the file

To add comments for a file:

1. Go to the file details page.
2. Go to **Add Comments** at the bottom of the page.
3. Type your comments in the field.
4. Click **Submit**.

Submitting Your Files to the Repository

Use the Web interface to submit your own files to the File Exchange repository. Click this link to go directly to the page where you can submit your file: MATLAB Central File Exchange — Submit New File

Frequently Asked Questions About File Exchange

In this section...

What Is File Exchange?

- The File Exchange tool allows you to find and get files created by users of MathWorks products.
- The files are at the MATLAB Central community area of the MathWorks Web site.

For more information, see "Before Using File Exchange" on page 7-2.

How Do I Use File Exchange?

1. Find files you want by
   - Entering search words.
   - Selecting tags (keywords).

- Sorting the list of files.

2. View details for a file by clicking the file name.

3. Get a file you want to use by clicking the download button.

4. After using a file, rate it and provide comments.

For an overview, watch this video: Accessing the File Exchange from the MATLAB Desktop.

For more information, see "How To Use the File Exchange Desktop Tool" on page 7-5.

How Does the File Exchange Desktop Tool Relate to File Exchange on the Web Site?

- The desktop tool accesses the File Exchange repository at MATLAB Central on the MathWorks Web site.
- You can also access the repository using the interface at the Web site.
- Both methods of access offer the same basic features, but there are some differences in features and results.

For more information, see "Ways to Access the File Exchange Repository" on page 7-3.

Why Do I See Only 50 Files and How Can I See More?

File Exchange lists up to 50 files at once for performance reasons:

- By default, you see the 50 most recent submissions.
- When you search, select tags, or change the sort order, you see up to 50 files that best match your criteria.
- If the list of files does not include files you want, change your criteria.

For more information, see "Viewing and Sorting the List of Files in File Exchange" on page 7-28.

What Are Tags and How Do I Use Them?

- Tags are keywords that users associate with files.
- Select tags to help you find files you are interested in.

For more information, see "Using Tags to Find Files in File Exchange" on page 7-14.

What Are the Tags Above the List of Files?

The tags shown above the list of files are the popular tags, in alphabetical order:

- When the search field is empty, you see the most popular tags for all files you can access in the File Exchange repository.
- After you perform a search or select tags, you see the most popular tags associated with the resulting list of files.

You can view popular tags as a cloud or list:

- In cloud view, the font size of the tag name indicates how popular the tag is.
- In list view, the number of times the tag has been applied appears in parentheses, numerically indicating its popularity.

For more information, see "Viewing Popular Tags for a List of Files" on page 7-15.

How Can I See Other Tags?

- Make the File Exchange tool wider, which could show additional popular tags.
- Change the search words, selected tags, or sort order to show different popular tags.
- Click **More Tags** to view more tags in a separate window.

For more information, see "Ways to View Tags" on page 7-15.

Why Are the Tags Changing?

When you select a tag, perform a search, or change the sort order, File Exchange looks through the repository for all files that match your criteria. The list of files changes to show up to 50 files that match your criteria. Because popular tags reflect the resulting list of files:

- The popular tags shown above the list of files change.
- The popularity of each tag changes:
  - In cloud view, the font size changes.
  - In list view, the number in parentheses changes.

For more information, see "Using Tags to Find Files in File Exchange" on page 7-14.

Is Search Looking Inside Files?

File Exchange search does not look inside files at the code or comments.

File Exchange looks for matching words in information associated with files. Search looks in:

- Titles
- Descriptions
- Tags

For more information about searching, click the arrow in the search field, and select **Using the Search Field to Find Files**.

How Can I Start Over When Looking for Files?

Clear the search field to return to the default list of files. For more information, see "Viewing the Default List of Files" on page 7-28.

How Can I Choose Where to Download a File To?

- To specify the folder for your downloaded files, use the file details page to perform the download.
- When you download from the list of files, the file downloads to the last location you downloaded to.

For more information, see “Downloading Files from the File Exchange Repository” on page 7-32.

How Do I Contribute My Files to the Repository?

To submit files you created to the File Exchange repository, use the Web interface, MATLAB Central File Exchange — Submit New File.

For more information, see “Contributing to the File Exchange Repository” on page 7-39.

8

Editing and Debugging M-Files

MATLAB software provides powerful tools for creating, editing, and debugging files, as detailed here. For information about the MATLAB language and writing M-files, see the Programming Fundamentals documentation.

Begin with Existing Code

In this section...

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 to modify the code.

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 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 Functions

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 and use it as a basis for your file. Be sure to save the file using a different name and in a folder that is not in `matlabroot/toolbox`. See "Saving M-Files" on page 8-94 for details.

Demos and Examples

The MATLAB product 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 "Using Demos and Code Examples" on page 4-12.

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

File Exchange enables you to use files that were created by other users. To save time and get new ideas, consider seeing if someone has already provided a file similar to what you need before creating a new file. In addition, consider submitting files you create for others to use. For more information, see Chapter 7, "File Exchange — Finding and Getting Files Created by Other Users"".

Ways to Edit, Evaluate, and Debug M-Files

There are several methods for creating, editing, evaluating, and debugging files with MATLAB software.

Creating and Editing Files — Options	Instructions
MATLAB Editor	See "Starting, Creating Files, and Closing the Editor" on page 8-7, and "Creating New Files in the Editor" on page 8-8. You can create, open, edit, and save M-files as well as other file types in the MATLAB Editor—see "Creating and Editing Other Text File Types" on page 8-12.
Any text editor, such as Emacs or vi	To specify another editor as the default for use with MATLAB software, 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 or debugging functions.

Debugging M-Files – Options	Instructions
General debugging tips	See "Finding Errors, Debugging, and Correcting M-Files" on page 8-116.
MATLAB Editor	See • "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 8-119 to identify errors and make improvements. • "Debugging Process and Features" on page 8-151 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 8-151.

See Customizing the Editor by Setting Preferences for information on setting up the editing and debugging environment to best meet your needs.

For information about the MATLAB language and writing M-files, see the Programming Fundamentals documentation.

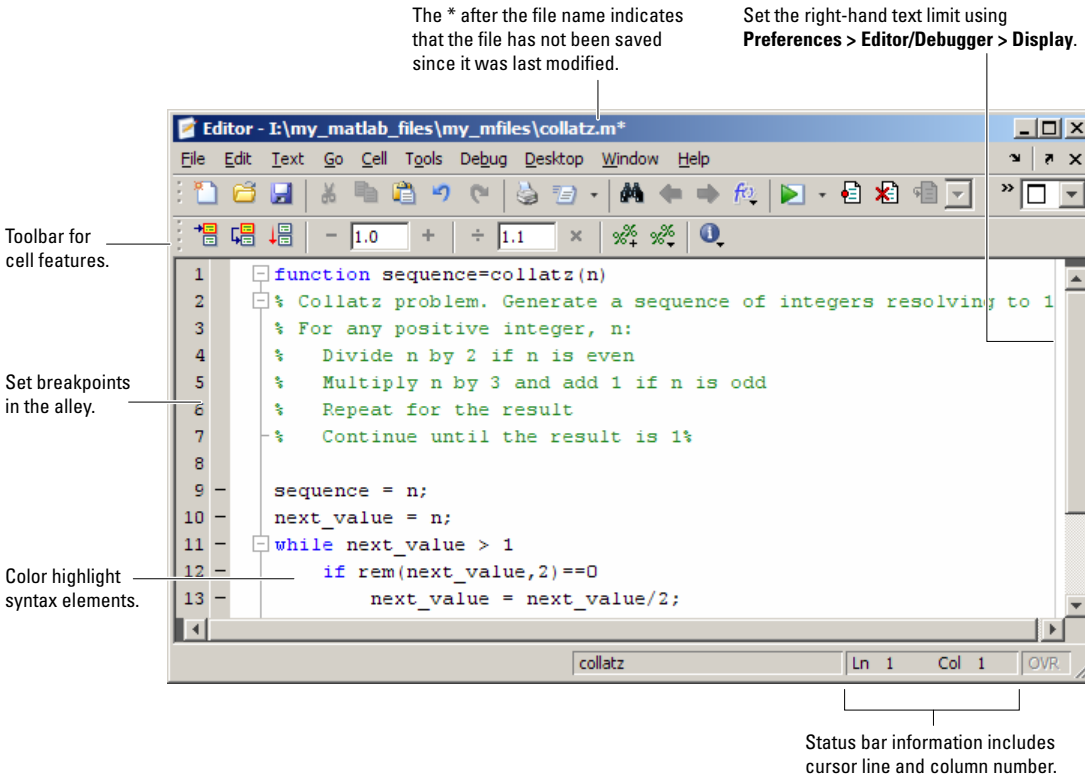
Starting, Creating Files, and Closing the Editor

In this section...

Starting the Editor

The MATLAB Editor provides a graphical user interface for basic text editing features for any file type, as well as for M-file debugging. The Editor is a single tool that you can use for editing, debugging, or both. There are various ways to start the Editor. The Editor automatically starts when you open a document or create a new one. Once started, you can customize the Editor to suit your needs.

This figure shows an example of the Editor outside of the desktop opened to an existing M-file, and calls out some of the tool's useful features.



Creating New Files in the Editor

You can create a new blank or prepopulated file in the Editor. A prepopulated file provides elements typically included in the file type you choose to open.

With the Editor as the current (active) window, you can:

- Create an empty M-file (such as for a script) or a text file (such as for an XML or HTML file).

  Choose **File > New > Blank M-File** or click the New M-File button on the MATLAB desktop toolbar.

- Create an M-file prepopulated with basic M-file function elements.

  Choose **File > New > Function M-File**.

- Create a class definition M-file (`classdef`), prepopulated with basic M-file class structure elements.

  Choose **File > New > Class M-File**.

In all cases, the Editor opens an untitled file in the MATLAB current folder.

Other tools also provide features for creating new M-files. For details, see:

- "Performing Actions on Statements in the Command History Window" on page 3-63.
- "Creating Folders Using the Current Folder Browser" on page 6-30.

Function Alternative for Creating New Files

You can do either of the following:

- To create a blank unnamed file in the Editor, type `edit` in the Command Window.
- To create a blank file named `filename.ext` in the Editor, type `edit filename.ext`. If `filename.ext` exists in the current folder or on the MATLAB search path, MATLAB opens the existing file. If `filename.ext` does not exist in the current folder or on the MATLAB search path, by default, a confirmation dialog box appears. It asks if you want to create the named file.

For more information about the confirmation dialog box, see "Confirmation Dialogs Preferences" on page 2-133.

Opening Existing Files in the Editor

To open an existing file in the Editor, click the Open file button  on the desktop or Editor 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 file name, and click **Open**. If you access the **Open** dialog box from the desktop, the current folder files are shown, but if you access it from the Editor, the files

in the folder for the current file are shown. For special considerations on the Macintosh platform, see "Using File Browser GUIs on Macintosh Platforms to Navigate Within the MATLAB Root Folder" on page 2-118.

The Editor opens, if it is not already open, with the file displayed. You can have multiple Editor files open at once, and the location of the files and the Editor are determined by document positioning guidelines. You can rearrange the documents to suit your needs. For details, see "Opening and Arranging Desktop Documents" on page 2-21.

To make a document in the Editor 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 software interprets any M-file that contains `%%` at the start of a line as including cells and the Editor reflects 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 Show Cell Mode 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**. If cell mode is disabled when you quit a MATLAB session, it is disabled the next time you start a MATLAB session; the converse is true as well.

Other Methods for Opening Files in the Editor

These are other ways to open files in the Editor:

- Drag a file from another MATLAB desktop tool or a Microsoft tool into the Editor. For example, drag files from the Current Folder browser, or from Windows Explorer.
- Open files from the Current Folder browser—see "Opening and Running Files Using the Current Folder Browser" on page 6-36.
- Select a file to open from the most recently used files, which are listed at the bottom of the **File** menu in the Editor 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 or another desktop tool such as the Command Window, select a file name, 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 8-73.
- Set a preference that specifies that a MATLAB session, upon startup, is 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. For example, type

```
edit collatz.m
```

to open the file `collatz.m` in the Editor, where `collatz.m` is on the search path or in the current folder. Use the relative or absolute path for the file you want to open if it is not on the search path or in the current folder.

Arranging Editor Documents

You can arrange the size and location of M-files and other text documents you open in the Editor. Editor documents follow the same arrangement practices as other desktop documents. For details, see "Opening and Arranging Desktop Documents" on page 2-21.

Creating and Editing Other Text File Types

You can edit any type of text file using the MATLAB Editor. For example, you can open and edit an HTML file. However, you can run or debug only M-files from the Editor.

When working with C/C++, Java, TLC, VHDL and Verilog programming languages, as well as XML or HTML, you can specify syntax highlighting and indenting preferences appropriate to those languages. Select **File > Preferences > Editor/Debugger > Language**. For details, click the **Help** button in the dialog box.

Closing the Editor

To close the Editor, click the **Close** box in the title bar of the Editor. This is different from the **Close** box in the menu bar of the Editor, which closes the current file when multiple files are open in a single window.

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 variables in the Variable Editor, and it will close the tools as well, that is, the Editor and Variable Editor, for example.

When you close the Editor and any of the open files have unsaved changes, you are prompted to save the files.

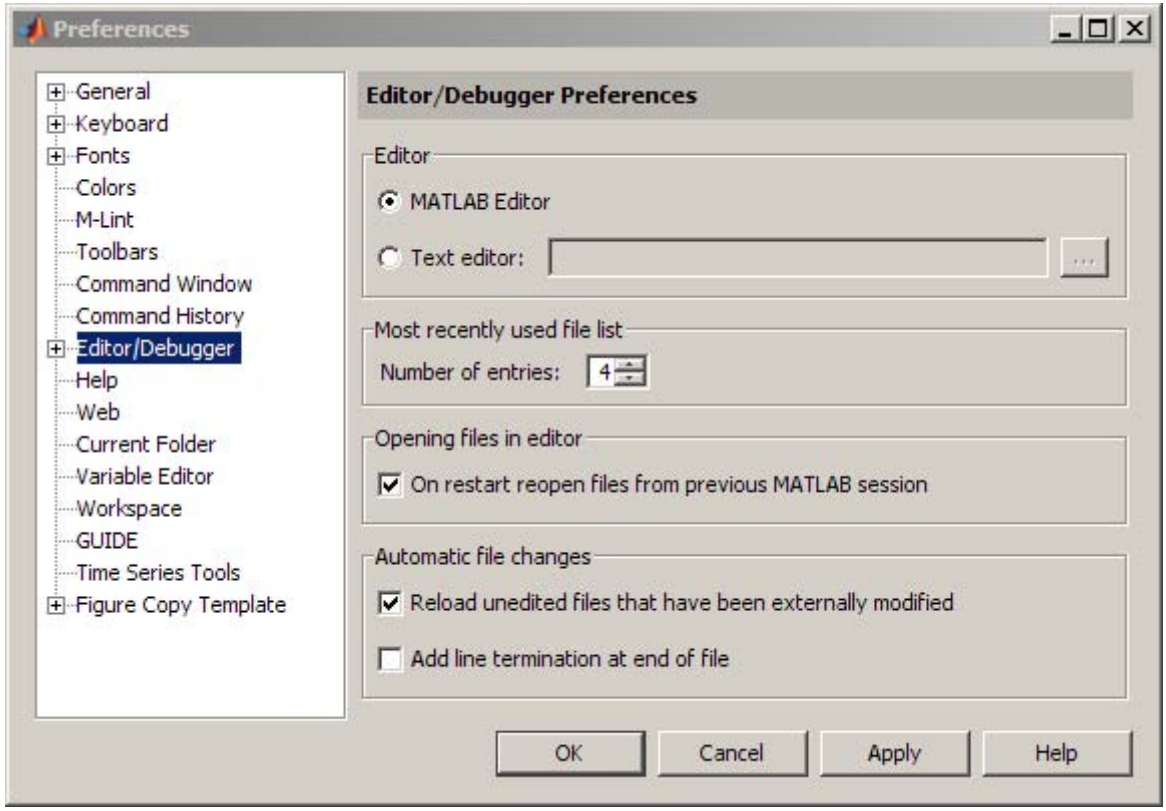
Customizing the Editor by Setting Preferences

In this section...

Overview of Setting Preferences for the Editor/Debugger

Using preferences, you can specify the default behavior for various aspects of the Editor.

To set preferences for the Editor, select **File > Preferences > Editor/Debugger**. The Preferences dialog box opens showing **Editor/Debugger Preferences**.



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.

You can also set preferences for the Editor toolbars. Select **File > Preferences > Toolbars**, and from the **Toolbar** drop-down list select **Editor** or **Editor Cell Mode**, depending on the toolbar for which you want to set preferences.

Setting General Preferences for the Editor/Debugger

Select **File > Preferences > Editor/Debugger** to specify the preferences on the main pane, listed here. Or click the **+** to see additional Editor/Debugger preferences.

Editor

MATLAB Editor. Selecting **MATLAB Editor** means that the MATLAB desktop uses the built-in Editor.

Text editor. To specify a text editor other than the MATLAB Editor, such as Emacs or vi, to be used when you open an M-file from within MATLAB, select **Text editor**. Specify the full path for the editor application you want to use.

For example, specify `c:/Applications/Emacs.exe` in the **Text editor** field, and then open a file using **Open** from the **File** menu in the MATLAB desktop. The file opens in Emacs instead of in the MATLAB Editor.

Note that even if you use another editor, Editor/Debugger preferences are still available because some of the Editor/Debugger preferences apply to other MATLAB tools.

Most Recently Used File List

Use this preference to specify the number of files that appear in the list of most recently used files at the bottom of the **File** menu. You select a file from the list to open it.

Opening Files in Editor

On restart. To start a MATLAB session and automatically open the files that were open when you last shut down MATLAB, select the item **On restart open files from previous MATLAB session**. If the item is not selected and you close MATLAB when there are files open in the Editor, the next time you start MATLAB, the Editor is not opened upon startup.

Automatic File Changes

Reload unedited files that have been externally modified. This option is useful when you edit files in the MATLAB Editor and outside of MATLAB.

With the option selected, the Editor automatically reloads another version of the file as described here. When a file is open in the Editor and the same file is also open outside of MATLAB, the Editor automatically uses the version open outside of MATLAB if you have not edited the file in the Editor.

When you clear this preference, MATLAB displays a dialog box notifying you that the current file was changed outside of MATLAB, and asks if you want to use the latest version.

Add line termination at end of line. With this preference selected, if the last line in the file is not empty, the Editor automatically adds a new empty line to the end of the file. Select this preference if you use your files with other products that expect a new line (sometimes referred to as a `<CR>`) at the end of the file.

Setting Display Preferences

Select **File > Preferences > Editor/Debugger > Display** to specify these preferences:

- “General Display Options” on page 8-17
- “Right-Hand Text Limit” on page 8-17
- “Cell Display Options” on page 8-19

See also Desktop Preferences.

General Display Options

Highlight Current Line. Select this preference to highlight the current line, that is, the row 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. Then specify the color used to highlight the line.

Show line numbers. Select this preference to show line numbers along the left edge of the Editor window. Showing line numbers allows you to easily use various Editor features, such as **Edit > Go To Line**. When you clear this item, line numbers are not shown.

Enable datatips in edit mode. Select this preference to see datatips while editing an M-file. By default, datatips do not display while editing (edit mode), although they always display while debugging (debug mode). In edit mode, the datatips display the values of variables in the base workspace, so this is useful for script M-files rather than function M-files. In edit mode for a function M-file, the datatip for a variable displays a value if that variable also exists in the base workspace—the datatip displays the value of the base workspace variable, not the value of the variable in the function M-file.

While you are debugging, you cannot turn off the display of datatips, and they show the value of the variables in the workspace selected in the **Stack**. For more information about datatips, see "Viewing Values as Data Tips in the Editor" on page 8-164.

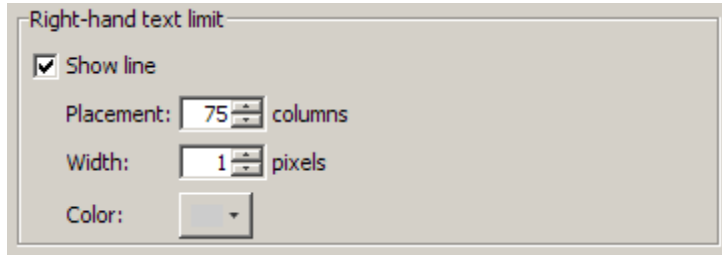
Right-Hand Text Limit

By default, a gray vertical line that is 1-pixel wide appears at column 75 in the Editor to indicate when a line of code becomes wider than desired. You might want to set a right-hand text limit for reasons such as the following:

- To prevent the need to scroll from left to right to see an entire line of text in the Editor
- To keep each line below a limit imposed by another text editor in which you intend to view the code
- To keep each line below a character limit required to ensure that the file will print without cropping text

You might need to print a test page to determine the most appropriate value for the printer printing the file, the margin settings for the printer, and the font size you are using.

Note This limit is a visual cue only and does not prevent text from exceeding the limit. For information on setting a value to automatically wrap comment text at a specified column number, see "Formatting Comments in M-Files" on page 8-44.



Show line. Select this preference to display the vertical line in the Editor. Clear it to hide the vertical line.

Placement. Type or select a new value for this field to change the column where the vertical line appears,

The actual column number where you set this preference is most useful when the font preference for the Editor is a monospaced font. In a monospace font, all characters are the same width. See "Setting Fonts Preferences for Desktop Tools" on page 2-141 for details.

Width. Type or select a value for this field to change the width (in pixels) of the vertical line.

Color. Click the down arrow next to the color block to open a palette of colors from which you can choose a new color for the vertical line. Increasing this value might improve the visibility of the vertical line on liquid crystal displays (LCDs) and projectors.

Cell Display Options

Use cells for testing and publishing code, as described in "Using Cells for Rapid Code Iteration and Publishing Results" on page 8-185.

Highlight cells. Select this item to highlight the current cell and select a color for the highlighting. Pale yellow is the default color.

Show lines between cells. Select this preference to display a faint gray horizontal line above each new cell. The line does not appear in the published M-file, nor in the printed M-file.

Setting Tab and Indent Preferences

Select **File > Preferences > Editor/Debugger > Tab** to specify the preferences that follow.

Tabs and Indents

Tab size. Specify the amount of space inserted when you press the **Tab** key. When you change the **Tab size**, it changes the tab size for existing lines in that file, unless the **Tab key inserts spaces** also is selected.

Tab key inserts spaces. Select this item if you want a series of spaces to be inserted when you press the **Tab** key. If the item is not selected, a tab acts as one space whose length is determined by **Tab size**.

Indent size. Specify the indent size for smart indenting.

Emacs-style Tab key smart indenting. This indenting convention is based on the style used by the Emacs editor and is similar to the **Enable smart indenting** preference. With this preference selected, lines are indented according to smart indenting preferences when you position the cursor in a line or select a group of lines, and then press the **Tab** key. Then, you cannot use tabs within a line.

See Also

Information about additional preferences for indenting is provided in these sections:

- "Setting Language Preferences" on page 8-20 for additional indenting preferences
- "Setting Keyboard Preferences for Desktop Tools" on page 2-138"Setting Keyboard Preferences for Desktop Tools" on page 2-138 for function hints, tab completion, and delimiter matching preferences
- "Specifying Options for MATLAB Using Preferences" on page 2-126

Setting Language Preferences

MATLAB applies language preferences based on the file name extension of the file open in the Editor.

To specify language preferences, such as syntax highlighting:

1. Select **File > Preferences > Editor/Debugger > Language**, for the type of file you are editing.
2. Select the **Language**.

   The preference dialog box displays the preferences for that language. If the language you are using is not an option, it means that setting language preferences for that language is not available.
3. Change preferences for the language you selected, and then click **Apply**.

For example, when you edit a file with an `.html` extension, the language preferences set for XML/HTML apply to that file. If you have a Java file open at the same time, the Java language preferences apply to the Java file.

File Extensions

To specify a file extension associated with the selected language, follow these steps:

1. Optionally, add or change the file extensions associated with the language in the **Language** field by using the **Add** and **Remove** buttons

   Changes you make to the file extensions do not apply to open files until you close and reopen them.

2 Select a file extension from the **File extensions** list.

- If a file has no extension (for example, a new untitled file), MATLAB treats it as an M-file.
- If a file has an extension, but it is not in the **File extensions** list, MATLAB does not apply any language preferences to that file.

3 Click **Apply**.

The following table presents the default file extensions for each language.

Language Preference	Default File Extensions
MATLAB (MATLAB M-file)	`.m` (uppercase or lowercase)
TLC	`.tlc`
VHDL	`.vhd`, `.vhdl`
Verilog	`.v`
C/C++	`.c`, `.cpp`, `.h`, `.hpp`
Java	`.java`
XML/HTML	`.xml`, `.xsl`, `.wsdl`, `.html`, `.htm`, `.shtml`

For details about preferences for each language, see

- "Setting MATLAB Language Preferences" on page 8-22
- "Setting TLC Language Preferences" on page 8-27
- "Setting VHDL Language Preferences" on page 8-28
- "Setting Verilog Language Preferences" on page 8-28
- "Setting C/C++ Language Preferences" on page 8-29

  "Setting Java Language Preferences" on page 8-31
- "Setting XML/HTML Language Preferences" on page 8-32

Setting MATLAB Language Preferences

Select **File > Preferences > Editor/Debugger**, and then from the **Languages**, choose **MATLAB** to specify these preferences for editing M-files:

- "Syntax" on page 8-22
- "Indenting" on page 8-22
- "Formatting Comments" on page 8-26
- "File extensions" on page 8-27

See also "Setting Preferences for M-Lint" on page 8-141.

Syntax

Enable syntax highlighting.

- Select to show colors that help you identify certain constructs, like matching `if/else` statements.

  Click **Set syntax colors** to open the **Colors** preference pane, and specify the colors that you want the Editor to use for syntax highlighting.
- Clear to show all text in black.

The syntax colors preferences apply to all tools that use syntax highlighting, including the Command Window, the Command History, the Editor and other tools. For a description of syntax highlighting, see "Setting Colors Preferences for Desktop Tools" on page 2-150.

Indenting

You can set preferences to have MATLAB automatically apply indenting to your M-file code. The indenting options you specify, however, apply only to lines you enter after changing the preference; they do not affect indenting for existing lines. For information on changing the indenting for existing lines, see "Manual Indenting" on page 8-53.

Enable Smart Indenting. Select this option to specify that you want:

- The body of loops to be automatically indented within the start and end of the loop statement.
- The lines you indent using tabs or spaces to result in subsequent lines automatically aligning with the indented line.
- Functions to be automatically indented as specified with the **Function indenting format** option.

After entering a new line, use **Text > Decrease Indent** to move text back to a previous indent level, if you want.

Although you can manually insert tabs at the start of a line, be aware that smart indenting may not work properly if you do. To correct selected lines, choose **Text > Smart Indent**.

Deselect **Enable smart indenting** to specify that you want:

- Lines to be aligned on the left by default
- To specify indenting manually using the tab and space keys
- Functions to not be automatically indented as specified by the **Function indenting format** option

When you deselect **Enable smart indenting** you can still indent functions, as specified by the **Function indenting format** option by selecting the functions in the Editor, and then choosing **Text > Smart Indent**.

To set the indent size for smart indenting and the tab size for manual indenting, see “Setting Tab and Indent Preferences” on page 8-19.

Example of Smart Indenting.

```
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
```

Created with Smart indenting selected. Did not insert tabs manually. Indenting was inserted automatically as text was entered.

Example of Smart Indenting Deselected with Manual Tabs.

```
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
```

Created with Smart indenting deselected. Indentation created by manually inserting tab before each indented line

Example of Smart Indenting Deselected Without Tabs.

```
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
```

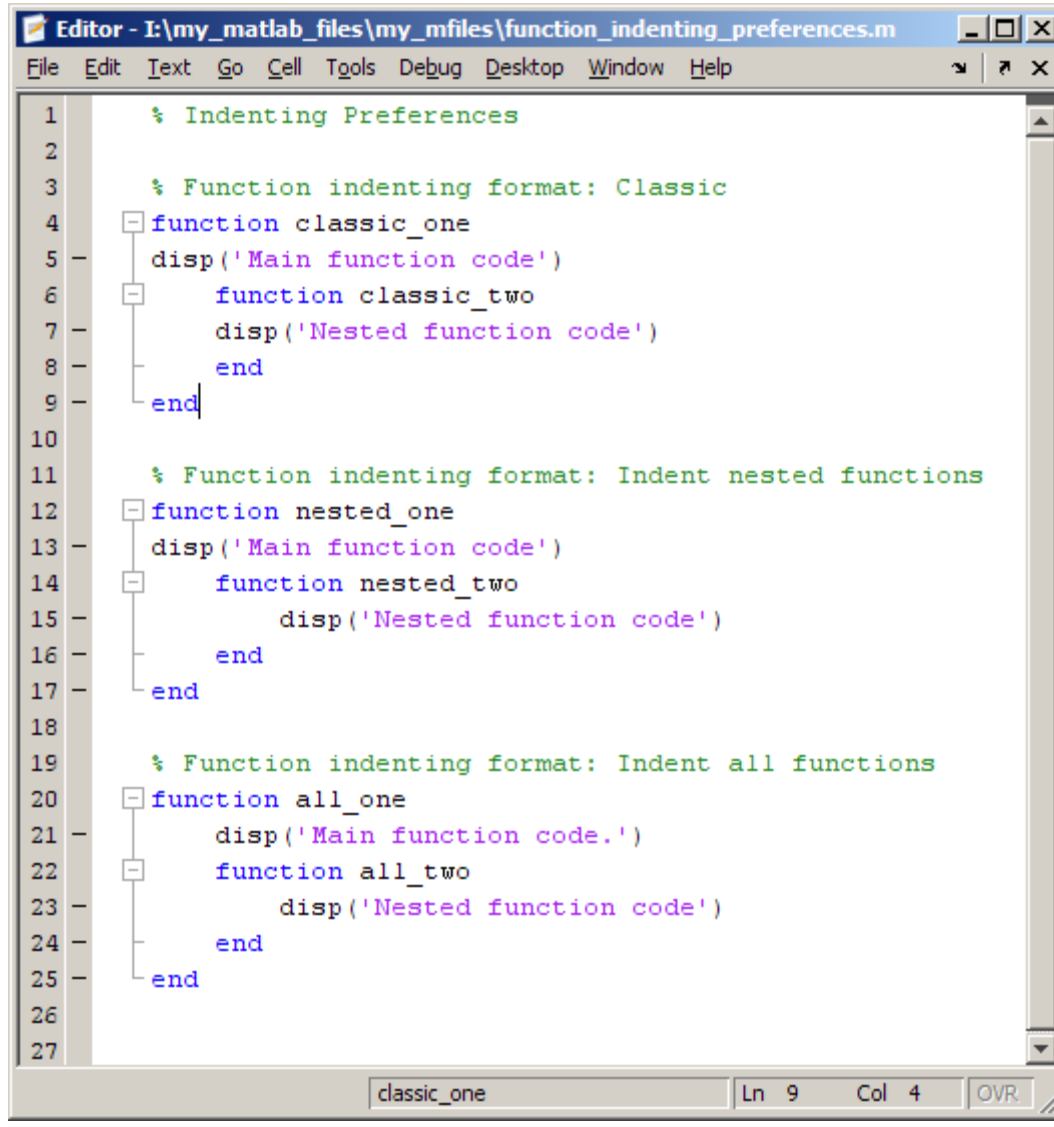
Created with Smart indenting disabled and no tabs manually inserted.

Function Indenting Format. Specify how functions indent in the Editor, as follows:

1. Select or deselect **Enable smart indenting**, depending on whether you want your function indenting choice applied automatically, as you type code in the Editor. See "Enable Smart Indenting" on page 8-22 for details.
2. Choose a function indenting format. The styles for indenting functions are:
   - **Classic** — The Editor aligns the function code with the function declaration.
   - **Indent nested functions** — The Editor indents the function code within a nested function.
   - **Indent all functions** — The Editor indents the function code for both main and nested functions.
3. For nested functions, provide an `end` statement at the start of a line for each function declaration.

   An `end` statement aligns with its function declaration only when the `end` statement appears at the start of the line.

Examples of Function Indenting Format Preference. The following image illustrates each of the function indenting formats.



Formatting Comments

Specify the **Max width**, that is, the maximum width, in number of columns, for M-file comments when you select the **Autowrap comments** preference.

The maximum width also applies when you use the **Wrap Selected Comments** feature. See "Formatting Comments in M-Files" on page 8-44 for details.

For example, assume you select **Autowrap comments** and set the maximum width to be 75 characters, which is the width that fits on a printed page using the default font for the Editor. When typing a comment line, as you reach the 75th column, the comment automatically continues on the next line.

See also the "Right-Hand Text Limit" on page 8-17 preference for the Editor/Debugger.

File extensions

See "File Extensions" on page 8-20 for general Language preferences.

Setting TLC Language Preferences

Target Language Compiler (TLC) is an integral part of Real-Time Workshop®. Select **File > Preferences > Editor/Debugger > Language > TLC** to specify these preferences for editing TLC files:

- "Syntax highlighting" on page 8-27
- "File Extensions" on page 8-28

See also "Setting Preferences for M-Lint" on page 8-141.

Syntax highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify TLC constructs, such as commands and macros. Optionally, change the colors used for these elements.

Commands. Color for TLC commands, such as `LibBlockMatrixParameter`.

Comments. Color for the comment indicator, `%%` or `/%...%/`, and its associated text.

C Strings. Color for C strings.

Macros. Color for TLC macros.

File Extensions

See “File Extensions” on page 8-20 for general Language preferences.

Setting VHDL Language Preferences

Select **File > Preferences > Editor/Debugger > Language** to specify these preferences for editing VHDL files:

Syntax highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify certain VHDL constructs, such as keywords and operators. Specify the colors used for these elements.

Keywords. Color for VHDL keywords, such as `SIGNAL`.

Comments. Color for the comment indicator, `--` , and its associated text.

Operators. Color for operators, such as `<=`.

Strings. Color for strings, such as `"0000"`.

File extensions

See “File Extensions” on page 8-20 for general Language preferences.

Setting Verilog Language Preferences

Select **File > Preferences > Editor/Debugger > Language** to specify these preferences for editing Verilog files:

Syntax highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify certain Verilog constructs, such as keywords and operators. Specify the colors used for these elements.

Keywords. Color for Verilog keywords, such as `assign`.

Comments. Color for the comment indicator, `//` , and its associated text.

Operators. Color for operators, such as `=`.

Strings. Color for strings, such as `"Test Completed'`.

File extensions

See “File Extensions” on page 8-20 for general Language preferences.

Setting C/C++ Language Preferences

Select **File > Preferences > Editor/Debugger > Language** to specify these preferences for editing C or C++ language files:

Syntax highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify certain C constructs, such as methods. Specify the colors used for these elements.

Show methods. Text style for methods to appear in when you type them: `Bold`, `Italic`, or `Plain` (no special highlighting).

Keywords. Color for keywords, such as `if`.

Strings. Color for terms enclosed in double quotation marks, for example, `"default"`.

Characters. Color for terms enclosed in single quotation marks, for example, `'a'`.

Comments. Color for text following the comment indicator, `//`, as well as for the block comment indicators, `/*` and `*/`, and the code in between.

Preprocessor. Color for text following the preprocessor symbol, `#`.

Bad characters. Color for illegal characters.

Indenting

You can set preferences to specify if you want MATLAB to automatically apply indenting to your C/C++ files. The indenting options you specify, however, apply only to lines you enter after changing the preference; they do not affect the indenting for existing lines. For information on changing the indenting for existing lines, see "Manual Indenting" on page 8-53.

Enable Smart Indenting. Select this option to specify that you want:

- The body of loops to be automatically indented within the start and end of the loop statement.
- The lines you indent using tabs or spaces to result in subsequent lines automatically aligning with the indented line.

After entering a new line, use **Shift+Tab** to move text back to a previous indent level if you want.

Although you can manually insert tabs at the start of a line, be aware that smart indenting may not work properly. Use the **Text** menu entry for **Smart Indent** to correct selected lines.

Deselect **Enable smart indenting** to specify that you want:

- Lines to be aligned on the left by default
- To specify indenting manually using the tab and space keys

To set the indent size for smart indenting and the tab size for manual indenting, see "Setting Tab and Indent Preferences" on page 8-19.

For examples see:

- "Example of Smart Indenting" on page 8-24
- "Example of Smart Indenting Deselected with Manual Tabs" on page 8-24
- "Example of Smart Indenting Deselected Without Tabs" on page 8-24

File extensions

See "File Extensions" on page 8-20 for general Language preferences.

Setting Java Language Preferences

Select **File > Preferences > Editor/Debugger > Language > Java** to specify these preferences for editing Java files:

- "Syntax highlighting" on page 8-31
- "Indenting" on page 8-32
- "File extensions" on page 8-32

See also "Setting Keyboard Preferences for Desktop Tools" on page 2-138 for the Editor/Debugger and Command Window and "Setting Language Preferences" on page 8-20.

Syntax highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify certain Java constructs, such as methods. Specify the colors used for these elements.

Show methods. Text style for methods to appear in when you type them: `Bold`, `Italic`, or `Plain` (no special highlighting).

Keywords. Color for keywords, such as `if`.

Strings. Color for terms enclosed in double quotation marks, for example, `"alive"`.

Characters. Color for terms enclosed in single quotation marks, for example, `'a'`.

Comments. Color for text following the comment indicator, `//`, as well as for the block comment indicators, `/*` and `*/`, and the code in between.

Bad characters. Color for illegal characters.

Indenting

Select or deselect **Enable smart editing** to specify whether you want the Editor to apply indenting to your Java files when you press the **Enter** key to type in a new line. The effects are the same as those for C/C++ files. For details and examples, see the C/C++ preference for indenting.

File extensions

See “File Extensions” on page 8-20 for general Language preferences.

Setting XML/HTML Language Preferences

Select **File > Preferences > Editor/Debugger > Language > XML/HTML** to specify these preferences for editing XML, XSL, WSDL, HTML, HTM, and SHTML files:

Syntax highlighting

Select the **Enable syntax highlighting** check box to show colors that help you identify XML, WSDL, and HTML constructs, such as elements and tags. Optionally, change the colors used for these items.

Attribute Name. Color for attribute names.

Attribute Value. Color for values, such as the source for an image, for example `"myimage.gif"`, in `<img src= myimage.gif >`.

CDATA Section. Color for a CDATA section in XML files.

Character. Color for characters and entities, such as the nonbreaking space, ` `.

Comment. Color for text contained within comment indicators, `<!--` and `-->`.

DOCTYPE Declaration. Color for DOCTYPE declarations.

Error. Color for invalid entries, such as `<+1>` for font size, which is deprecated in favor of style sheets in the HTML 4.01 specification.

See the W3C Web site for details.

Operator. Color for operators, such as the equal sign (=).

Processing Instruction. Color of the instructions to the application that processes the XML file.

Tag. Color for tags and elements, such as `<font>` or `<img>`.

Indenting

Select or deselect **Enable smart editing** to specify whether you want the Editor to apply indenting to your files when you press the **Enter** key to type in a new line. The effects are the same as those for C/C++ files. For details and examples, see the C/C++ preference for indenting.

File extensions

See “File Extensions” on page 8-20 for general Language preferences.

Setting Code Folding Preferences

Select **File > Preferences > Editor/Debugger > Code Folding** to set these preferences for hiding and revealing code in M-files, including function and class code, function and class help code, programming control blocks, and so on.

- "Enable Code Folding by Programming Construct" on page 8-34
- "Fold Initially" on page 8-34

See also "Appearance of an M-File — Making Files More Readable" on page 8-52, and Working with M-Files.

Enable Code Folding for M-Files

Select this option to enable code folding in M-files; clear this option to disable code folding in M-files. (Use the table that appears below this option to enable and disable code folding for selected programming constructs within an M-file.)

This option has the following effects:

- If you select this option, code folding is enabled for all programming constructs that you enable in the table that appears below this option in the Preferences dialog box.
- If you clear this option, code folding is disabled for all programming constructs, regardless of the programming constructs that you may have previously enabled in the table that appears below this option in the Preferences dialog box.

Enable Code Folding by Programming Construct

If you select the **Enable code folding for M-files** option, you can independently enable or disable code folding for individual programming constructs (function code, function help, programming control blocks, class code, and so on) by selecting or clearing the **Enable** check box that corresponds to each construct in the table on the Preferences dialog box. By default, code folding is enabled for all programming constructs except if/else blocks and switch/case blocks.

Fold Initially

If you enable code folding for a programming construct, you can specify how the Editor displays that construct the first time that you open an M-file that existed prior to MATLAB version 7.5. Select or clear the **Fold Initially** check box associated with a construct to direct the Editor to collapse or expand the associated construct, respectively, the first time you open an M-file that

existed prior to MATLAB version 7.5 (R2007b) using MATLAB version 7.5 or later.

Setting Autosave Preferences

Select **File > Preferences > Editor/Debugger > Autosave** to specify these preferences for the Editor's autosave feature.

Enable autosave in the MATLAB Editor

The MATLAB Editor automatically saves a copy of the current version of the file you are editing. Clear this check box if you do not want the MATLAB Editor to save the copy automatically.

Save Options

Save every n minutes. Specify how often you want the Editor to save, automatically, a copy of the file you are editing.

Save untitled files. Select this check box if you want the Editor to save automatically a copy of new files that you have not yet saved, which are therefore untitled. If selected, the first autosave file is `Untitled.asv`. If the folder already contains a file named `Untitled.asv`, the autosave file is `Untitled2.asv`, and so on, for additional unnamed files.

If the autosave feature creates `Untitled.asv` and you later save the file as `filename.m`, the next autosave version is `filename.asv`. `Untitled.asv` remains until you delete it.

Close Options

Automatically delete autosave files. With this preference selected, MATLAB deletes the autosave file when you close the source file in the Editor.

Filename

Specify the extension used for autosave files. The default setting for Microsoft Windows platforms is the extension `.asv` (for *a*uto*sav*e), making the autosave

file `filename.asv`. For Microsoft Windows and UNIX platforms, you can select **Replace extension with** and specify any extension.

For UNIX platforms, the default is **Append file with** the tilde (~) character, making the autosave file `filename.m~`. For Windows and UNIX platforms, you can select **Append file with** and specify a different string to append to the file.

Location

Specify the full path for the folder where you want autosave files stored. You can specify a single folder for all autosave files, such as a folder you create called `autosave_files`. For the Editor to create an autosave file, you must have write-permission for the specified location.

If you do not specify a location, MATLAB:

- Stores the autosave file for each named file in the same folder as the source file, that is, the file you are editing.
- Stores the autosave file for each untitled file in the folder that was the current folder when you opened the file.

Additional Information about Editor/Debugger Preferences

Additional information that relates to the Editor and preferences includes:

- "Setting Autosave Preferences" on page 8-35
- "Setting Fonts Preferences for Desktop Tools" on page 2-141
- "Setting Colors Preferences for Desktop Tools" on page 2-150
- "Specifying Options for MATLAB Using Preferences" on page 2-126
- "Setting Keyboard Preferences for Desktop Tools" on page 2-138

Entering Statements in the Editor

In this section...

Using Command Window Features in the Editor

After opening an existing file or creating a new file in the Editor, enter statements in the file. Use the same practices as for entering statements in the Command Window as described in Chapter 3, "Running Functions — Command Window and History":

- "Case and Space Sensitivity" on page 3-14
- "Entering Multiple Lines Without Running Them" on page 3-16
- "Entering Multiple Functions in a Line" on page 3-17
- "Entering Multiple-Line (Long) Statements — Line Continuation" on page 3-17
- "Suppressing Output" on page 3-44
- "Formatting and Spacing Numeric Output" on page 3-45
- "Matching Delimiters (Parentheses)" on page 3-21
- "Viewing Function Syntax Hints While Entering a Statement" on page 3-30
- "Getting Help for a Function Shown in the Command Window or Editor" on page 3-35
- "Finding Functions Using the Function Browser" on page 3-37

Entering Text in Insert or Overwrite Mode

On Windows and UNIX platforms, the Editor enables you to enter text in either insert or overwrite mode. By default, when you enter text in the Editor, you use *insert mode*. In this mode, the Editor inserts the text you type within the existing text. In *overwrite mode*, the text you type overwrites the existing text.

Note The Macintosh platform does not support overwrite mode.

Determining the Current Typing Mode

The Editor indicates the current mode as follows:

- In insert mode, the cursor is a vertical bar and the text **OVR** appears dimmed in the status bar.
- In overwrite mode, the cursor is a wide block and the text **OVR** is not dimmed in the status bar.

Toggling Between Insert and Overwrite Mode

To toggle between insert and overwrite mode, follow these steps:

1. In the Editor, place the cursor where you want to enter text.
2. Press the **Insert** key to toggle the typing mode from insert to overwrite mode, or vice versa.

The **Insert** key is the default keyboard shortcut for changing the typing mode. For details, on changing keyboard shortcuts, see “Customizing Keyboard Shortcuts” on page 2-78.

Changing the Case of Selected Text

To change the case of text in the Editor, select the text. 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 the MATLAB Editor or Command Window. 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')
```

Undoing and Redoing Editor Actions

You can undo many of the Editor 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 file name. 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" on page 8-40

Commenting in M-Files Using the MATLAB Editor

You can comment the current line or a selection of lines in an M-file:

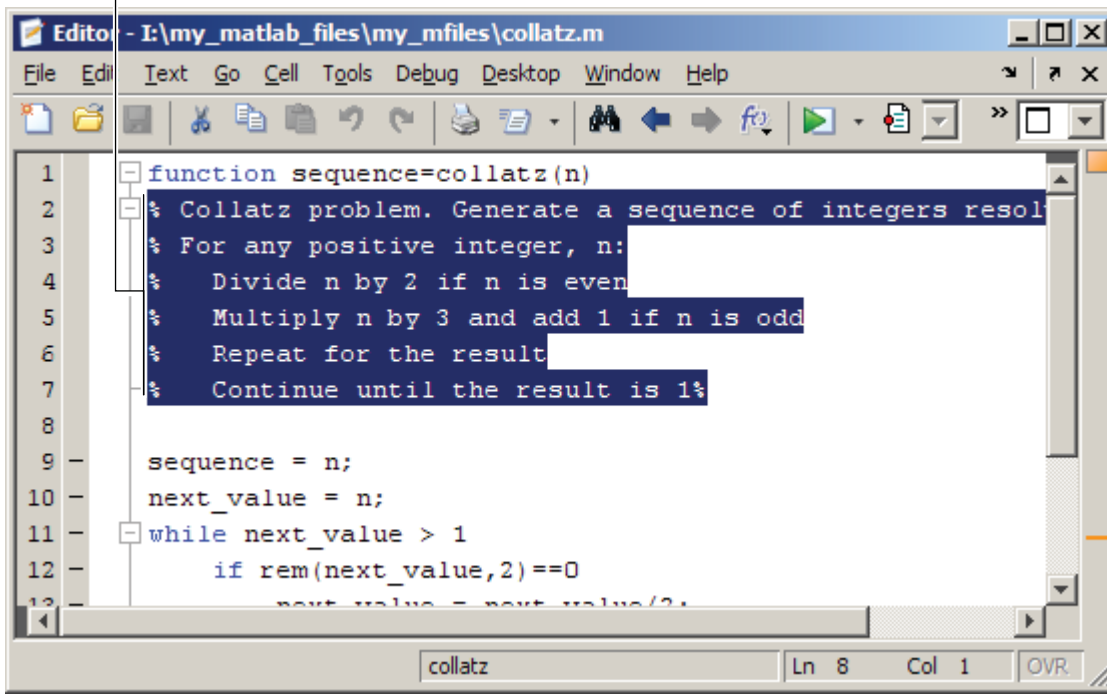
1. For a single line, position the cursor in that line. For multiple lines, click in the line and then drag or **Shift**+click to select multiple lines.
2. Select **Comment** from the **Text** menu, or right-click and select it from the context menu.

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 8-52.

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.

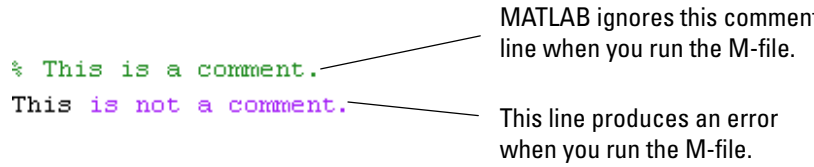


Commenting in Java and C/C++ Files Using the MATLAB Editor

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

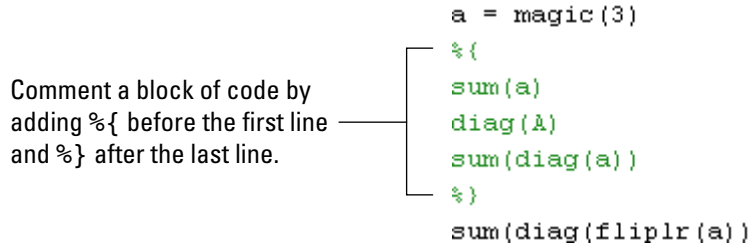
You can make any line in an M-file a comment by typing `%` at the beginning of the line. To put a comment within a line, type `%` followed by the comment text; MATLAB software treats all the information after the `%` on a line as a comment.



To uncomment any line, delete the comment symbol, %.

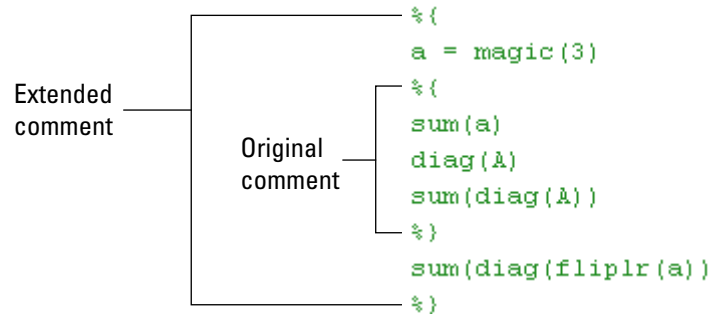
To comment a contiguous group of lines, type %{ before the first line and %} after the last line you want to comment. This is referred to as a block comment. The lines that contain %{ and %} can contain spaces, but not contain any other text. After typing the opening block comment symbol, %{, all subsequent 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.



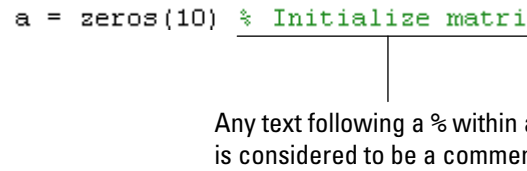
You can easily extend a block comment without losing the original block comment, that is, create a nested block comment, as shown in the following example.

Create a nested comment, that is, a block comment within a block comment.



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 software ignores any text on the line after the `%`.



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. 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. When typing a comment line, as you reach the 75th column, the comment automatically continues on the next line.

Completing Statements in the Editor — Tab Completion

The Editor 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 folder
- Class folders (including @-folders) and package folders
- 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
- MATLAB objects

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 selected. For details, see "Setting Keyboard Preferences for Desktop Tools" on page 2-138.

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 following, whereas the Editor tab completion does not:

- Completion of file names
- Help browsing

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

This example illustrates a basic use for tab completion in the Editor. In an M-file opened in the Editor, type the beginning of a function or model on the MATLAB search path or in the current folder, for example,

```
horz
```

and press **Tab**. The Editor 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 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 also moves the cursor to the right when you try to complete a file name; file name tab completion is not supported in the Editor, 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 completes `horzcat`.

The Editor 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 completes the variable name

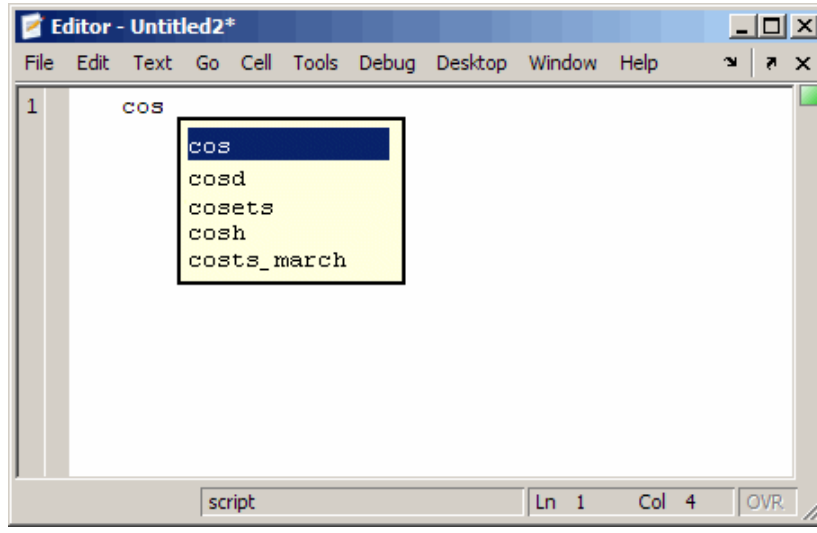
`costs_march`. If the Editor displays `No Completions Found`, `costs_march` does not exist in the current workspace.

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 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, type

```
cos
```

and press **Tab**. The Editor 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.

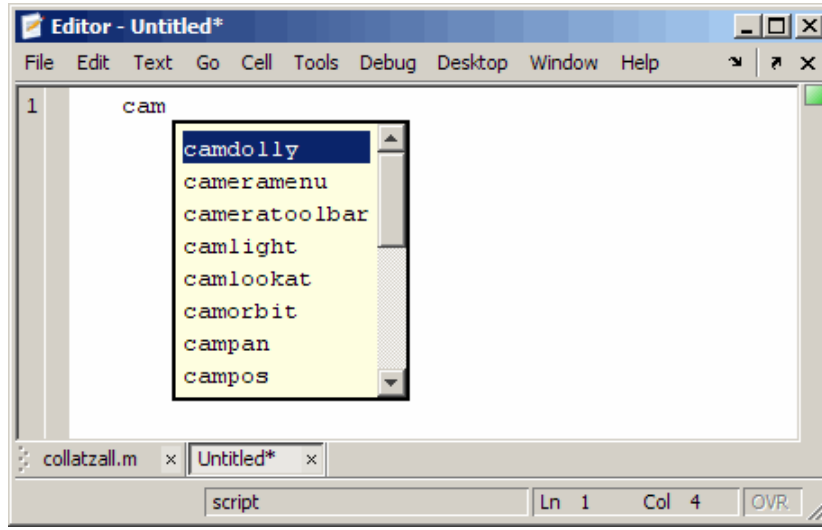
Continue typing to make your entry unique. For example, type the next character, such as `t` in the example. The Editor selects the first item in the list that matches what you typed, in this case, `costs_march`. Press **Enter** (or

Return) or **Tab** to select that item, which completes the name in the M-file. In the example, the Editor displays `costs_march` at the prompt.

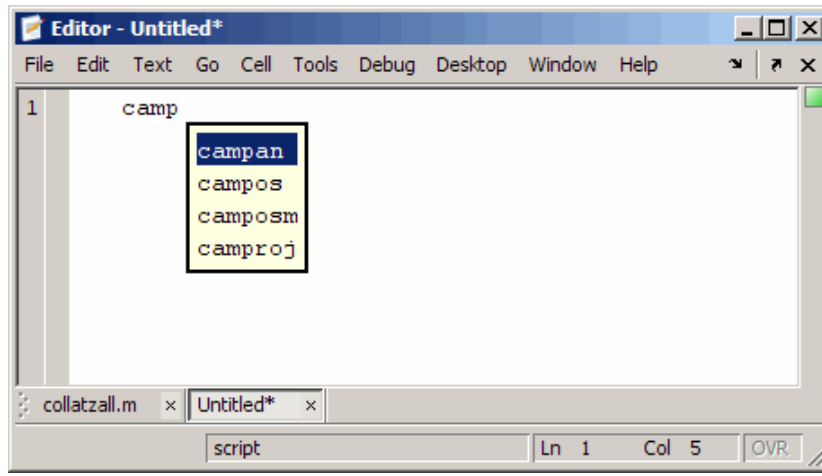
You can navigate the list of possible completions using up and down arrow keys, and **Page Up** and **Page Down** keys. You can clear the list without selecting anything by pressing **Esc**. The list of possible completions might include items that are not valid commands, such as private functions.

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 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
```

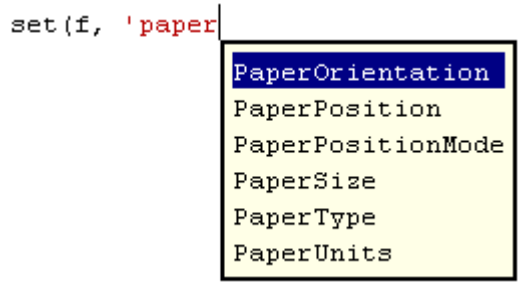
and press **Tab**, which completes the entry `mystruct.name`, where `mystruct` is in the current workspace and contains no other fields that begin with `n`.

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 displays



Select a property from the list. For example, type

```
u
```

and press **Enter**. The Editor 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 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.

Appearance of an M-File — Making Files More Readable

In this section...

Note You can specify the default behaviors for some of these features—see "Specifying Options for MATLAB Using Preferences" on page 2-126.

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. For more information, see the Command Window documentation for "Highlighting Syntax to Help Ensure Correct Entries" on page 3-20.

When you paste or drag a selection from the Editor to another application, such as Microsoft Word, the pasted text maintains the syntax highlighting colors and font characteristics from the Editor. MATLAB software pastes the selection to the clipboard in RTF format, which many Microsoft 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, `MATLAB`. Under **Indenting**, select **Enable smart indenting**, and then click **OK**. Deselect **Enable smart indenting** instead if you want to indent manually. For more information about indenting preferences, click **Help** in the Preferences dialog box. Specify the indenting size and other options by selecting **File > Preferences > Editor/Debugger > Tab**.

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 read 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, select **File > Preferences > Editor/Debugger > Tab** and then click **Help**.

Function Indenting

You can select from three indenting options when you enter a subfunction or a nested function (a function within a function) in the Editor. For details, see "Function Indenting Format" on page 8-25.

Line and Column Numbers

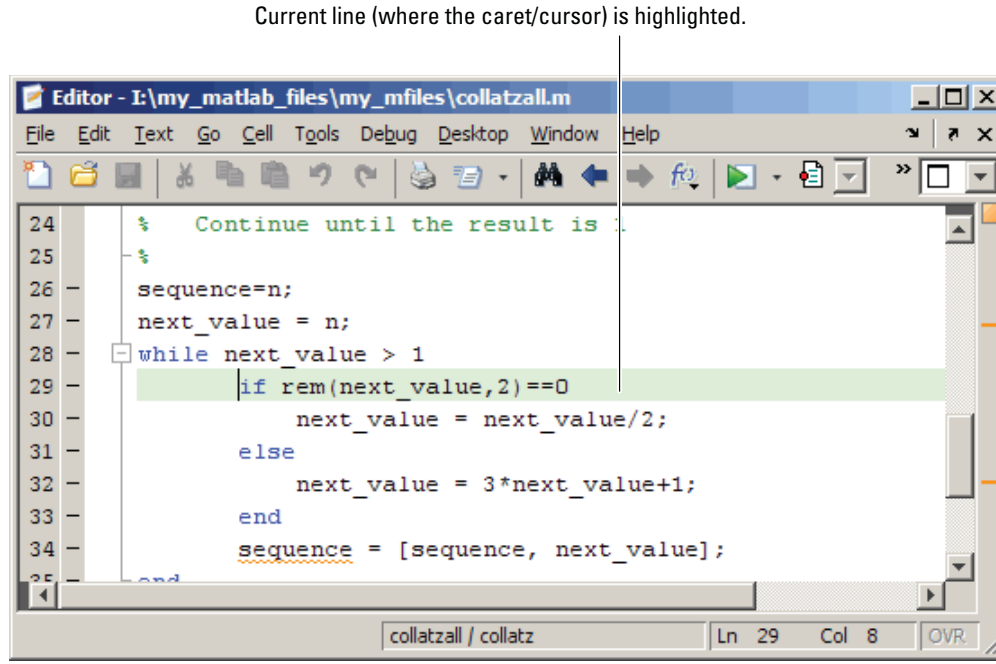
Line numbers display along the left side of the Editor window. You can elect not to show the line numbers using preferences. For details, select **File > Preferences > Editor/Debugger > Display**, and then click **Help**.

The line and column numbers for the current cursor position are shown in the far right side of the status bar in the Editor.

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.



Right-Hand Text Limit

By default, a light gray vertical line (rule) appears at column 75 in the Editor, indicating where a line becomes wider than desired. This is useful, for example, if you need to keep each line below a limit imposed by another text editor in which you intend to view the code. You can change the width and color of the vertical line, as well as the column number at which the vertical line appears. If you want, you can hide the vertical line. For more information, select **File > Preferences > Editor/Debugger > Display**, and then click **Help**.

Note This limit is a visual cue only and does not prevent text from exceeding the limit. For information on setting a value to automatically wrap comment text at a specified column number, see "Formatting Comments in M-Files" on page 8-44.

Class, Function, or Subfunction

The right side of the Editor status bar shows the class, function, or subfunction where the cursor is currently placed, depending on the type of file you are viewing, as follows:

- Class file — The name of the class followed by the name of the current function (if any) that the cursor is within. This is true regardless of the type of function in which the cursor is placed (nested, in a methods block, outside a classdef file, and so on).
- Function file — The name of the main function followed by the name of the current function the cursor is within (if any). This is true regardless of the type of function in which the cursor is placed (subfunction or nested).

Code Folding — Expanding and Collapsing M-File Constructs

Code folding is the ability to expand and collapse certain M-file programming constructs. This improves readability when an M-file contains numerous subfunctions or other blocks of code that you want to hide when you are not currently working with that part of the file.

You can set preferences to enable or disable the ability to expand and collapse the following M-file programming constructs:

- Help block comments
- Cells used for rapid code iteration and publishing
- Class code
- Class enumeration blocks
- Class event blocks
- Class method blocks
- Class properties blocks
- For and parfor blocks
- Function and class help
- Function code

- If/else blocks
- Single program, multiple data (spmd) blocks
- Switch/case blocks
- Try/catch blocks
- While blocks

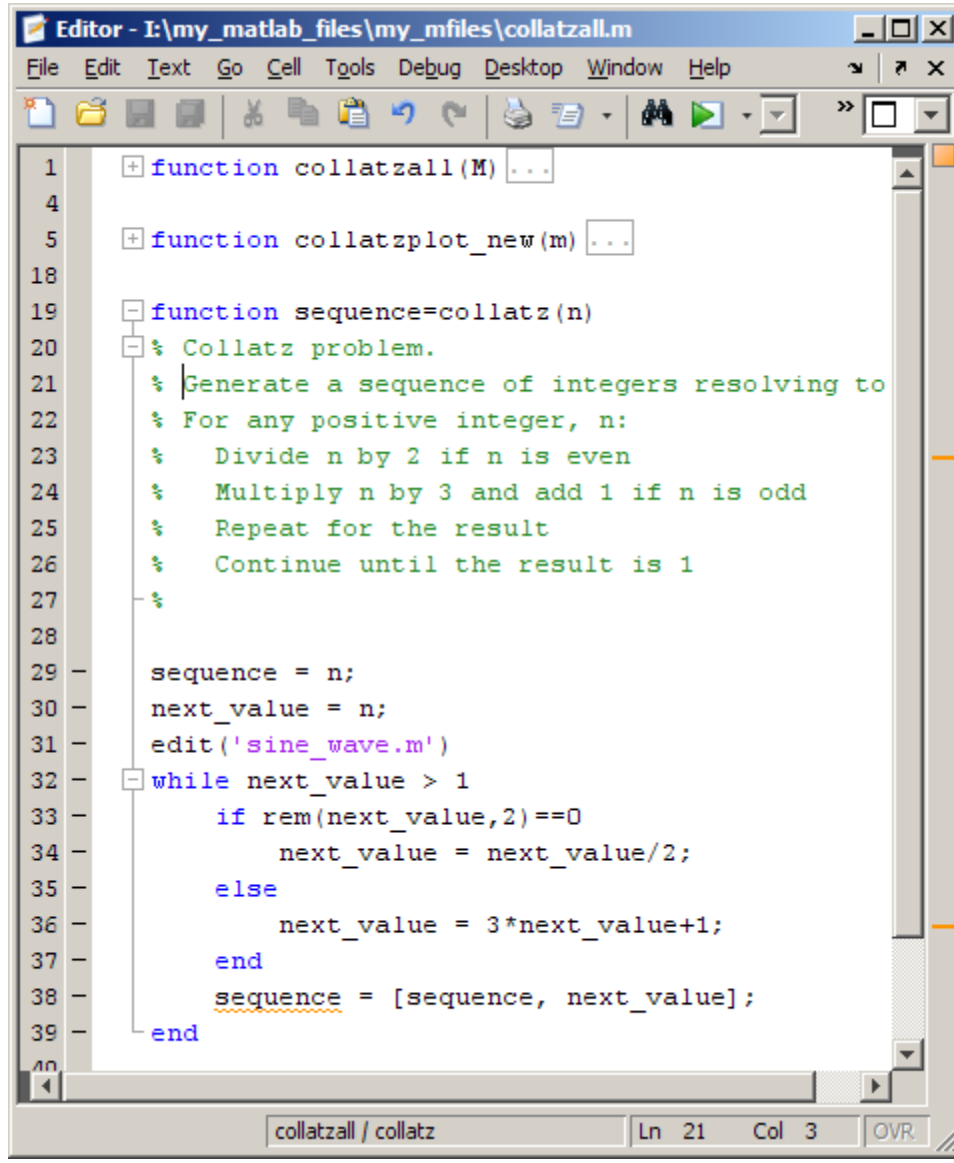
By default, code folding is enabled for all programming constructs except if/else blocks and switch/case blocks. Select **File > Preferences > Editor/Debugger > Code Folding**, and then click **Help** for details on setting preferences.

When you fold a construct, all the code associated with that construct is collapsed such that the Editor displays only the first line of the construct prepended by the plus sign (+) and appended with an ellipsis (...) to indicate there is more code. When you expand a construct, all the code associated with that construct appears and the first line of the construct is prepended with a minus sign (−).

To open the code used in the images in this section, enter the following in the Command window:

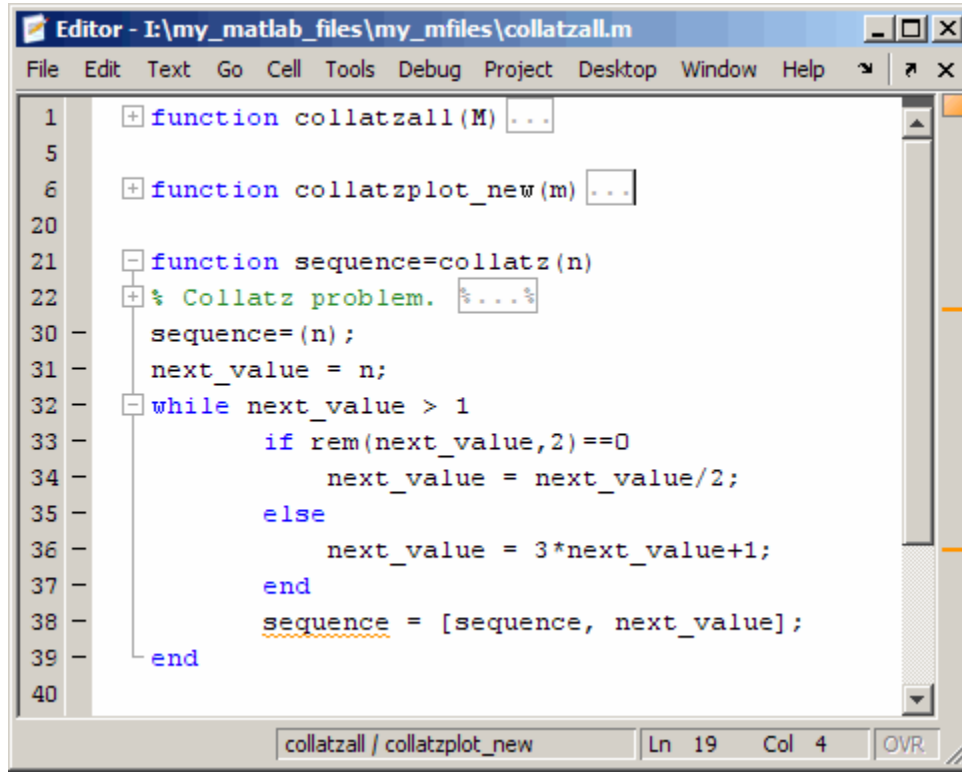
```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...
'examples','collatzall.m'))
```

The following image shows the `collatzall` and `collatzplot_new` functions collapsed and the `collatz` function code expanded.



When you expand a function or class, but collapse its associated help block code, the Editor displays all the function or class code and just the H1 line of

the help code. The H1 line ends with a commented ellipsis `%...%` to indicate there is additional help code, as shown in the following image.



To expand code for a construct that is currently collapsed, do one of the following:

- Click the plus sign ⊞ 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 ⊟ 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.

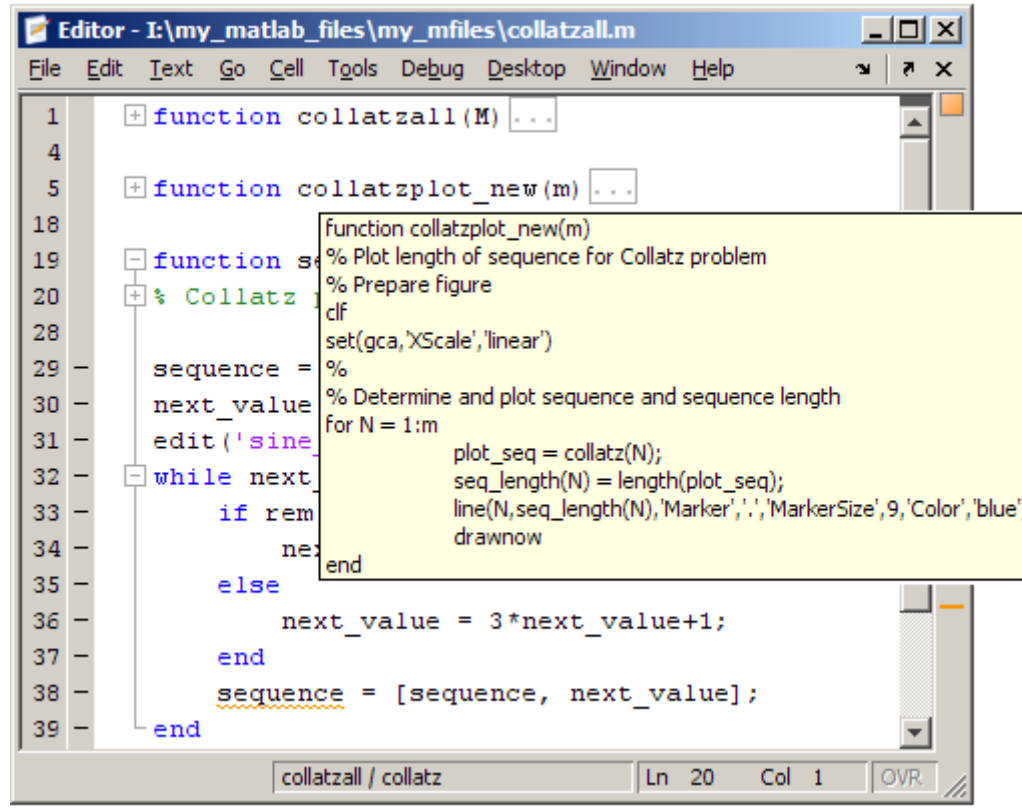
To expand or collapse all of the code in an M-file, place your cursor anywhere within the M-file, right-click, and then select **Code Folding > Expand All** or **Code Folding > Collapse All** from the context menu.

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 Programming Fundamentals documentation.

Viewing Folded Code in a Tooltip

You can view code that is currently folded by positioning the pointer over its ellipsis  . The code appears in a Tooltip. This lets you quickly view the code without unfolding it.

The following image shows the Tooltip that appears when you place the pointer over the ellipsis on line 6 of `collatzall.m` when the `collatzplot_new` function is folded.

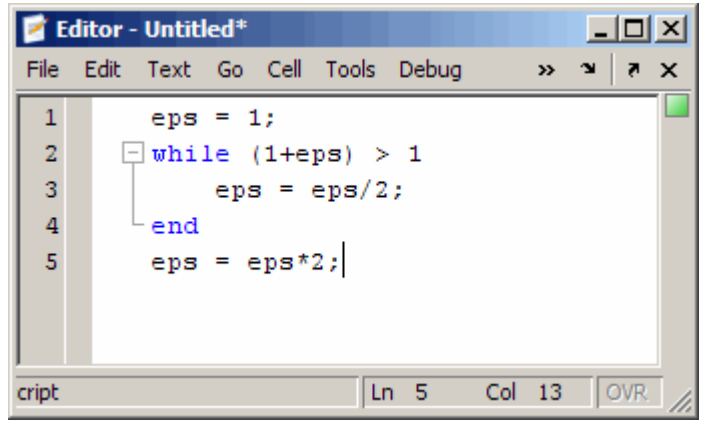


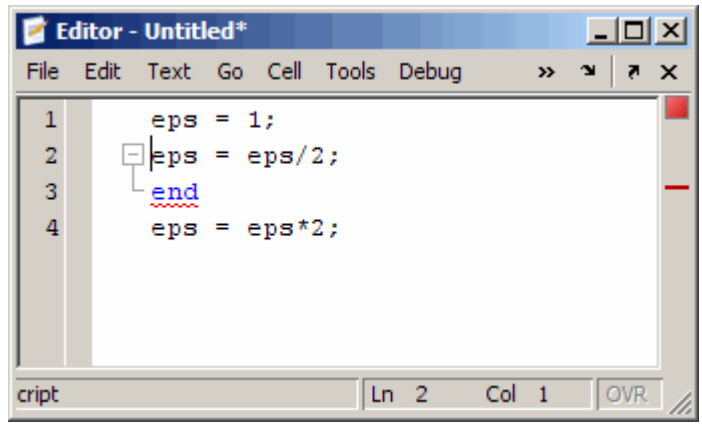
Code Folding Behavior and Preferences

Be aware of the following:

- You can change the current code folding settings, by selecting **File > Preferences > Editor/Debugger > Code Folding**. If needed, click **Help** for assistance.
- By default, the first time you open an M-file that existed before MATLAB Version 7.5 (R2007b) using MATLAB Version 7.5 (R2007b) or later, 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.
- If your code contains syntax errors, the code folding indicators may appear to be placed in the wrong location. For example, suppose your code currently appears as shown in the first figure that follows. If you delete the `while` statement, it introduces a syntax error at line 3, as shown in the second figure that follows. Notice that the minus sign remains in the same location it held for the syntactically correct code. After you correct the syntax error, the Editor adjusts and displays the code folding indicators appropriately.



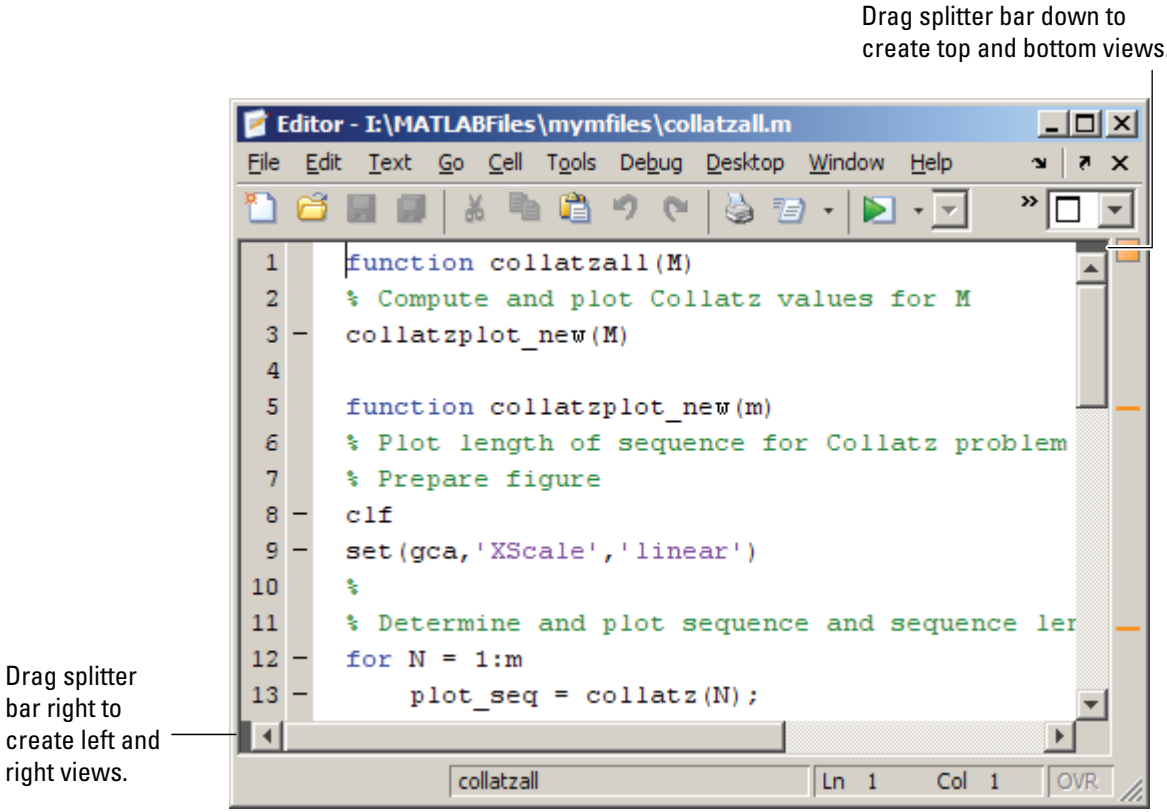


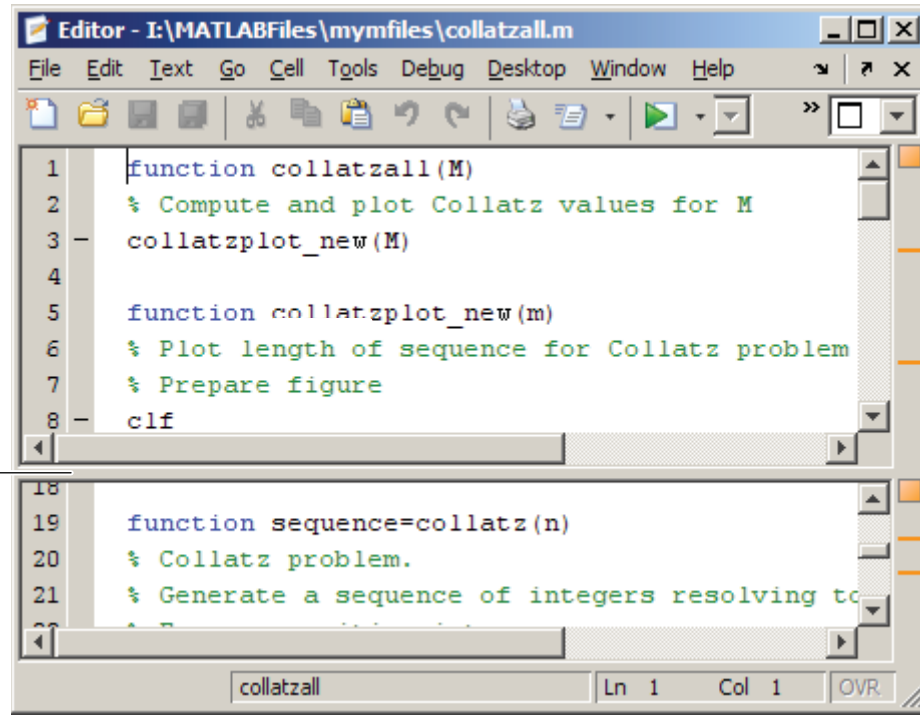
Split Screen Display

You can simultaneously display two different parts of a file in the Editor. 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**.

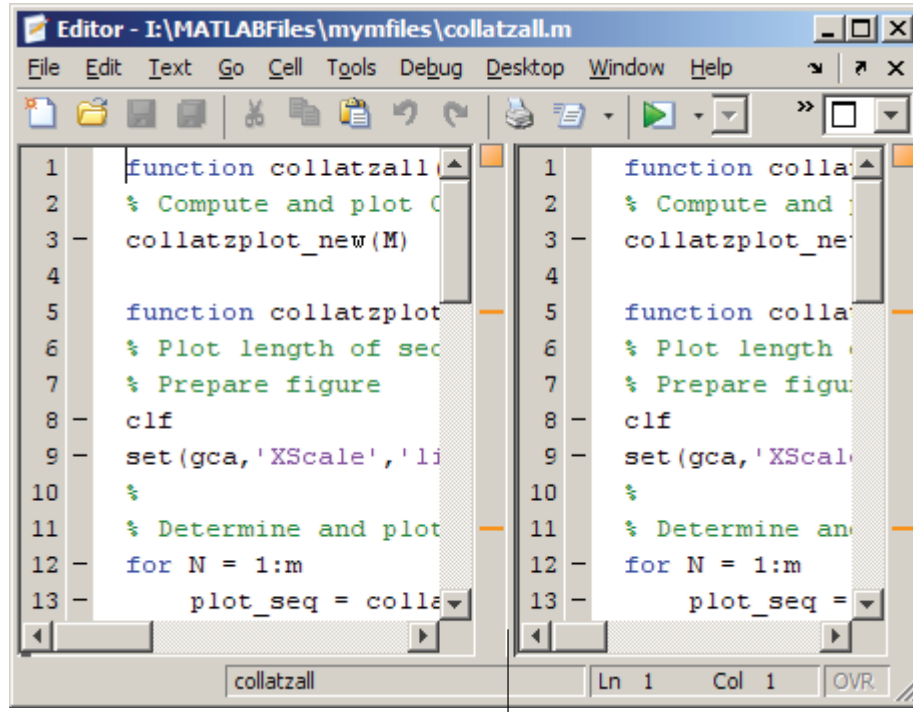
Alternatively, when there is a scroll bar, split the document into top and bottom views by dragging the splitter bar, as shown in the following illustration, down from above the vertical scroll bar. Similarly, to split into left and right views, drag the splitter bar from the left of the horizontal scroll bar. The mouse pointer assumes a double-headed arrow shape when you position it on the splitter bar.


Drag splitter bar down to create top and bottom views.
Drag splitter bar right to create left and right views.
Editor - I:\MATLABFiles\mymfiles\collatzall.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1 function collatzall(M)
2 % Compute and plot Collatz values for M
3 collatzplot_new(M)
4
5 function collatzplot_new(m)
6 % Plot length of sequence for Collatz problem
7 % Prepare figure
8 clf
9 set(gca,'XScale','linear')
10 %
11 % Determine and plot sequence and sequence ler
12 for N = 1:m
13 plot_seq = collatz(N);
collatzall
Ln 1 Col 1
OVR



Double-click the splitter to remove the split.

Drag the splitter to resize the views.



Double-click the splitter to remove the split.

Drag the splitter to resize the views.

Resize of the views by dragging the splitter. The mouse pointer assumes an arrow shape when you position it on the splitter.

Only one view is active at any time, meaning, you will see only the cursor in one of the views. To change the active view, select **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 views 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 "Positioning Documents" on page 2-26 for instructions on displaying multiple documents simultaneously.

Navigating in an M-File

In this section...

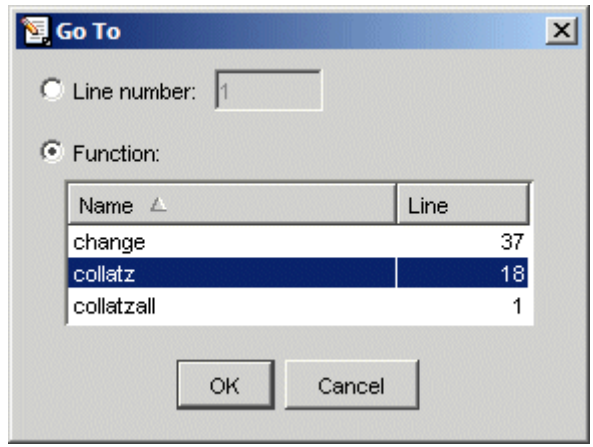
Note See also "Finding Text in Files" on page 8-75.

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. For both class and function files, the functions are listed in alphabetical order—except that in function files, the name of the main function always appears at the top of the list.

Going to a Cell

For M-file scripts that contain cells, 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 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
```

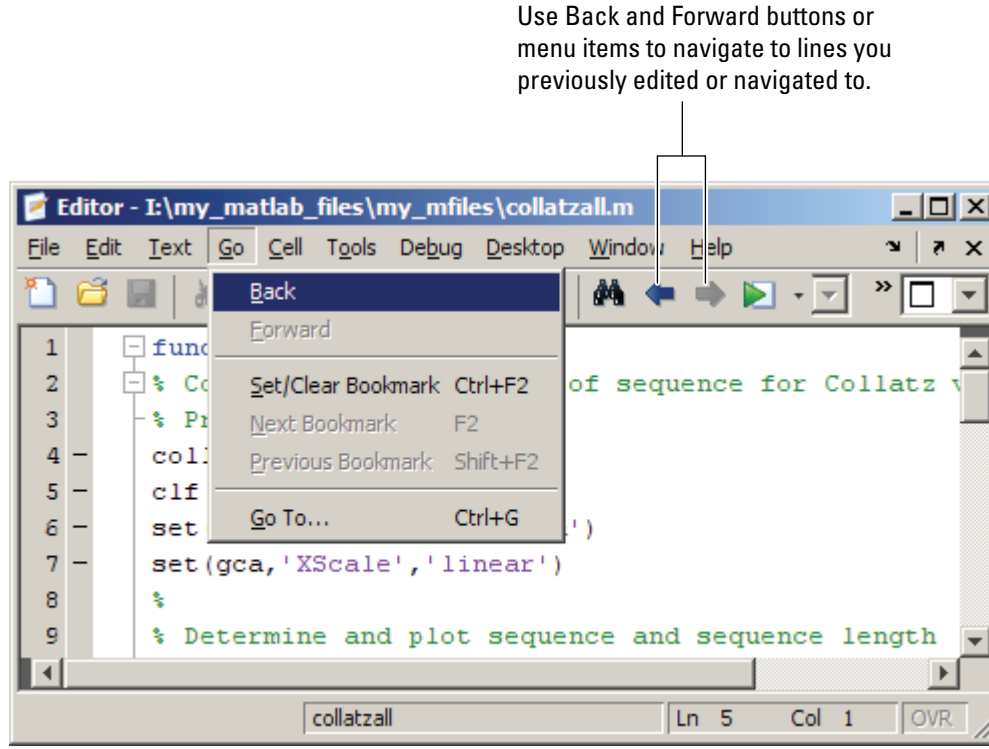
To go to a bookmark, select **Next Bookmark** or **Previous Bookmark** from the **Go** menu.

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.



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 to which you navigated using 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 Break** 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.

Feature	Examples	Notes
Find features	**Edit > Find and Replace**, **Find Next**, **Find Previous**, and **Find Selection**	None
Incremental search	**Ctrl+Shift+S** and **Ctrl+Shift+R**	The Examples column shows the default keyboard shortcuts for performing a forward and backward incremental search on Windows. For more information, see "Identifying Keyboard Shortcuts" on page 2-75.
Show Function button		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
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. 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 performs a different action, as described in the table that follows.

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 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.
Figure file (`.fig`)	Opens in a figure window.
Variable	Opens in the Variable Editor.
Model	Opens in Simulink.
Other	If the selection is some other type, **Open selection** looks for a matching file in a private folder in the current folder and performs the appropriate action.

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.

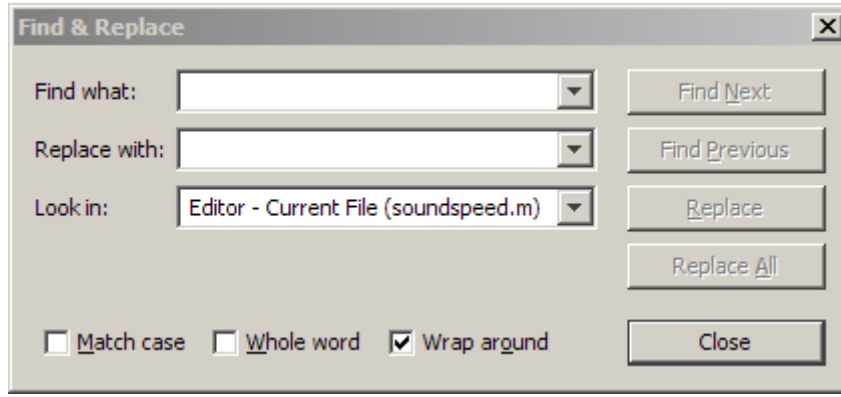
To find the previous occurrence of selected text (find backwards) in the current file, select **Find Previous** from the **Edit** menu. The previous occurrence of the text is selected. Repeat to continue finding the previous occurrences of the text.

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 toolbar, or select **Edit > Find and Replace**. Complete the resulting **Find Replace** dialog box.



The search begins at the current cursor position. The Editor finds the text you specified and highlights it. To find another occurrence, click **Find Next** or **Find Previous**, or use the keyboard shortcuts for these actions.

The MATLAB software 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 keyboard shortcuts 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. To determine the keyboard shortcuts for **Find Next** and **Find Previous** , see “Identifying Keyboard Shortcuts” on page 2-75

Change the selection in the **Look in** field to search for the specified text in other Microsoft 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 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 folders and file names that include specified text, or whose contents contain specified text, use **Edit > Find Files**. For details, see "Finding Files and Folders" on page 6-20.

Performing an Incremental Search in the Editor

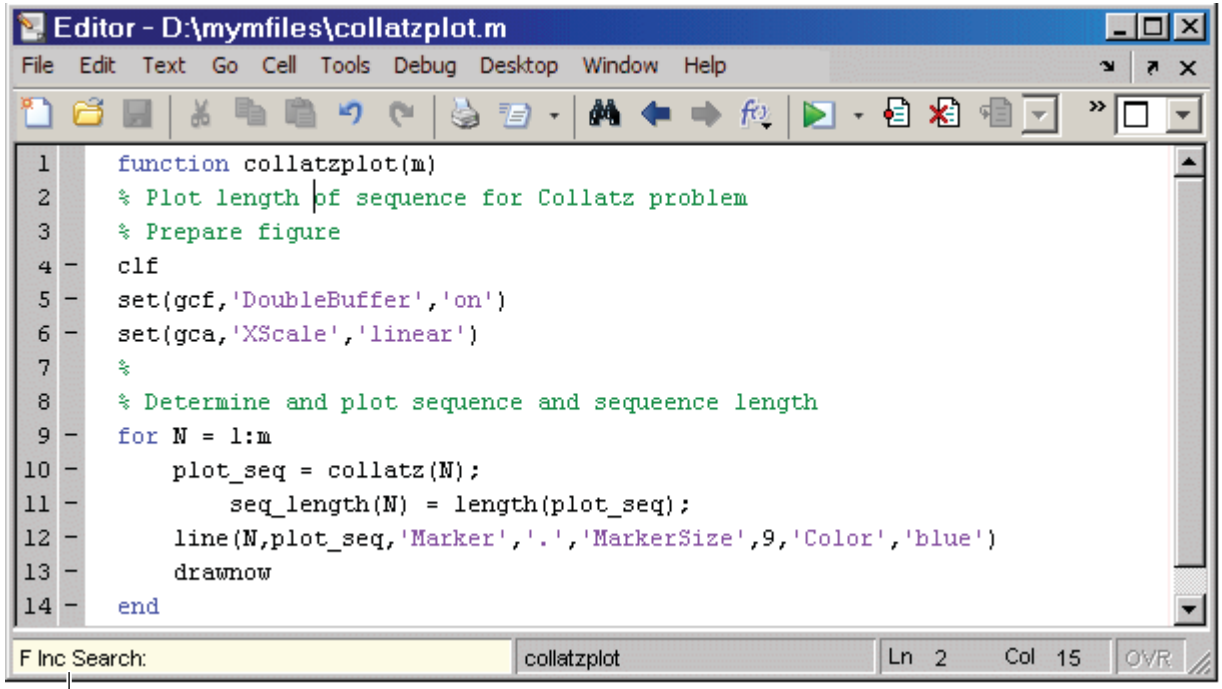
With the incremental search feature, the cursor moves to the next or previous occurrence of the specified text in the current file. It is similar to the Emacs search feature.

To use the incremental search feature in the Editor, use the steps that follow. However, be aware that this example uses keyboard shortcuts from the **Windows Default Set** for initiating a forward or backward incremental search. Once in incremental search mode, the keyboard shortcuts are fixed. That is, you cannot customize keyboard shortcuts within incremental search mode.

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

2 Press **Ctrl+Shift+S** to search forward or **Ctrl+Shift+R** to search backward if you are using the default Windows keyboard shortcuts.

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.

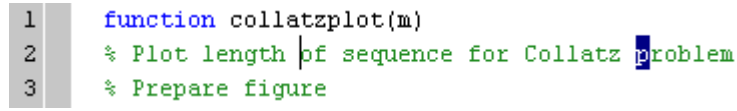


Incremental search field.

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 the 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` exists 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 a search 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.

For information on viewing and setting keyboard shortcuts, see “Identifying Keyboard Shortcuts” on page 2-75 and “Customizing Keyboard Shortcuts” on page 2-78

Comparing Files and Folders

In this section...

What Is the File and Folder Comparisons Tool?

The File and Folder Comparisons tool determines and displays the differences between two files or two folders (sometimes referred to as directories).

You can use this tool to:

- Compare lines in two text files (some other applications refer to this as a file diff operation).
- Compare variables in two MAT-files.
- Determine whether the contents of two binary files are the same.
- Compare two folders to determine which file names are unique to each folder.
- Compare two folders to determine if files with the same name in each folder have the same content.

Comparing Two Text Files

When you use the File and Folders Comparisons tool to compare two text files, a window opens and presents the two files side by side, along with symbols to indicate how you can adjust the files to make them match. This

is useful, for example, when you want to compare the latest version of a text file to an autosave version.

To compare two text files, follow these steps:

1. Open one of the text files you want to compare in the Editor.

   To open the example file provided, `lengthofline.m`, run the following command in the Command Window:

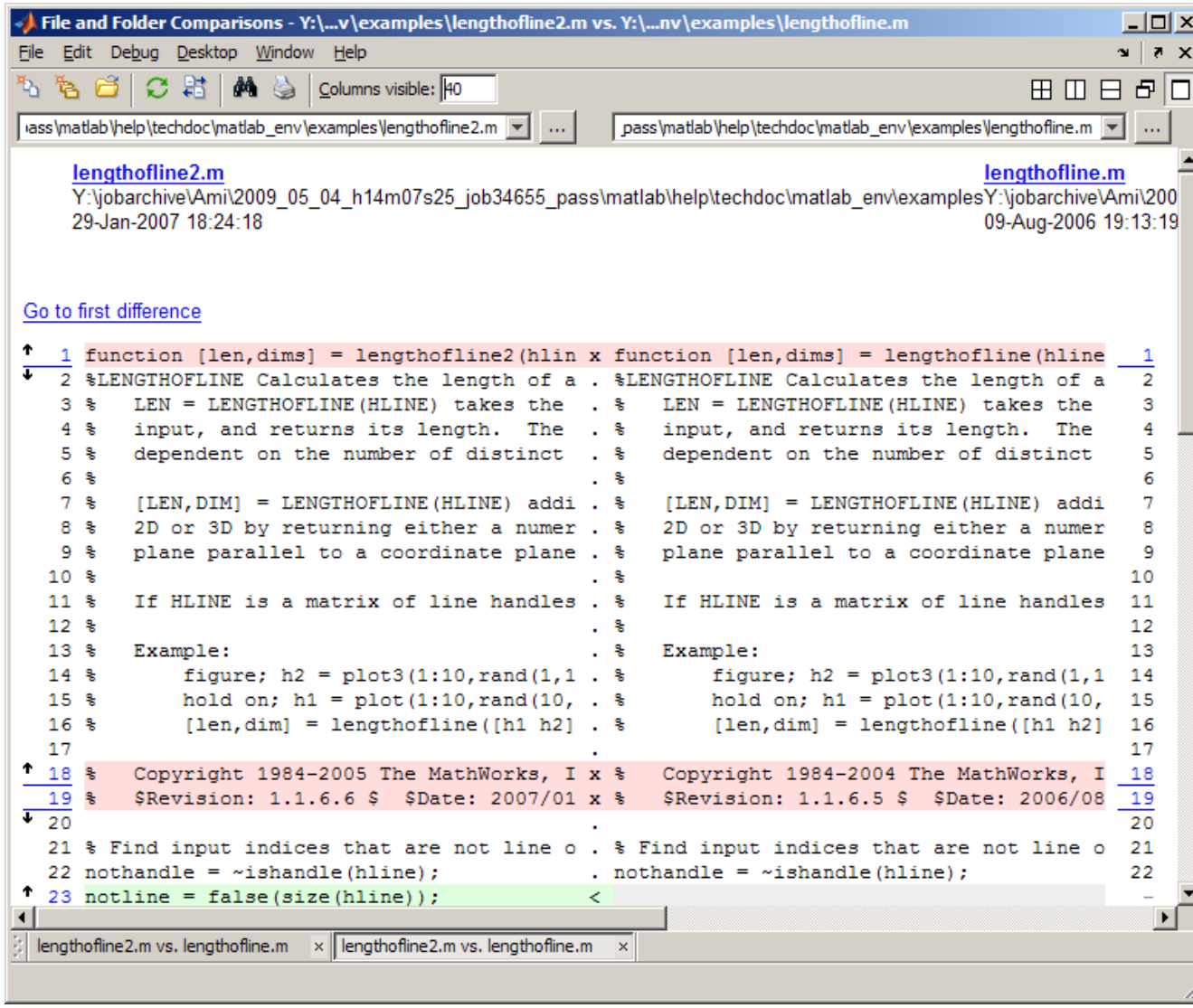
   ```
   open(fullfile(matlabroot,'help','techdoc','matlab_env',...
    'examples','lengthofline.m'))
   ```

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` from the folder where you found `lengthofline.m`. Other options available are the following:

   - **Tools > Compare Against > Autosave Version** to compare the open file to the Editor's automatic copy, *`filename`*`.asv`. For more information, see "Autosave" on page 8-95.
   - **Tools > Compare Against Version on Disk** to compare an open file that has been changed, but not saved, to the saved version.

   The File and Folder Comparisons tool opens, displaying the files side by side and highlighting lines that do not match, as follows:

   - Pink highlighting and an x at the start of a line indicate that the content of the lines differs between the two files.
   - Green highlighting and a > at the start of a line indicate a line that exists in the file presented on the right, but not in the file presented on the left.
   - Green highlighting and a < at the end of a line indicate a line that exists in the file presented on the left, but not in the file presented on the right.



3 Use the features of the File and Folder Comparison tool to work with the results.

Typically, when this tool compares two text files, it does not do a simple line-by-line comparison. In the previous image, for example, the tool determines that `lengthofline2.m`, has a line of code that does not exist in `lengthofline.m`, and highlights it (line 23) in green. Also notice that the tool takes the additional line into account and determines that the line containing the `end` statement in each file matches, even though the `end` statement does not occur on the same line number.

If the files being compared are extremely long, however, the tool may run out of memory in attempting to perform the file comparison. It then displays the message, `Maximum file length exceeded. Defaulting to line-by-line comparison.` In this case, the tool highlights the lines containing the `end` statement because in performing a simple line-by-line comparison it finds that the last line in one file does not match the last line in the other file.

Comparing Two MAT-Files

You can use the File and Folders Comparisons tool to compare two MAT-files. The tool presents the variables in the two files side by side, which enables you to:

- See which variables are common to each file and which are unique.
- Load the contents of the variables into the Variable Editor.
- Load the MAT-files into the workspace.

To compare two MAT-files, follow these steps:

1. Select **Desktop > File and Folder Comparisons**.

   The File and Folder Comparisons window opens a dialog box that is empty except for the title bar, menu bar, and a toolbar.

2. Click the New file comparison button:  .

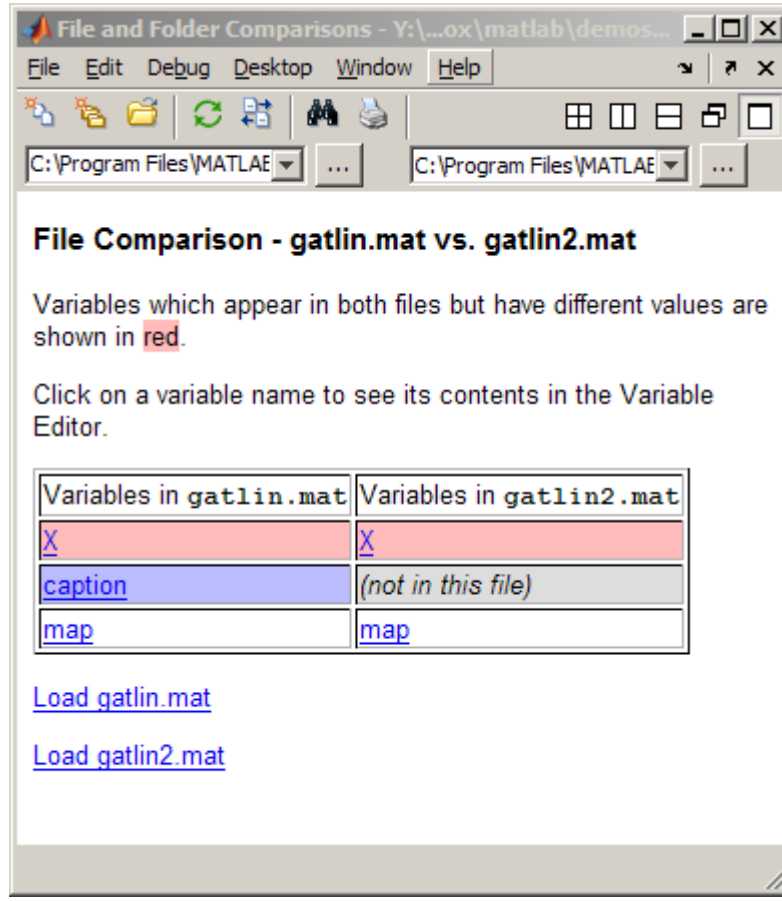
3. In left side of the File and Folder Comparisons window, click the **Select a File** button  . Then browse to and select the name of one of the MAT-files that you want to compare.

4 In right side of the File and Folder Comparisons window, enter the full file name of the other MAT-file that you want to compare.

The File and Folder Comparisons tool displays the file variable names side by side and highlights variables that do not match, as follows:

- Pink highlighting indicates that the values of the variables differ between the two files.
- Green highlighting indicates a variable that exists in the file displaying on the right, but not in the file displaying on the left.
- Purple highlighting indicates a variable that exists in the file displaying on the left, but not in the file displaying on the right.

The following image shows the results when you compare *matlabroot*`/toolbox/matlab/demos/gatlin2.mat` to *matlabroot*`/toolbox/matlab/demos/gatlin.mat`. To determine your *matlabroot* folder, type `matlabroot` in the Command Window.



5 To view the contents of a variable in the Variable Editor, click the name of that variable.

6 To load the variables for a specified file into the workspace, click a load link.

Comparing Two Binary Files

When you use the File and Folder Comparisons tool to compare two non-MAT-file binary files, such as DLL files or MEX-files, the tool returns a message indicating whether the files are the same.

To compare two binary files, follow the same steps in "Comparing Two MAT-Files" on page 8-84. If the files are the same, the tool displays the message: `The files are` **`identical`**. If the files differ, the tool displays the message: `The files are` **`different`**. `MATLAB cannot display the differences between files of these types`.

Comparing Two Folders

When you use the File and Folder Comparisons tool to compare two folders, a window opens and presents the contents of the folders, side by side. The tool enables you to:

- Determine the files that the folders have in common.
- Determine if files with identical names that are common to both folders also have identical content.
- Open for comparison two files that are common to both folders, but have different content.
- Open for comparison two subfolders that are common to both folders, but have different content.
- Open a file for viewing in the Editor.

To compare two folders, follow these steps:

1. Select **Desktop > File and Folder Comparisons**.

   The File and Folder Comparisons window opens.

2. Select **File > New Folder Comparison** or click the New folder comparison button  .

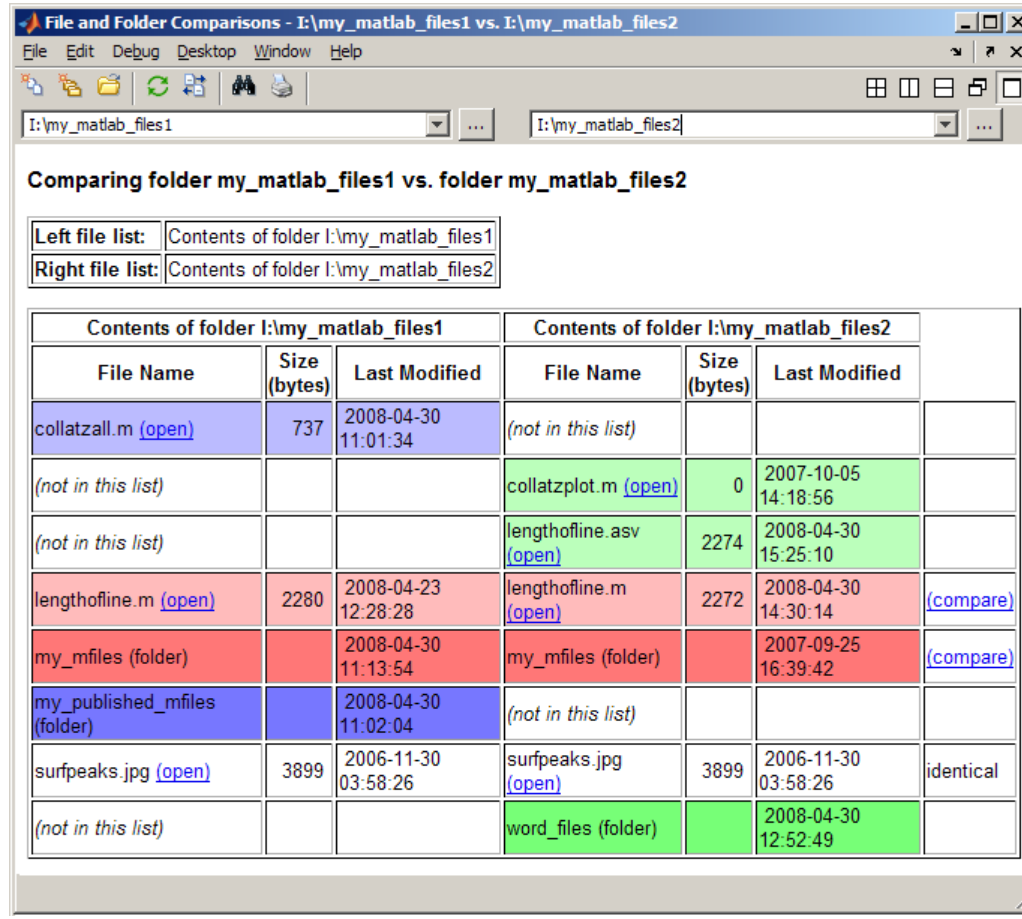
   The File and Folder Comparisons window refreshes with a `Type a folder name here` field on each side of the tool.

3. Type or browse to a folder on each side of the tool.

   The File and Folder Comparisons tool displays the contents of the folders side by side and highlights files and subfolders that do not match, as follows:

- Light red highlighting indicates that the contents of the files differ.
- Dark red highlighting indicates that the contents of the subfolders differ
- Light green highlighting indicates a file that exists in the folder on the right, but not in the folder on the left.
- Dark green highlighting indicates a subfolder that exists in the folder on the right, but not in the folder on the left.
- Light blue highlighting indicates a file that exists in the folder on the left, but not in the folder on the right.
- Dark blue highlighting indicates a subfolder that exists in the folder on the left, but not in the folder on the right.

The following image shows an example of the File and Folder Comparisons tool when two folders are compared.



Contents of folder I:\my_matlab_files1			Contents of folder I:\my_matlab_files2			
File Name	Size (bytes)	Last Modified	File Name	Size (bytes)	Last Modified	
collatzall.m (open)	737	2008-04-30 11:01:34	*(not in this list)*			
(not in this list)			collatzplot.m (open)	0	2007-10-05 14:18:56	
(not in this list)			lengthofline.asv (open)	2274	2008-04-30 15:25:10	
lengthofline.m (open)	2280	2008-04-23 12:28:28	lengthofline.m (open)	2272	2008-04-30 14:30:14	(compare)
my_mfiles (folder)		2008-04-30 11:13:54	my_mfiles (folder)		2007-09-25 16:39:42	(compare)
my_published_mfiles (folder)		2008-04-30 11:02:04	*(not in this list)*			
surfpeaks.jpg (open)	3899	2006-11-30 03:58:26	surfpeaks.jpg (open)	3899	2006-11-30 03:58:26	identical
(not in this list)			word_files (folder)		2008-04-30 12:52:49	

4 Click the `open` link next to a file name to open that file in the Editor.

5 Click the `compare` link next to a set of folder names that are highlighted in dark red to refresh the File and Folder Comparisons tool with the two highlighted folders presented for comparison.

6 Click the `compare` link next to a set of file names that are highlighted in light red to refresh the File and Folder Comparisons tool with the two highlighted files presented for comparison.

Using Features of the File and Folder Comparisons Tool

The File and Folder Comparisons tool provides features that let you do any of the tasks described in the following sections:

Navigating from One Difference to the Next

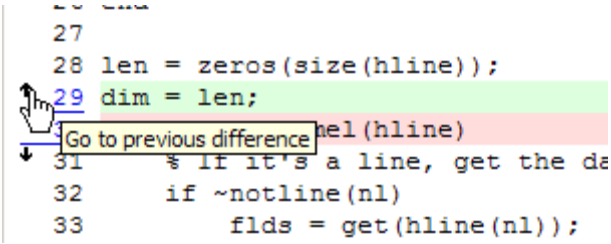
Because text files can be lengthy, the File and Folder Comparisons tool provides links to help you navigate from one difference to the next. Do one of the following to display a different set differences, relative to the current set of differences displaying in the Editor:

- To navigate to a previous set of lines that differ, click the up arrow.

  If there is no previous set of lines that differ, the up arrow takes you to the beginning of the file.

- To navigate to the next set of lines that differ, click the down arrow.

  If there are no additional sets of lines that differ, the down arrow takes you to the end of the file.



Increasing or Decreasing Line Lengths Shown for Text Files

When comparing text files, the display is 60 columns wide, by default. To increase the display width, type a high number in the **Columns visible** field, and then drag the vertical edges of the window to make it wider. If keeping the window size narrow results in more columns appearing for the file on the left than for the file on the right, reduce the number for **Columns visible** to display a sufficient number of columns for both files, given the window width.

Exchanging Positions

To move the file or folder on the left side to the right side and vice versa, select **File > Swap Sides**, or click the Swap sides button .

Showing Updated Files

After making changes to and saving the files in the Editor, update the results in the File Comparisons tool by selecting **File > Refresh** or clicking the Refresh button .

Finding Text

To find a phrase in the current display, select **Edit > Find**, or click the Find text button . The resulting Find dialog box is the same as the one you use in the Command Window. For more information, see “Finding Text Currently Displayed in the Command Window” on page 3-49.

Viewing a Summary of Differences

To see a summary of difference between two text files, scroll to the bottom of the File and Folder Comparisons tool. There you find a list such as the following:

- Number of matching lines:52
- Number of unmatched lines in left-hand file: 12
- Number of unmatched lines in right-hand file: 15

Replacing a File or Folder Being Compared with Another File or Folder

If the tool is currently comparing files, replace an existing file in the tool by doing the following:

1. Locate the file and folder field  above the file that you want to replace.
2. Do one of the following:
   - Drag a file from Windows Explorer to the file and folder field.
   - Type the path to a file in the file and folder field.
   - Click the Browse for file button  next to the file and folder field, and then select a file.
   - Place the cursor in the file and folder field, and then select **File > Open**.

If the tool is currently comparing folders, you can replace an existing folder. Type the path to a folder, or browse to a folder in the file and folder field.

Viewing New Comparisons

You can perform another file comparison by selecting **File > New File Comparison** or clicking the New file comparison button  .

You can perform another folder comparison by selecting **File > New Folder Comparison** or clicking the New folder comparison button  .

Viewing Previous Comparisons

You can see the results of previous comparisons in the current session by selecting that comparison's entry in the document bar (as shown at the bottom of the window in the illustration in "Comparing Two Text Files" on

page 8-81). If you close the File and Folder Comparisons tool, the current and previous comparisons are lost.

Accessing the File and Folder Comparisons Tool

In addition to the methods shown in the previous sections, you can also access the File and Folder Comparisons tool using one of these methods:

- From the MATLAB desktop, select **Desktop > File and Folder Comparisons**.
- From the Current Folder browser, select a file or folder, right-click, and from the context menu, select **Compare Against**.
- For two files or subfolders in the same folder, from the Current Folder browser, select the files or folders, right-click, and from the context menu, select **Compare Selected Files** or **Compare Selected Folders**.

Supply the files or folders to compare as described in "Comparing Two Text Files" on page 8-81 and "Comparing Two Folders" on page 8-87, respectively.

Function Alternative for Comparing Files and Folders

Use the `visdiff` function to open the File and Folder Comparisons tool from the Command Window. For example, type:

```
visdiff('lengthofline.m', 'lengthofline2.m')
```

Saving, Printing, and Closing Files in the Editor

In this section...

Saving M-Files

After making changes to a file, an asterisk (*) follows the file name in the title bar of the Editor. This asterisk 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 clicking the Save button on the toolbar. If the file has not been changed, **Save** appears dimmed, but you can instead use **Save As** from the **File** menu to save to a different file name.
- **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 software automatically assigns the `.m` extension to the file name. If you do not want an extension, type a `.` (period) after the file name.
- **Save All** — Saves all open files to their existing file names. For all newly created files, the **Save file as** dialog box opens, where you assign a name to each untitled file and save it. Another way to save all open files is by clicking the Save All button. This button is not on the toolbar by default, however. For information on adding it, see "Setting Toolbars Preferences for Desktop Tools" on page 2-156.

You cannot save an M-file while in debug mode. If you try to, MATLAB desktop 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 8-175.

Recommendations on Saving M-Files

The MathWorks™ recommends that you save M-files you create and M-files from The MathWorks that you edit to a folder that is not in the *matlabroot*/toolbox folder tree. If you keep your files in *matlabroot*/toolbox folders, they can be overwritten when you install a new version of MATLAB software.

Be aware that locations of files in the *matlabroot*/toolbox folder tree are loaded and cached in memory at the beginning of each MATLAB session to improve performance. Therefore, if you save files to *matlabroot*/toolbox folders using an external editor, or add or remove files from these folders 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 folders using an external editor, run `clear functionname` before you use these files in the current session. For more information, see `rehash` or "Toolbox Path Caching in the MATLAB Program" on page 1-22.

Autosave

As you make changes to a file in the Editor, every 5 minutes the Editor 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, select **File > Preferences > Editor/Debugger > Autosave**, and then click **Help**.

If the file you are editing is in a read-only folder and the autosave preference for location is the source file folder, 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, 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 using **File > Source Control**. For more information, see Chapter 12, "Source Control Interface".

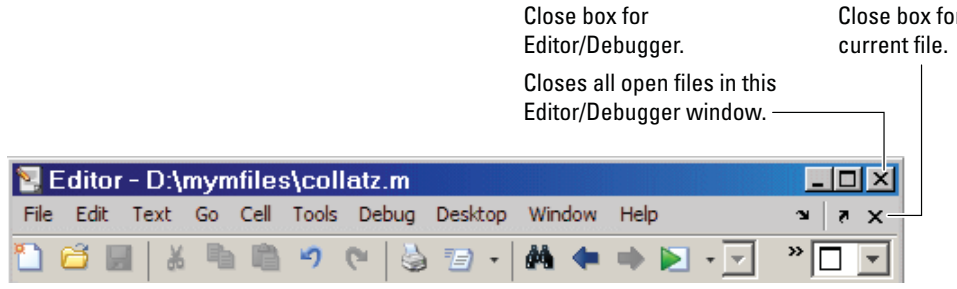
Printing M-Files

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 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 2-119.

Closing M-Files

To close the current M-file, select **Close *filename*** from the **File** menu, or click the Close box in the Editor menu bar. This is different from the Close box in the titlebar of the Editor, which closes all open files in that Editor window.



To close all files within the Editor, select **Window > Close Editor Documents**. This does not close any files undocked from the Editor. The Editor 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 Variable 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

In this section...

Running M-Files with No Input Arguments in the Editor

In the Editor, 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 multiple files open. Alternatively, select **Debug > Run *`filename`***.

If the file is neither in a folder on the search path nor in the current folder, a dialog box appears with options that allow you to run the file. You can either change the current folder to the folder containing the file, or you can add the folder containing the file to the search path.

If the file has unsaved changes, running it from the Editor automatically saves the changes before running. In that event, the **Debug** 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 *data tip* (like a Tooltip for data). You need to set the preference to show data tips in edit mode—select

File > Preferences > Editor/Debugger > Display, and for **General Display Options**, select the check box for **Enable datatips in edit mode**.

Using Run Configurations to Run M-Files with Input Arguments in the Editor

In the Editor, you can provide values for a function's input arguments using a run configuration, and then run that configuration to use the assigned values. When you are editing a function M-file, use a run configuration as an alternative to running the function in the MATLAB Command Window. You can associate multiple run configurations with an M-file to assign different input values. MATLAB saves the run configurations between sessions to a file named `run_configurations.m`. (For details, see "About the run_configurations.m File" on page 8-108.)

Consider the function `collatzplot_new.m`, which computes and plots the Collatz sequence for any given positive integer. This function requires you to specify the integer as an input value. You cannot simply run `collatplot_new.m` in the Editor because the input value is not defined. One way to specify the input value is to run the M-file in the Command Window. Run configurations allow you to run `collatzplot_new(`*`specific value`*`)` in the Editor.

You can also use run configurations to provide preparatory or setup information before running an M-file, whether it takes input arguments or not.

Note M-File run configurations use the base MATLAB workspace. Therefore, a value that you assign to a variable in an M-file run configuration overwrites the value for that variable (assuming it currently exists) in the base workspace.

Create and Use a Run Configuration for an M-File

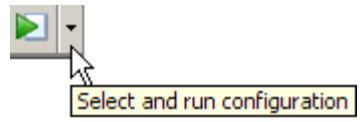
Follow these steps to create and use a run configuration for an M-file in the Editor. These steps specify Editor toolbar buttons, but you can also use equivalent options in the **Debug** menu.

1 Open the file you want to run in the Editor. For example, open `collatzplot_new.m` by running the following command:

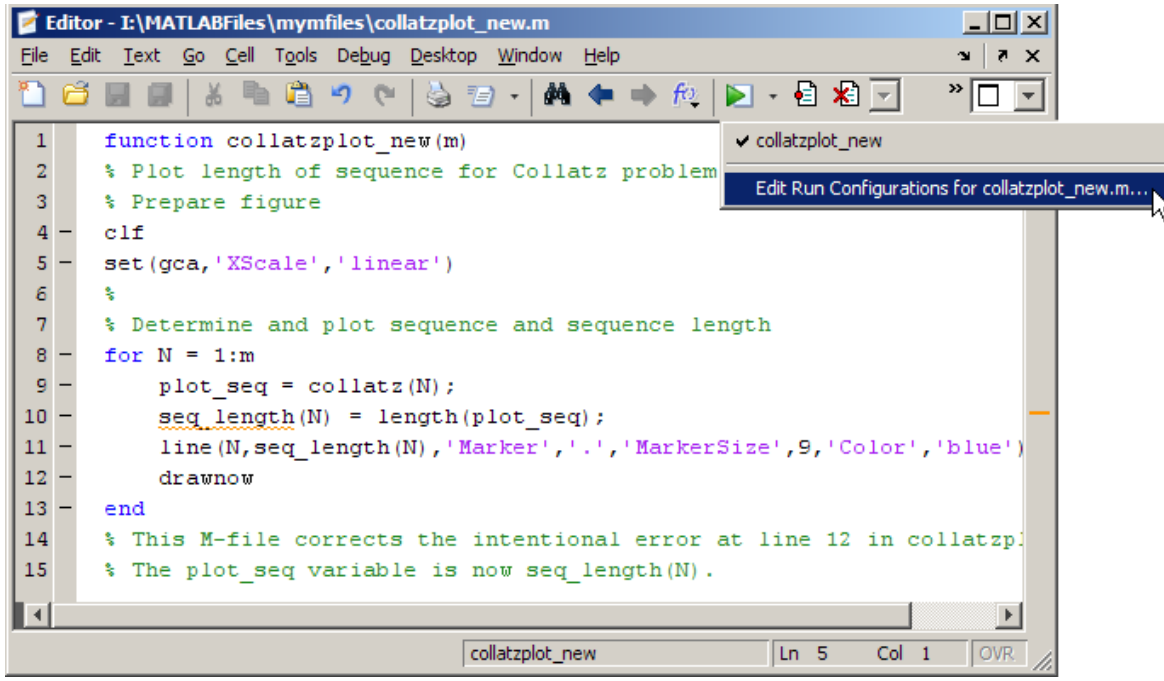
```
edit(fullfile(matlabroot,'help','techdoc',...
'matlab_env','examples','collatzplot_new.m'))
```

To work with `collatzplot_new.m` on your system, save the file to a folder for which you have write permission. In the example, the file is saved to `I:\my_matlab_files\my_mfiles\collatzplot_new.m`.

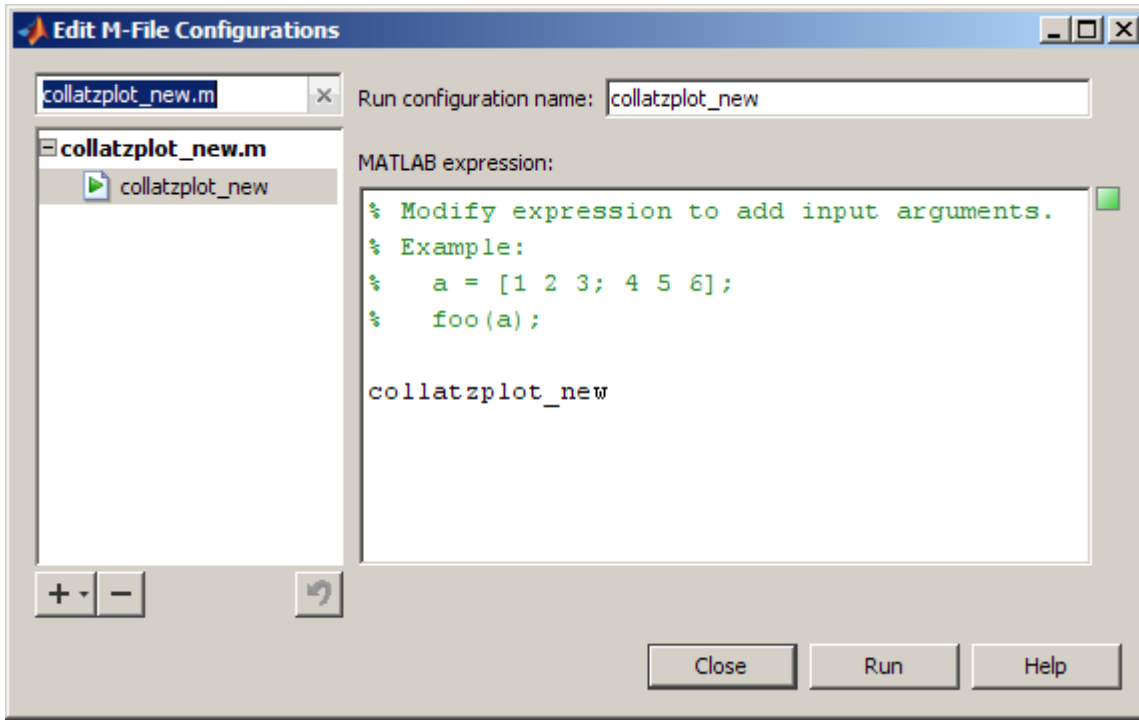
2 Click the down arrow on the Run button in the Editor toolbar,



and then select **Edit Run Configurations for *`file-name`***, where *`file-name`* in this example is `collatzplot_new.m`.

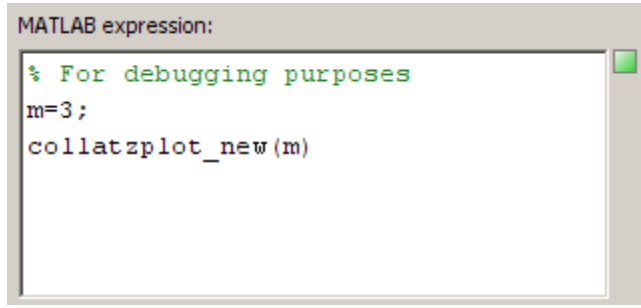


The Edit M-File Configurations dialog box opens, with a default run configuration template for `collatzplot_new.m`.



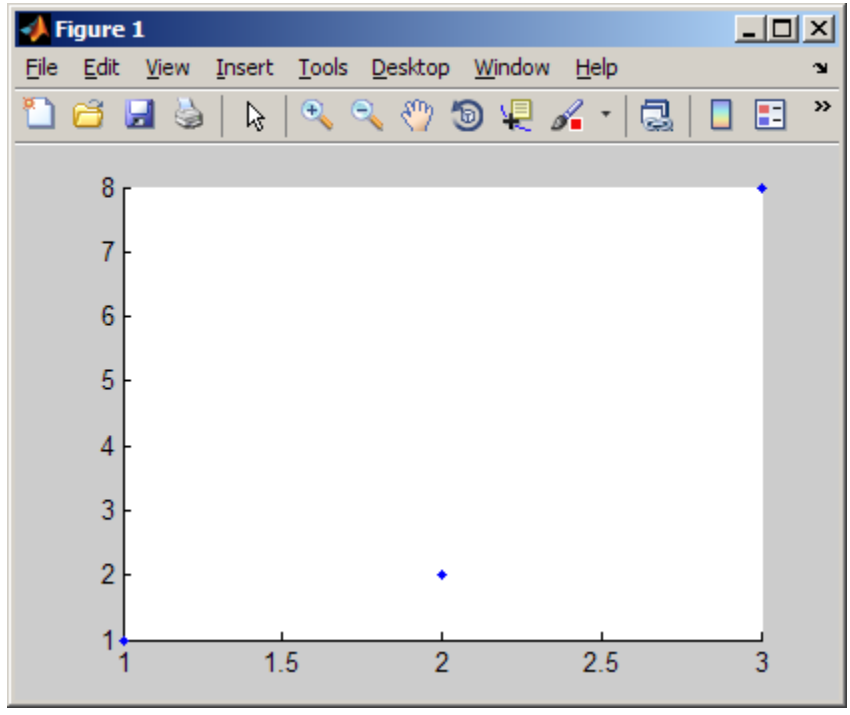
3 In the **MATLAB expression** area of the dialog box, enter MATLAB statements that you want to run. Delete the existing comments or replace them with comments relevant to your run configuration. To undo and redo, use the keyboard shortcuts for your platform, such as **Ctrl+Z** and **Ctrl+Y** for Microsoft Windows platforms.

In this example, set `m` equal to `3`, which is a small value useful for debugging purposes. Complete the statement to run `collatzplot_new(m)`.



The **MATLAB expression** area provides syntax highlighting and shows M-Lint messages, similar to the Editor.

4 To ensure your run configuration executes as expected, click **Run** to execute the statements in the **MATLAB expression** field. 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 the **Run configuration name** field in the Edit M-File Configurations dialog box. By default, the run configuration name is the M-file name. If you expect to create multiple run configurations for an M-file, assign each a name that helps you identify the configuration. In this example, name the run configuration `collatzplot_new_test`.

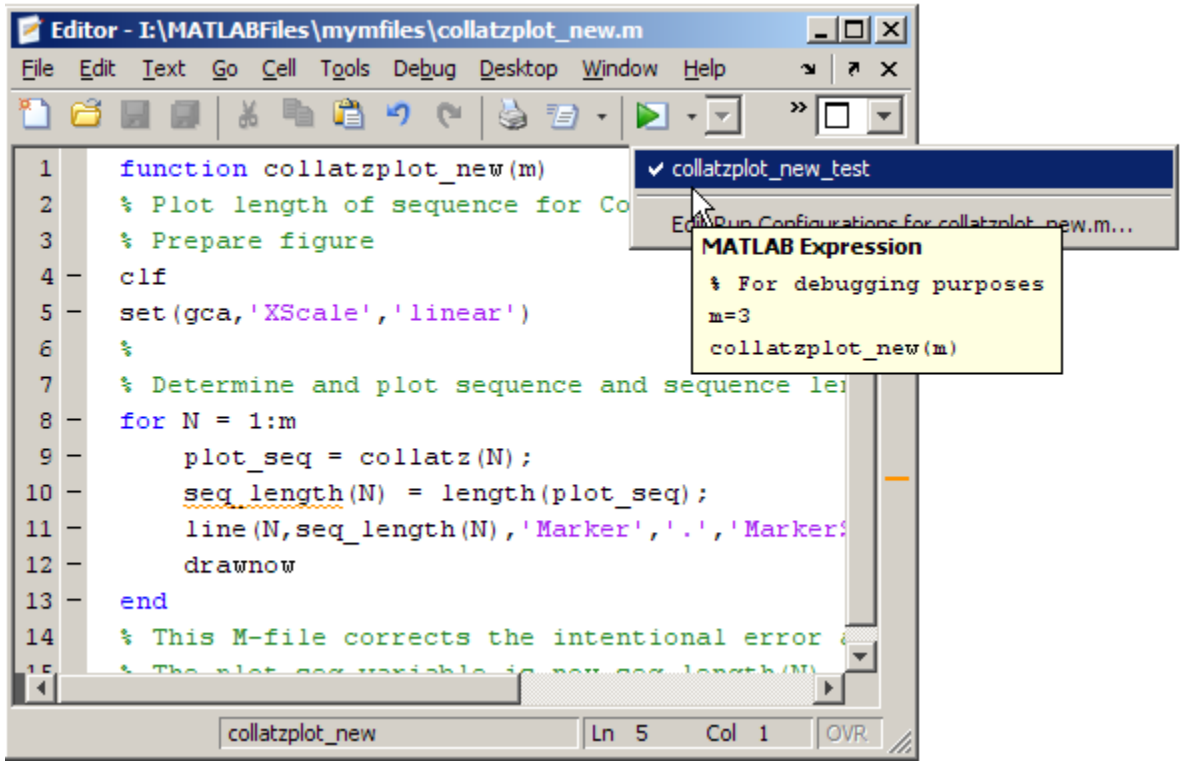
   MATLAB automatically saves the run configuration and its association with the M-file in the `run_configurations.m` file in your preferences folder.

   For more information, see "About the run_configurations.m File" on page 8-108.

7. To close the Edit M-File Configurations dialog box, click **Close**.

8. After creating a run configuration, you can view and use the configuration without opening the Edit M-File Configurations dialog box.

   In the Editor toolbar, click the down arrow on the Run button  and position the mouse pointer on a run configuration name. The MATLAB desktop displays a Tooltip showing the run configuration's **MATLAB Expression** so you can see what will run.



9 To use the run configuration, select the run configuration name. MATLAB runs the expression you specified in the run 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 execute the run configuration from the toolbar to see the effects of the M-file changes.

Create and Execute Multiple Run Configurations for an M-File

You can create multiple run configurations for a given M-file, allowing you to run with different values for input arguments, each for a different purpose. Create a named run configuration for each purpose, all associated with the M-file. Then any time you open the M-file, choose and execute the run configuration you want. For example, for `collatzplot_new(m)` you might use three values for `m` and have three run configurations:

- Small value, for example, `3`, for debugging and testing
- Realistic value, for example, `200` or more, for a specific project
- Random value to observe changes

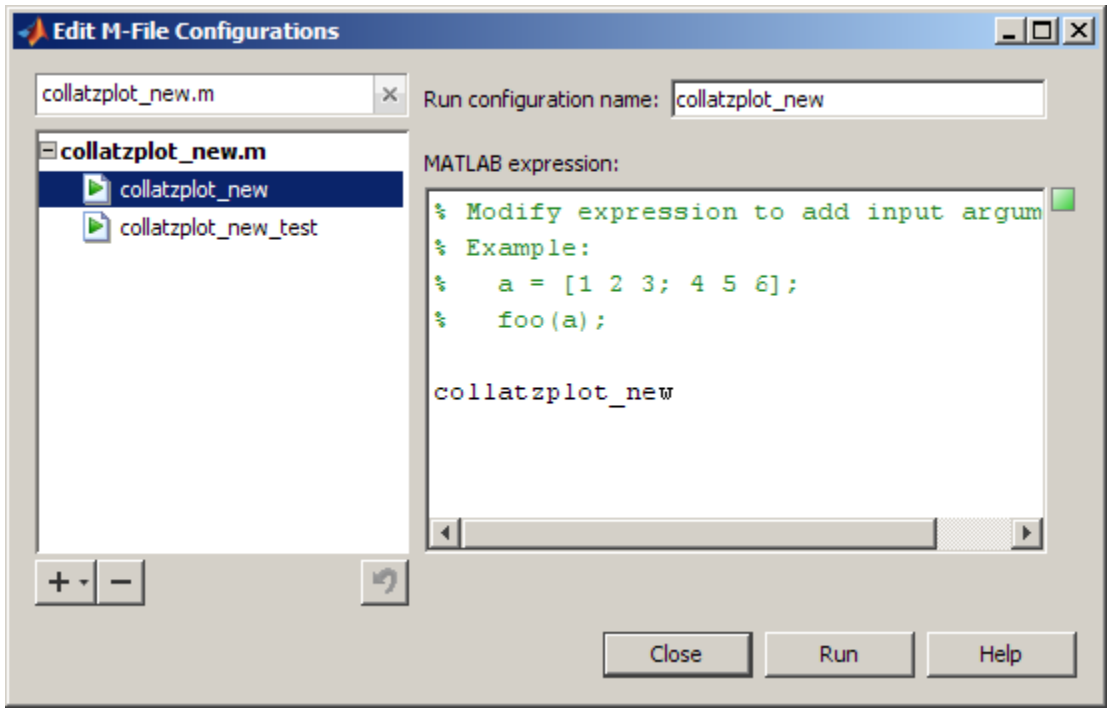
1. Open the Edit M-File Configurations dialog box, and then do the following:

   a. Select the M-file to which you want to add a run configuration, or select a configuration associated with that M-file.

   b. Click the Add button  (under the list of M-files and configurations), and then click **Run Configuration**.

   MATLAB creates a new default run configuration template, in this example, `collatzplot_new`.

   The example shows `collatzplot_new` and its default expression, as well as one previously created run configuration associated with `collatzplot_new.m`, `collatzplot_new_test`.



2 In the Edit M-File Configurations dialog box, modify, run, and name the new run configurations as you did for the initial run configuration, `collatplot_new_test`, as described in "Create and Use a Run Configuration for an M-File" on page 8-99.

For example, rename `collatzplot_new` to `collatzplot_new_largevalue`, and replace the default template expression with:

```
m=200;
collatzplot_new(m)
```

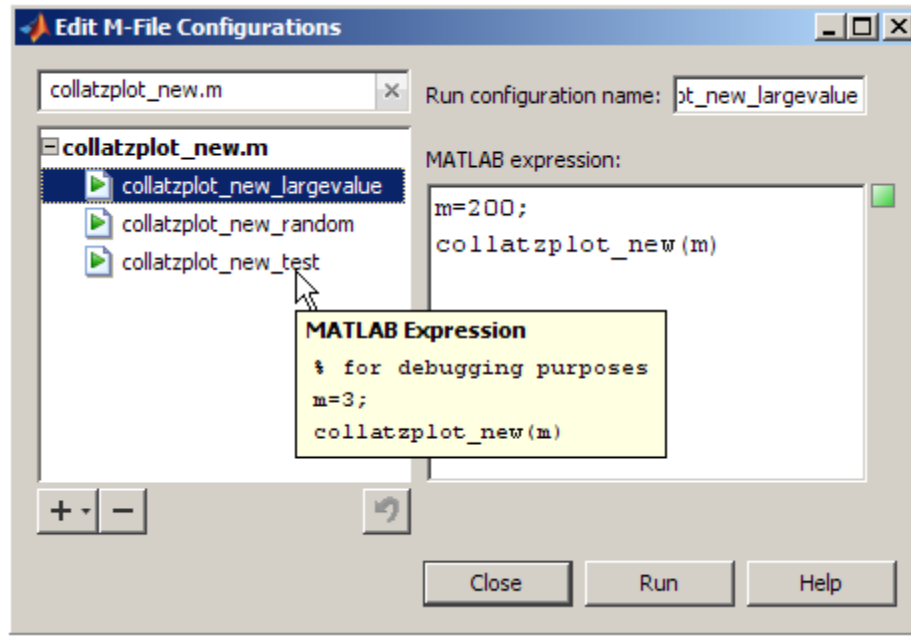
To create another run configuration, click the down arrow next to the Add button + again, and then click **Run Configuration**. Rename `collatzplot_new_2` to `collatzplot_new_random` and replace the default template expression with:

```
% Random value
```

```
m=int16(rand*50);
collatzplot_new(m)
clear all
```

3 Select a run configuration in the listing to see and modify its expression, or to rename the configuration. Click the expanders next to an M-file name (plus **+** and minus **-** signs on Windows platforms) 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 mouse 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 To close the Edit M-File Configurations dialog box, click **Close**. MATLAB saves the configurations and their associations with the M-file in the `run_configurations.m` file in your preferences folder.

For more information, see “About the run_configurations.m File” on page 8-108.

About the run_configurations.m File

When you create one or more run configurations using the Edit M-File Configurations dialog box, the Editor creates or updates the `run_configurations.m` file in your preferences folder (the folder MATLAB returns when you run `prefdir`). This is a text file that you can view and use to evaluate M-files.

Although you can port this file from the preferences folder on one system to another, there can only be one `run_configurations.m` file on a system. Therefore, you should only do this if you have not already created configurations on the second system. In addition, because this file may contain references to file paths, you need to be sure the specified M-files and paths exist on the second system.

The MathWorks recommends that you do not update this file in the Editor or a text editor. Changes you make using tools other than the Edit M-File Configurations dialog box may be overwritten.

Each time you change a run configuration using the Edit M-File Configurations dialog box, MATLAB updates the `run_configurations.m` file as well as the `publish_configurations.m` file. See “About the publish_configurations.m File” on page 10-109 for more information about that file.

Find Configurations

Follow these steps to find run or publish configurations. (For information on publish configurations, see “Producing Published Output from M-Files” on page 10-68.)

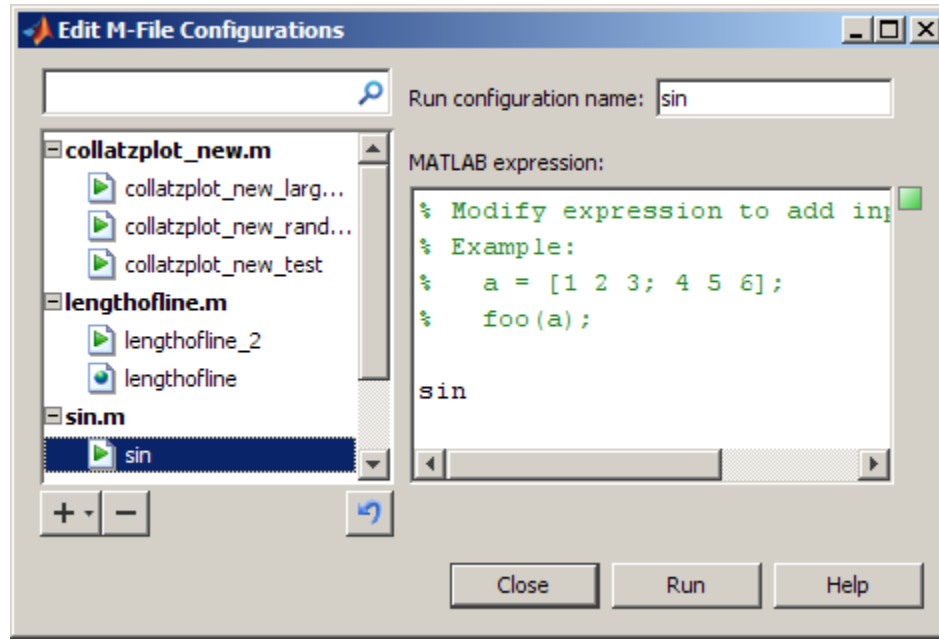
1. Open any M-file in the Editor. For example, open the MATLAB function `sin`.

2. Open the Edit M-File Configurations dialog box. MATLAB automatically creates a default configuration for `sin.m`, if none exists.

In the left pane, MATLAB lists all configurations currently defined for `sin.m`.

3 Click the X icon to clear the filter field.

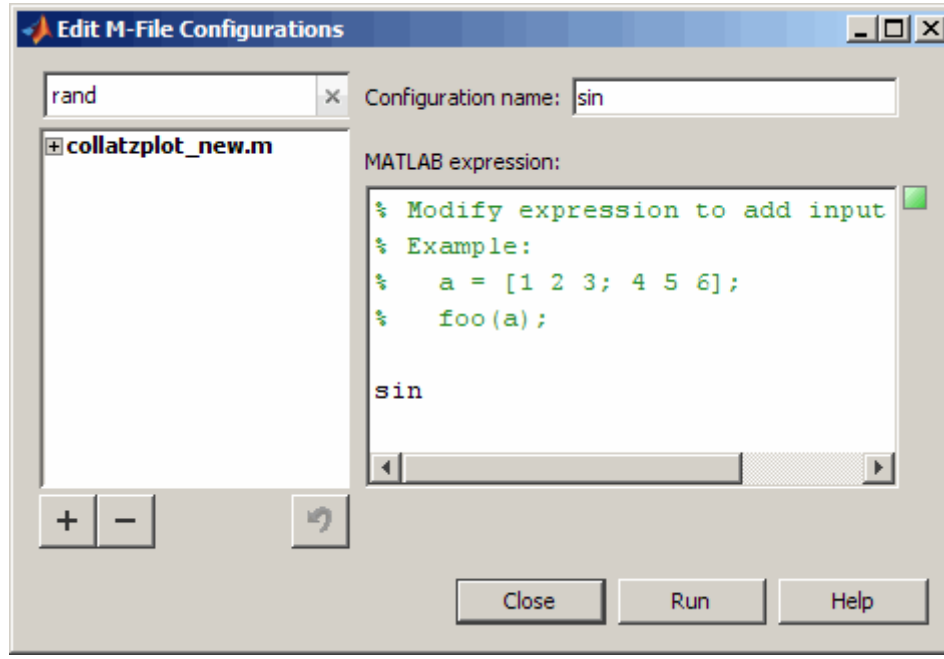
4 In the left pane, MATLAB lists all M-files containing configurations.



5 Type a term in the filter field  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`.



6 If you cannot view the entire name of a configuration, drag the separator bar to the right of the list, making the left pane wider.

7 To see the expression in that configuration, select the configuration, or position the mouse pointer over the name.

8 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 is empty.

Remove Configurations

If you no longer need a run or publish 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. (For information on publish

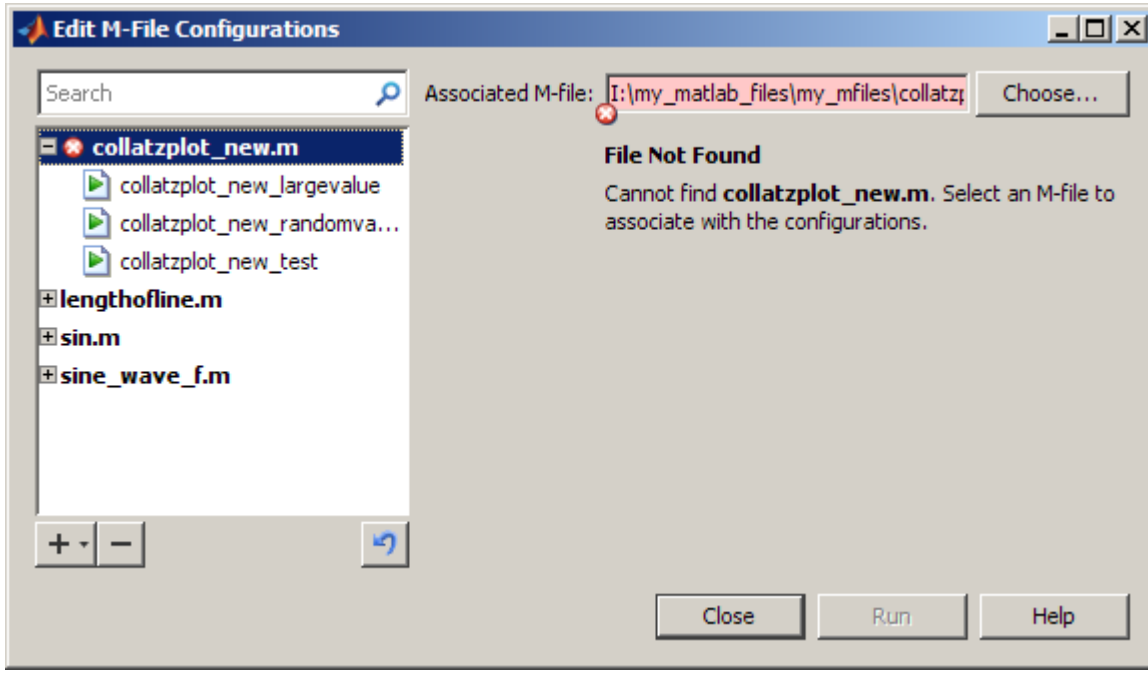
configurations, see “Producing Published Output from M-Files” on page 10-68.)

1. Open any M-file in the Editor.
2. Open the Edit M-File Configurations dialog box.
3. Do one of the following in the panel on the left:
   - If you want to remove a single configuration, select that configuration.
   - If you want to remove all the run and publish configurations for an M-file, select the M-file
4. Click the Remove button .
5. To undo the last deletion, click the Undo button . You cannot undo the last deletion after you close this dialog box.

Reassociate and Rename Configurations

Each run and publish configuration is associated with a specific M-file. If you move or rename an M-file that has configurations, 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_fixed.m`, so the configurations associated with `collatzplot_new.m` need to be reassociated with `collatzplot_fixed.m`.



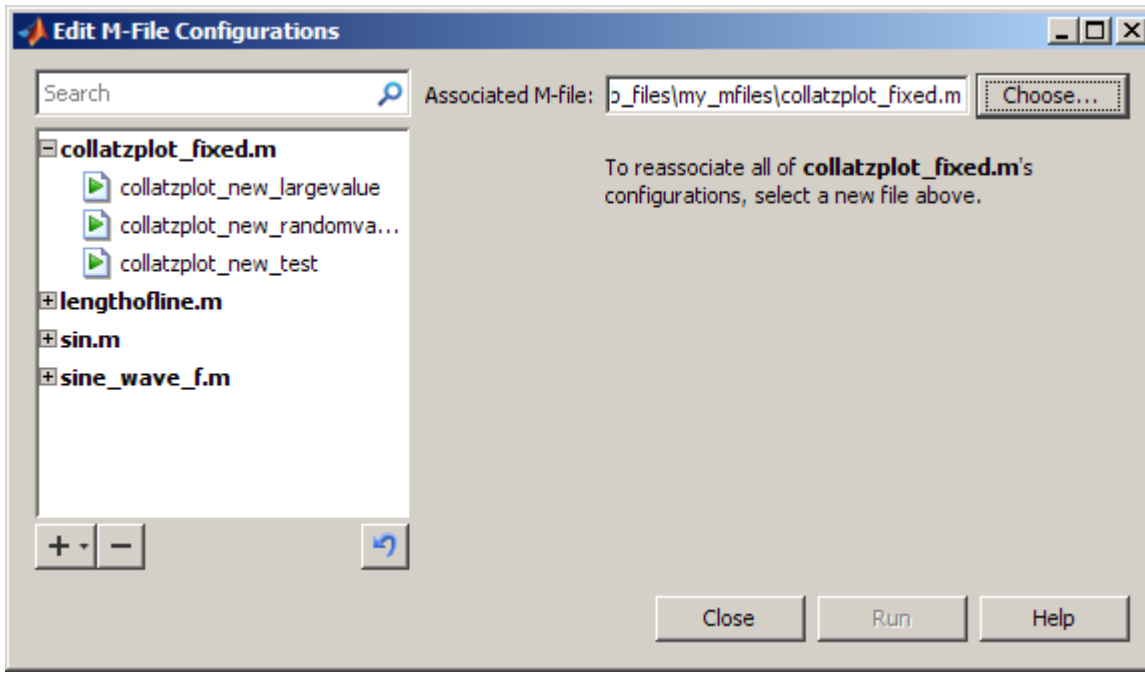
To reassociate a configuration:

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 reassociate the configurations. Click **Open**.

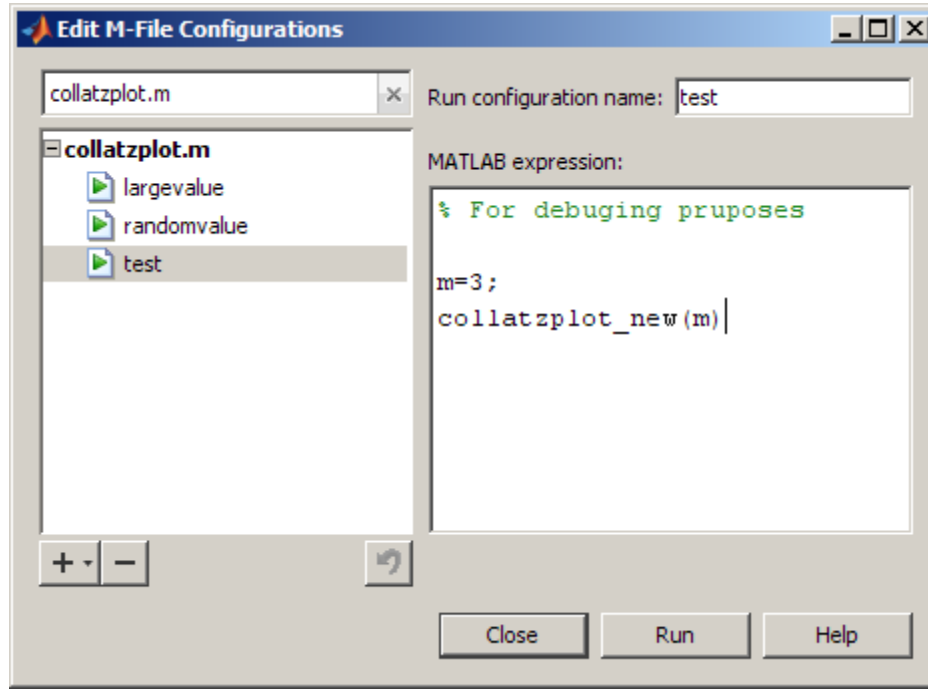
   In this example, you want to reassociate the configurations with `collatzplot_fixed.m`; select `collatzplot_fixed.m`, and then click **Open**.

   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.

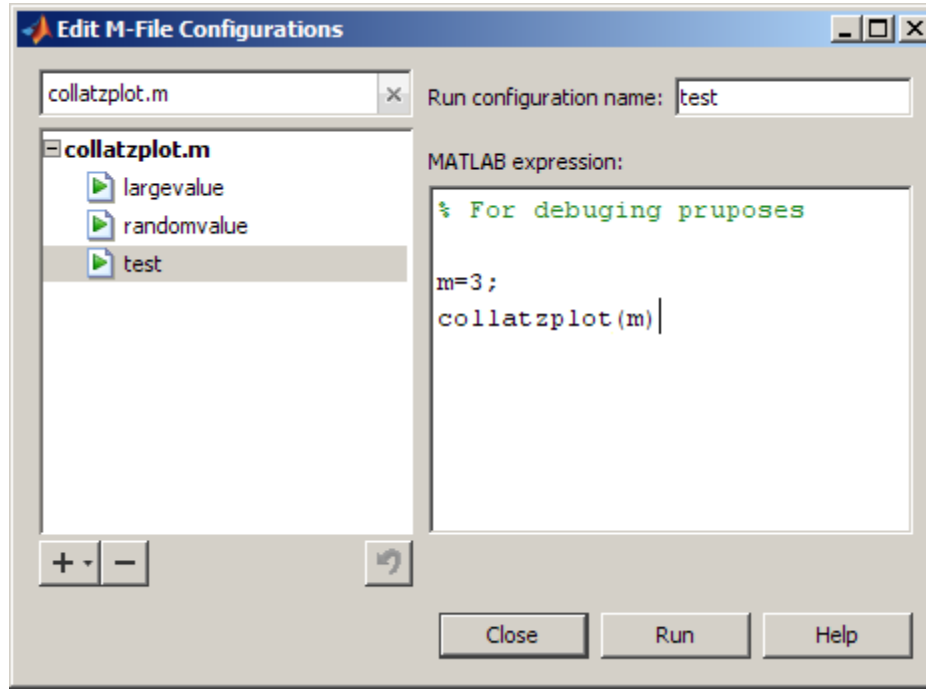


3 You might want to rename the configurations to be consistent with the new M-file name, or at least to not reflect the former M-file name. This is not required, but it is a good practice. To do so, select a configuration from the list in the left pane. In the right pane, edit the value for the configuration name. Depending on the type of configuration that you are renaming, the field is labeled either **Run configuration name** or **Publish configuration name**. Repeat this step for all run and publish configurations associated with the M-file.

In this example, remove `collatzplot_new` from the start of each run configuration name.



4 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)`.



Other Ways to Run M-Files from the Editor

- See "Running an M-File with Breakpoints" on page 8-160 for additional information about running M-files while debugging.
- While debugging, you can execute sections of an M-file even though there are changes. See "Running Sections in M-Files That Have Unsaved Changes" on page 8-175.
- 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 8-185.

Finding Errors, Debugging, and Correcting M-Files

This section introduces general techniques for finding errors and using the M-Lint automatic code analyzer to detect possible areas for improvement in M-files. It then illustrates the MATLAB debugger features in the Editor, as well as 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 8-52 "Matching Delimiters (Parentheses)" on page 3-21

Technique or Tool	Description	For More Information
Error Messages	When you run an M-file with a syntax error, MATLAB software 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 then press **Ctrl+Enter**. (This is the default keyboard shortcut for the **Display Hyperlinked Error** action on Windows). The offending M-file opens in the Editor, 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.	"Identifying Keyboard Shortcuts" on page 2-75
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, 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.	"Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 8-119 and the reference page for the `mlint` function
Editor, Graphical Debugger, and MATLAB Debugging Functions	The MATLAB Editor, 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 8-151

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, 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 8-185
Profiler	Use the Profiler to help you improve performance and detect problems in your M-files. Access the Profiler from the Editor by selecting **Tools > Open Profiler**.	"Profiling for Improving Performance" on page 9-27
Reports	The M-File reports helps you polish and package M-files before providing them to others to use. Access all of the reports from the Current Folder browser.	"Using M-File Reports" on page 9-2

Checking M-File Code for Problems Using the M-Lint Code Analyzer

In this section...

What Is the M-Lint Code Analyzer?

The M-Lint code analyzer, also referred to as just M-Lint, checks your code for problems and recommends modifications to maximize performance and maintainability.

Ways to Use the M-Lint Code Analyzer

You can use M-Lint in three different ways, all of which report the same messages:

- Continuously check code in the Editor 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 extended messages, automatic code correction, or both. Details about using the continuous checking and correction interface in the Editor is explained in "Using M-Lint Automatic Code Analyzer in the Editor" on page 8-120

- Run a report for an existing M-file:

  1. From an M-file in the Editor, select **Tools > M-Lint > Show M-Lint Report**.
  2. Make any changes to your file based on the M-Lint messages in the report.

3. Save the file.
4. Rerun the report to see if your changes addressed the issues noted in the M-Lint messages.

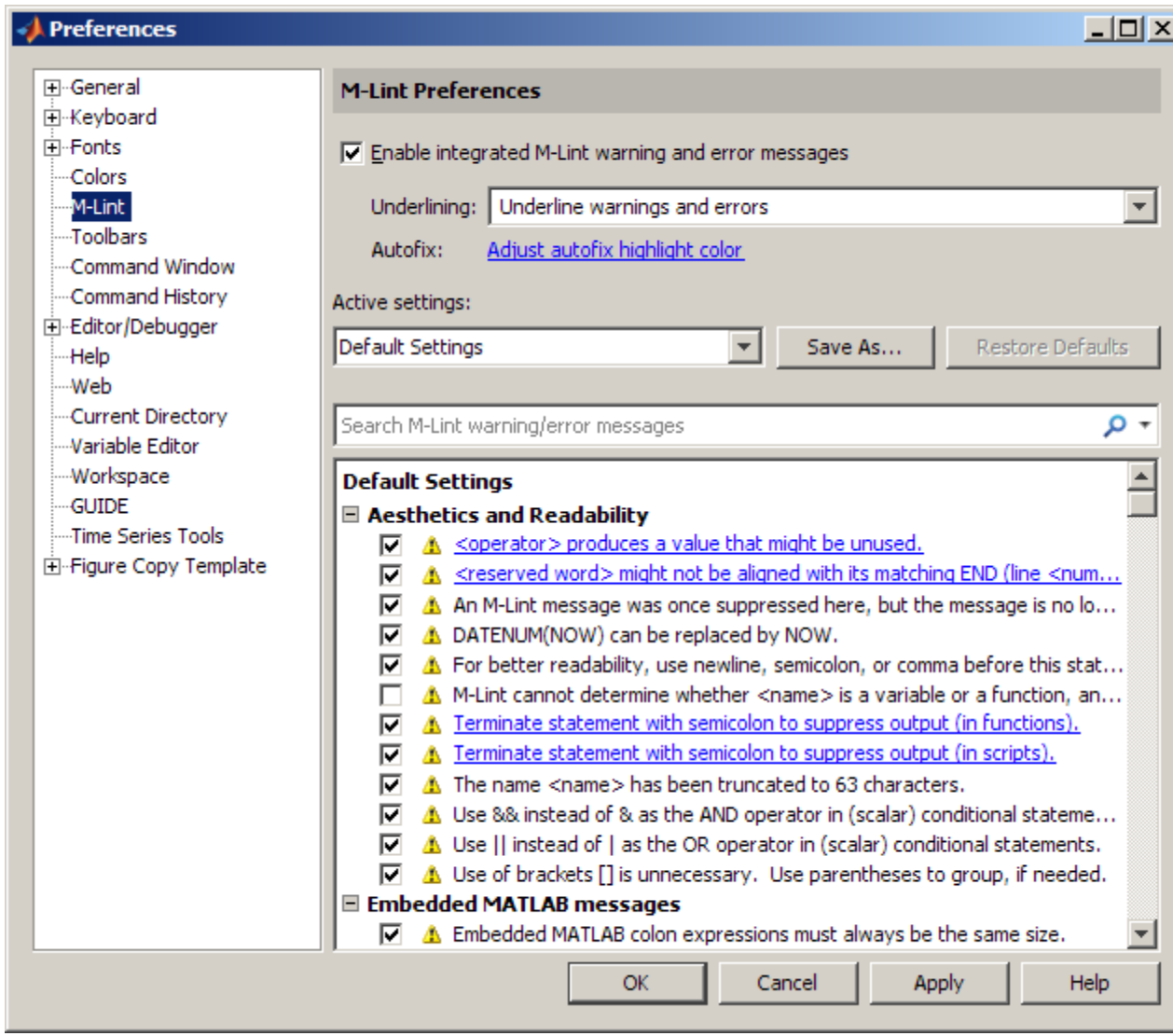
- Run a report for all files in a folder:
  1. In the Current Folder browser, click the **Actions** button  .
  2. Select **Reports > M-Lint Code Check Report**.
  3. Make any changes to your files based on the M-Lint messages in the report.

     For details, see "M-Lint Code Check Report" on page 9-22.
  4. Save the file.
  5. Rerun the report to see if your changes addressed the issues noted in the M-Lint messages.

Using M-Lint Automatic Code Analyzer in the Editor

To use the M-Lint continuous code checking in an M-file in the Editor, follow these steps:

1. Ensure the M-Lint messaging preference is enabled: Select **File > Preferences > M-Lint**, and then 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`.



2 Click **OK**.

3 Open an M-file in the Editor. This example uses the sample file `lengthofline.m` that ships with the MATLAB software:

a Open the example file:

```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...
     'examples','lengthofline.m'))
```

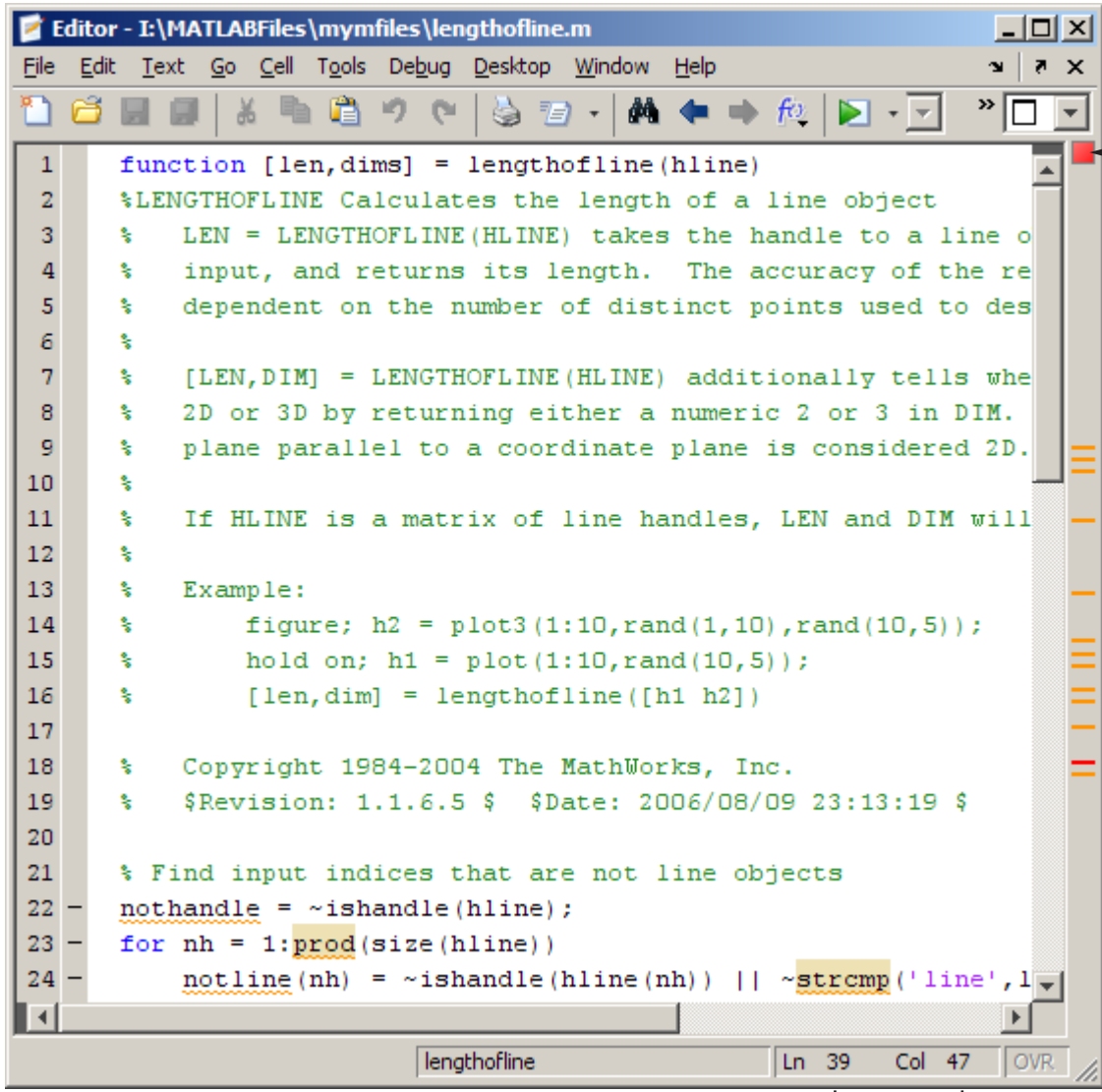
b Save the example file to a folder to which you have write access. For the example, `lengthofline.m` is saved to `I:\MATLABFiles\mymfiles`.

4 The M-Lint message indicator at the top right edge of the window conveys the M-Lint messages reported for the file:

- **Red** indicates 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** indicates warnings or opportunities for improvement, but no errors, were detected.
- **Green** indicates 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 file:
Red: Errors detected
Orange: Warnings or improvement opportunities detected
Green: No errors, warnings or improvement opprotunities detected
Click indicator to go to next line that has an associated M-Lint message.



Current cursor position

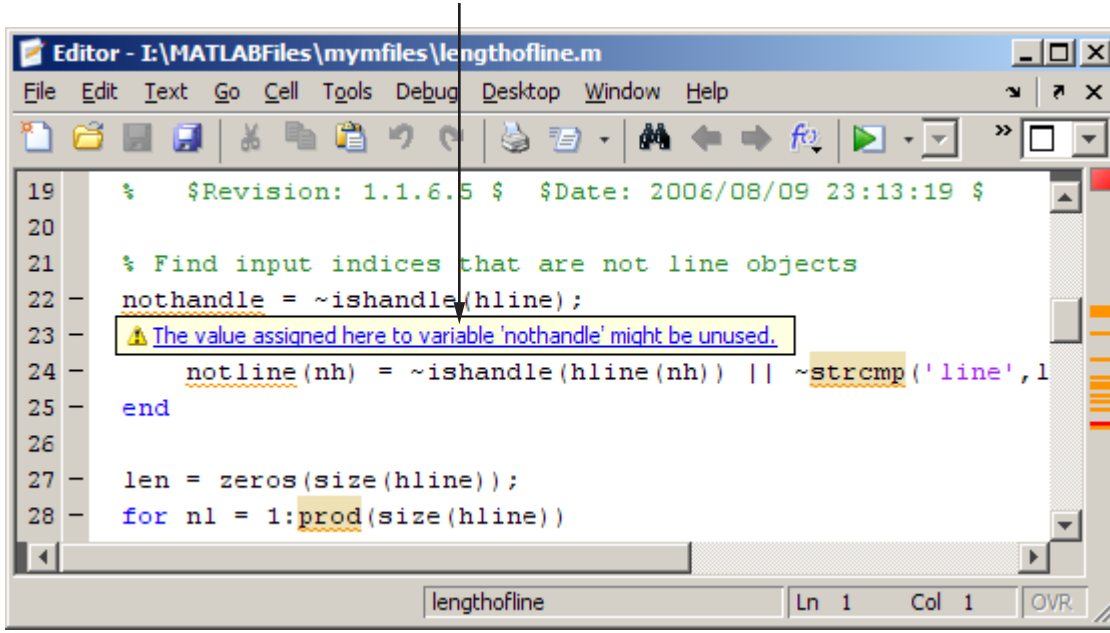
5 Click the M-Lint message indicator to go to the next code fragment containing an M-Lint message. The next code fragment is relative to the current cursor position, viewable in the status bar.

In the `lengthofline` example, the first message is at line 22. The cursor moves to the beginning of line 22.

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.

6 View the M-Lint message by moving the mouse pointer within the underlined fragment. The message appears with a yellow highlighted background, similar to data tips (see "Viewing Values as Data Tips in the Editor" on page 8-164).

Position cursor within orange underlined code fragment and the Editor displays the related M-Lint message.



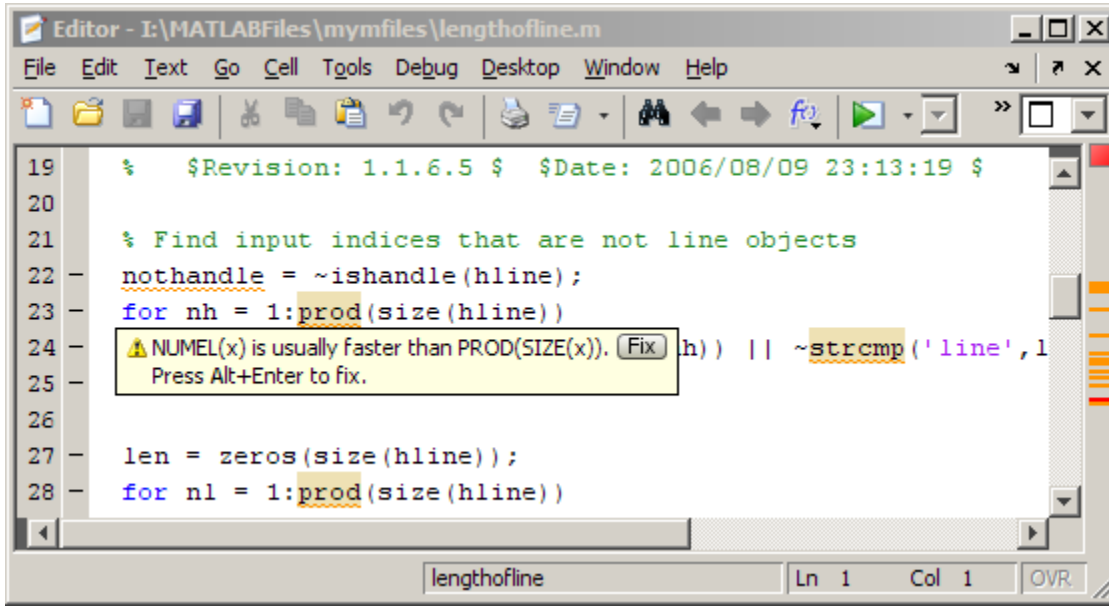
This message means that in line 22, `nothandle` is assigned a value, but is probably not used anywhere after that in the file. The line might be extraneous and you could delete it. But it might be that you actually

intended to use the variable, as shown in step 7 of this example. Notice that the message is a link to additional information. M-Lint message links are discussed in steps 9 and 10.

7 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.

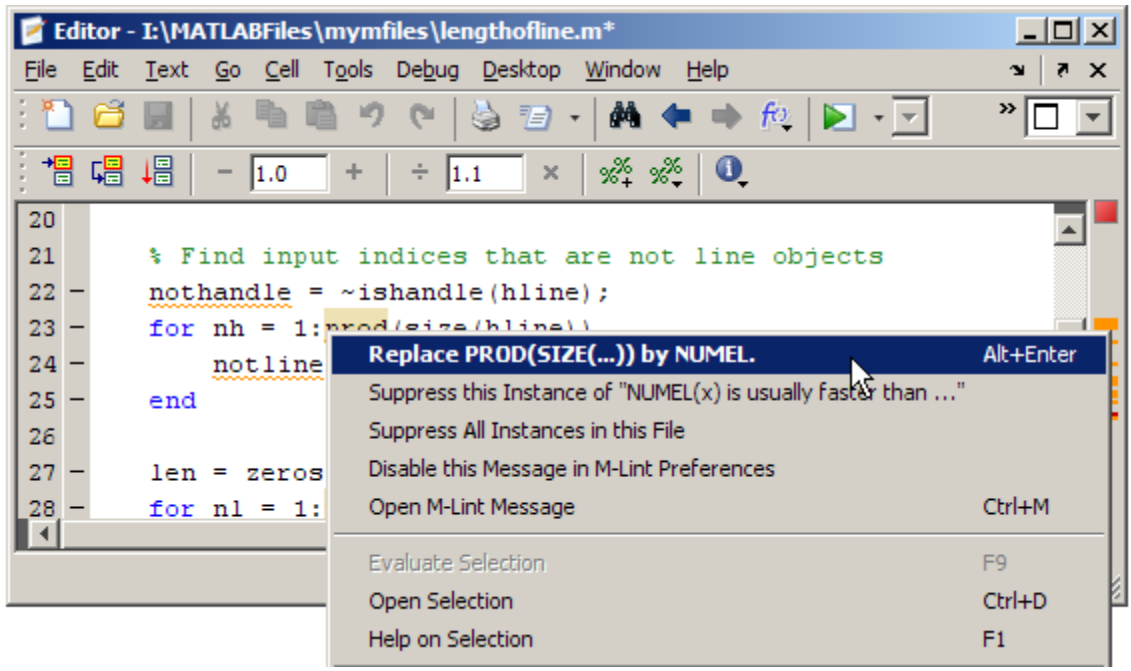
In this example, the intention was to use `nothandle` as a performance improvement by determining the value prior to the loop. Changing `~ishandle(hline(nh))` in line 24 to `nothandle(nh)` means there is no longer an M-Lint message associated with line 22.

8 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 provides a button to apply the automatic fix. It also indicates the keyboard controls (**Alt+Enter**) to apply it.

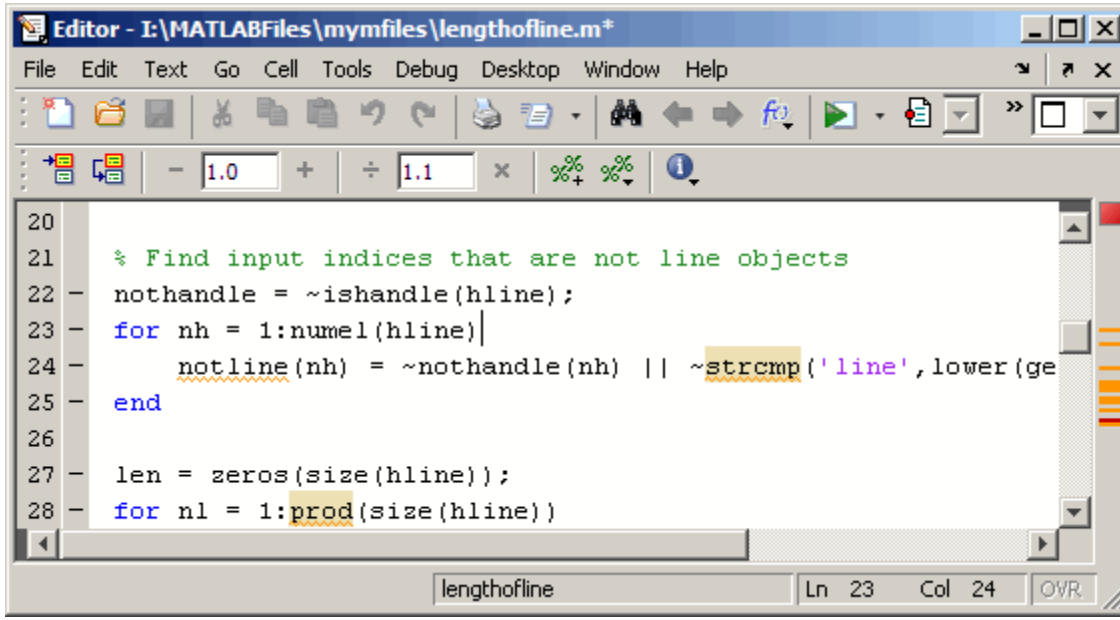


To view what the automated fix is, right-click the highlighted code (for a single-button mouse, use **Ctrl**+click). The first item in the context

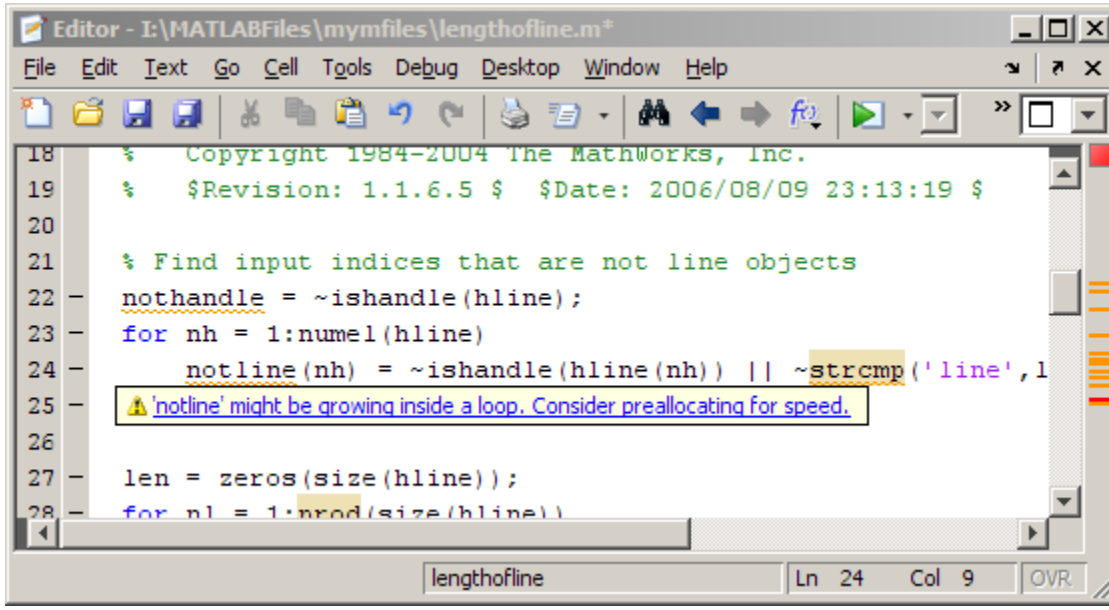
menu indicates the automatic fix that M-Lint can perform. Select it and M-Lint automatically corrects the code. In this example, M-Lint replaces `prod(size(hline))` with `numel(hline)`.



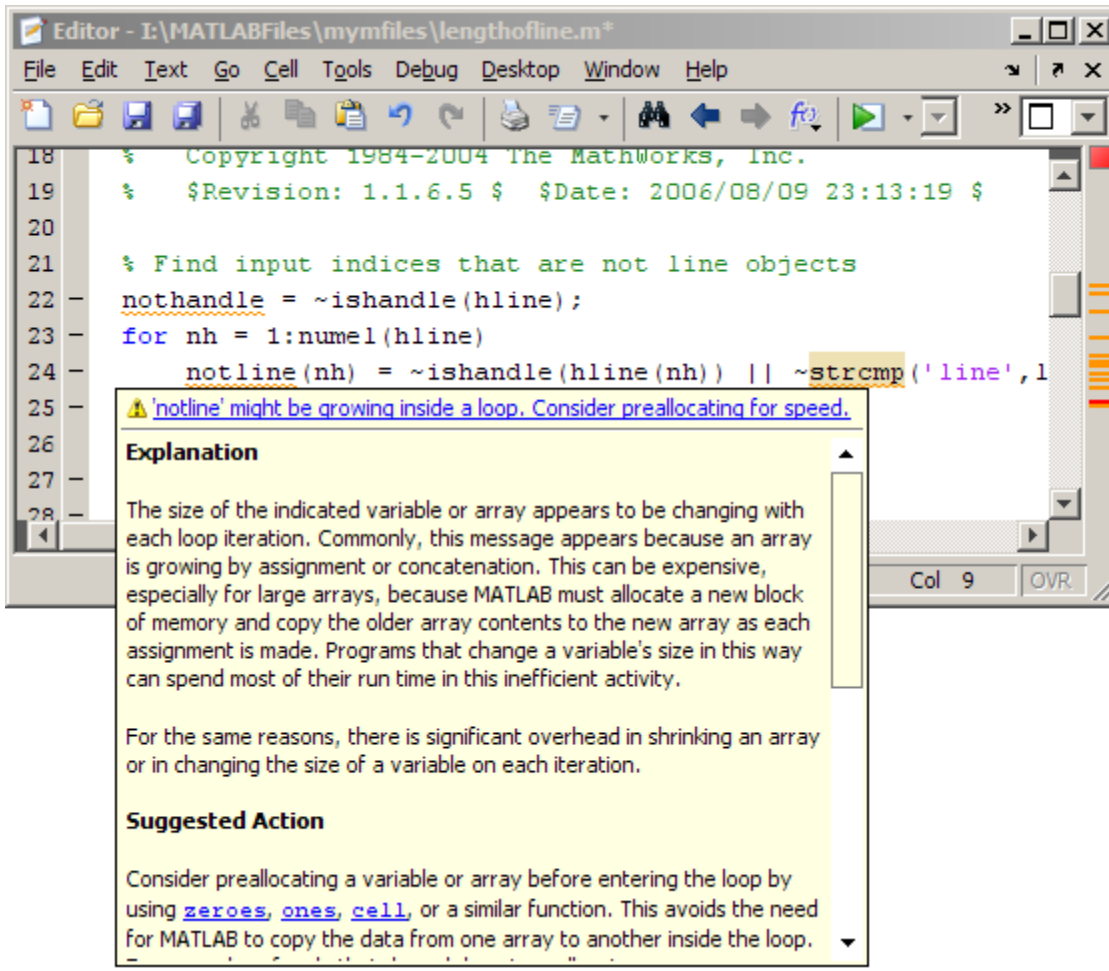
After you apply the fix, M-Lint removes the underline from `prod` in line 23.



9 Click the M-Lint message indicator to go to the next message, on line 24. When you view this M-Lint message, it appears as a link to indicate that there is more information about the message. Not all messages have additional information.



10 Click the link in the M-Lint Message. The window expands to display an explanation and user action.



11 Click the M-Lint message indicator to go to the next message, or use the M-Lint message bar.

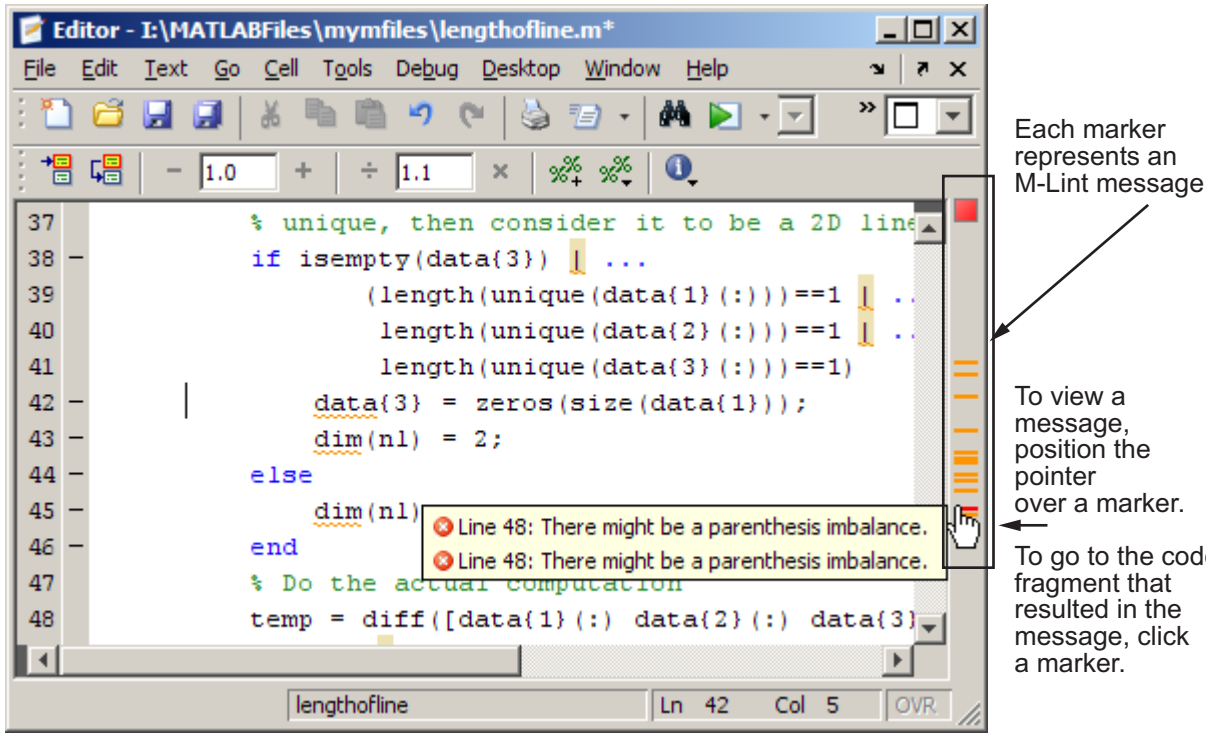
Each marker in the bar represents one or more lines that have associated M-Lint messages.

- Position the mouse 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 (error) 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. (There can be multiple messages per line.) 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.

```
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.



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 following these steps:

i Choose **File > Preferences > Keyboard > Delimiter Matching**, and then select **Match on arrow key**, if it is not already selected.

ii Move the arrow key over each of the delimiters to see if MATLAB indicates a mismatch.

   In the example, it may appear that there are no mismatched delimiters. However, M-Lint detects the semicolon in parentheses: `data{3}(;)`, and interprets it as the end of a statement. M-Lint reports that the two statements on line 48 each have a delimiter imbalance.

iii In line 48, change `data{3}(;)` to `data{3}(:)`.

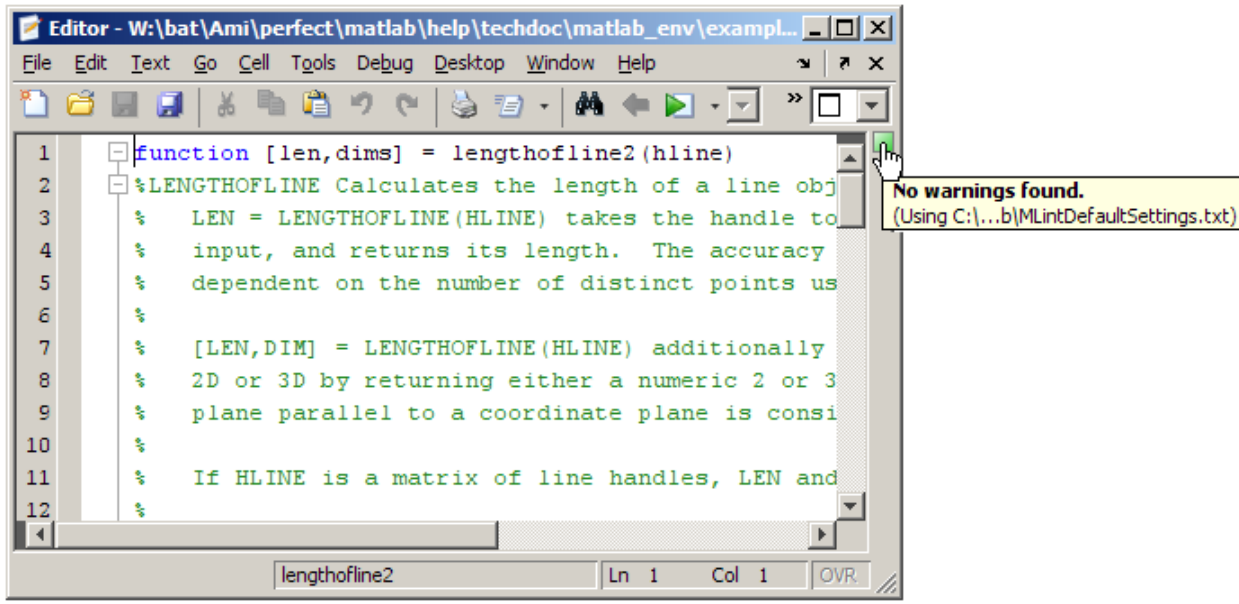
   When you make the change, the underline no longer appears in line 48. M-Lint now interprets the code at line 48 as a single statement. The single change addresses the issues in both of the M-Lint messages for line 48.

   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.

If there are multiple messages associated with a line, there might be multiple underlined code fragments that are adjacent, as in the previous step, making it difficult to display the message of interest. In these cases, it is easier to view the messages through the marker on the message bar than moving the arrow over each delimiter.

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`. Open the example file with the command:

```
open(fullfile(matlabroot,'help','techdoc',...
     'matlab_env', 'examples','lengthofline2.m'))
```



Suppressing M-Lint Indicators and Messages

Depending on the stage at which you are in completing the M-file, you might want to restrict the M-Lint underlining. You can do this by using the M-Lint preference referred to in step 1, in "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 8-119. For example, when first coding, you might prefer to underline only errors because warnings 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, and 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 (turn off) the M-Lint indicators for warning and error messages:

You cannot suppress M-Lint error messages such as syntax errors. Therefore, instructions on suppressing M-Lint messages do not apply to those types of messages.

Suppress an Instance of an M-Lint Message in the Current File

You can instruct M-Lint to suppress a specific instance of a message in the current file. For example, using the code presented in "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 8-119, follow these steps:

1. In line 49, right-click at the first M-Lint underline (for a single-button mouse, use **Ctrl**+click) .
2. From the context menu, select `Suppress this Instance of "Terminate statement with semicolon..."`.

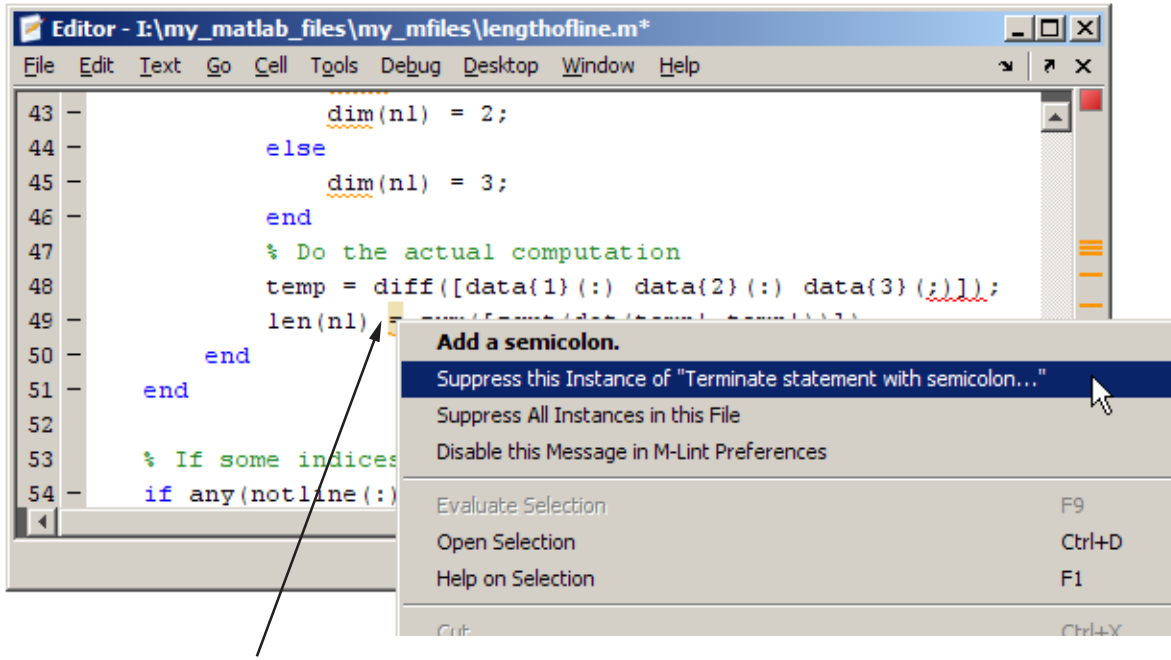
M-Lint adds a `%#ok<NOPRT>` to the end of the line, which instructs MATLAB not to check for a terminating semicolon at that line. M-Lint removes the underline and mark in the M-Lint indicator bar for that message.

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, in the code presented in "Checking M-File Code for Problems Using the

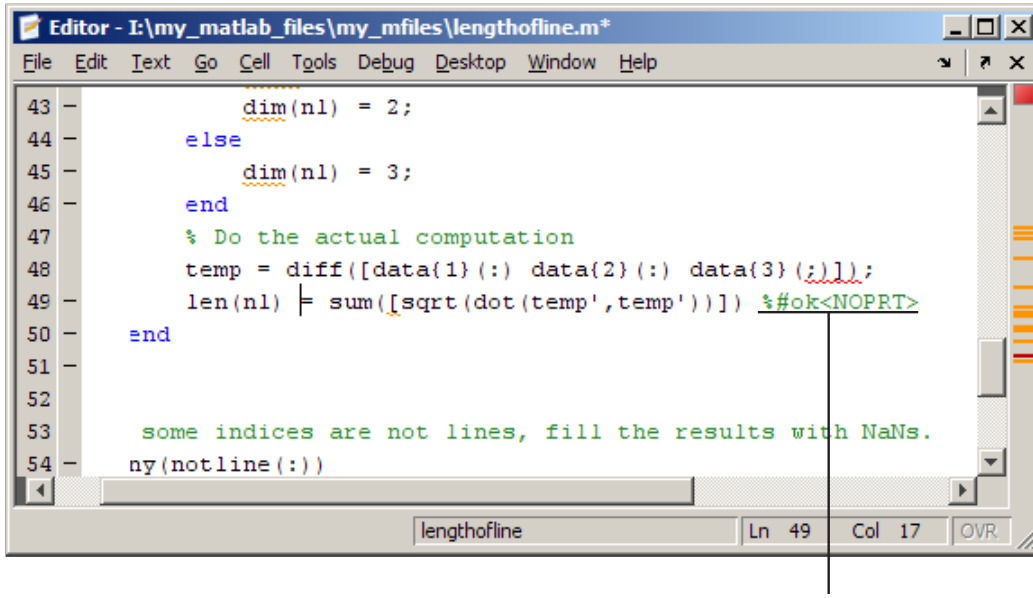
M-Lint Code Analyzer" on page 8-119, ignoring both messages for line 49 adds `%#ok<NBRAK,NOPRT>`.

Even if M-Lint preferences are set to enable this message, the specific instance of the message suppressed in this way does 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>` string from the line.

For more information about `%#ok`, see the `mlint` function reference page.



Right-click at an M-Lint underline and select the option instructing M-Lint to suppress only this instance of the message.



Suppress All Instances of an M-Lint Message in the Current File

You can instruct M-Lint to suppress all instances of a specific message in the current file. For example, using the code presented in "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 8-119, follow these steps:

1. In line 49, right-click at the first M-Lint underline (for a single-button mouse, use **Ctrl**+click).
2. From the context menu, select `Suppress all Instances in this File`.

M-Lint adds a `%#ok<*NOPRT>` to the end of the line, which instructs MATLAB to not check for a terminating semicolon throughout the file. M-Lint removes all underlines from the code, as well as marks in the M-Lint indicator bar that correspond to this message.

If there are two messages on a line that you do not want M-Lint to display any instances of in the current file, right-click separately at each underline, and then select the appropriate entry from the context menu. M-Lint expands the `%#ok` syntax. For the example, in the code presented in "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 8-119, ignoring both messages for line 49 adds `%#ok<*NBRAK,*NOPRT>`.

Even if M-Lint preferences are set to enable this message, the message does 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 in the file, delete the `%#ok<*NOPRT>` string from the line.

For more information about `%#ok`, see the `mlint` function reference page.

Suppress All Instances of an M-Lint Message in All Files

You can instruct M-Lint to disable all instances of an M-Lint message in all files. For example, using the code presented in "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 8-119, follow these steps:

1. In line 49, right-click at the first M-Lint underline (for a single-button mouse, use **Ctrl**+click).
2. From the context menu, select `Disable this Message in M-Lint Preferences`.

This modifies the M-Lint preference setting. For more information about M-Lint preferences, including how to restore MATLAB default settings, select **File > Preferences > M-Lint**, and then click **Help**.

Specify Default M-File Message Settings

You can specify that you want certain messages to be disabled by default when you open any M-file. Typically, you do this if you find that you do not want certain messages or categories of messages to be enabled for all or most of your M-files.

Follow these steps:

1. Select **File > Preferences > M-Lint**

The Preferences dialog box opens and displays the M-Lint Preferences panel.

2. Disable specific messages, or categories of messages.

   The **Active Settings** field now contains the value `Default Settings (modified)`.

3. Click Apply.

Now, each M-file you open uses the modified default settings. If you want to restore the factory-installed default settings, decide if you want to save the current settings to a file, as described in "Specify Nondefault M-File Message Settings" on page 8-137, and then click **Restore Defaults**.

For more information about M-Lint settings and preferences, click **Help** in the M-Lint preferences panel.

Specify Nondefault M-File Message Settings

You can specify that you want certain messages to be enabled or disabled, and then save those settings to a file. When you want to use a settings file with a particular M-file, you select it from the M-Lint preferences panel. That setting file remains in effect until you select another settings file. Typically, you change the settings file when you have a subset of M-files for which you want to use a particular settings file.

Follow these steps:

1. Select **File > Preferences > M-Lint**

   The Preferences dialog box opens and displays the M-Lint Preferences panel.

2. Enable or disable specific messages, or categories of messages.

3. Click **Save as** and save the settings to a `txt` file.

4. Click **OK**.

You can reuse these settings for any M-file, or provide the settings file to another user.

To use the saved settings:

1. In the Editor, select **Tools > M-Lint**.

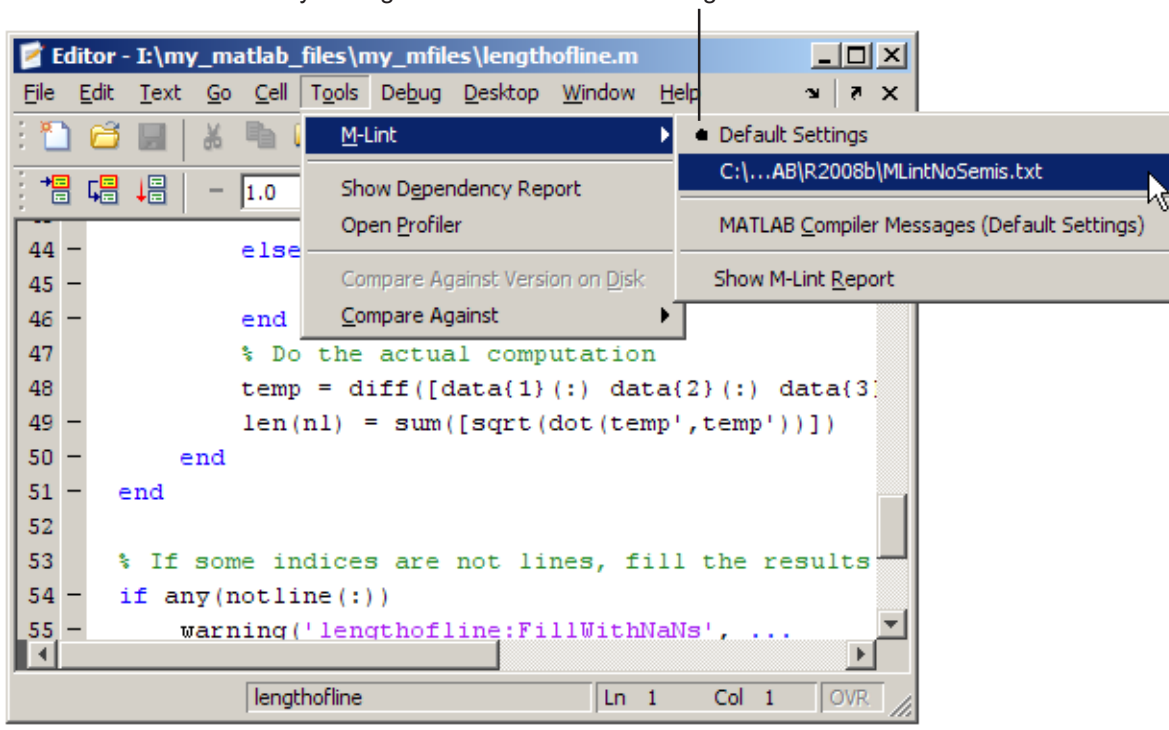
   The currently-selected setting choice displays, preceded by a bullet point.

2. Choose from any of the settings files, such as the `MLintNoSemis` example, as shown here.

   The settings you choose are in effect for all M-files until you select another set of M-Lint settings.

M-Lint default settings are currently selected (as indicated by the bullet preceding that menu item).

Select any settings file to use its M-Lint settings.



Understanding M-File Code Containing Suppressed M-Lint Messages

If you are given code that contains suppressed M-Lint messages, you might want to review those messages without the need to unsuppress them first. A messages might be in a suppressed state for any of the following reasons:

- One or more `%#ok<message-ID>` pragmas are on a line of code that elicits an M-Lint message specified by `<message-ID>`.
- One or more `%#ok<*message-ID>` pragmas are in a file that elicits the M-Lint message specified by `<message-ID>`.
- It is deselected in the M-Lint Preferences panel.

- It is disabled by default.

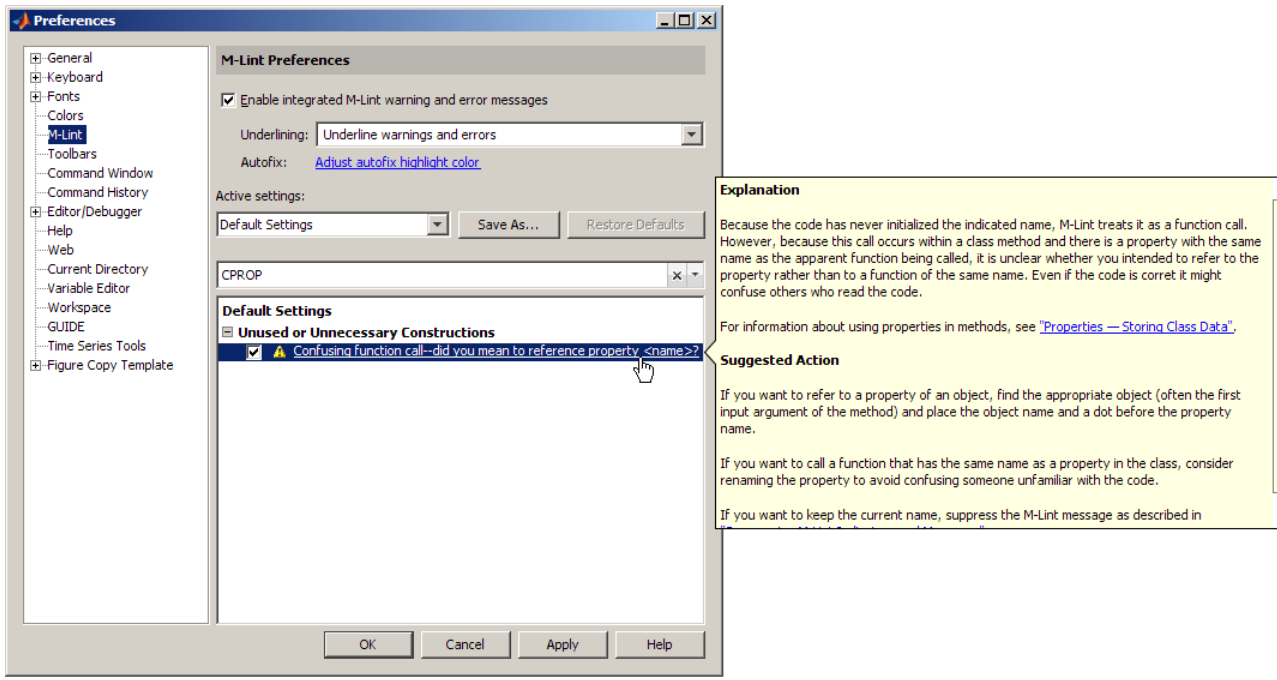
To determine the reasons why some M-Lint messages might be suppressed, follow these steps:

1. Search the M-file for the `%#ok` pragma and create a list of all the message-IDs associated with that pragma.

2. Open the M-Lint Preferences dialog box by choosing **File > Preferences > M-Lint**.

3. In the filter field, enter one of the message IDs, if any, you found in step 1.

   The message list now contains only the message that corresponds to that ID. If the message is a hyperlink, you can click it to see an explanation and suggested action for the message. This might provide insight into why the message is suppressed or disabled. The following image shows how the Preferences dialog box appears when you enter `CPROP` in the filter field, for example.



4. Click the button to clear the filter field, and then repeat step 3 for each message ID you found in step 1.

5. Click the button to clear the filter field.

6. Filter the messages to display those that are disabled by default and disabled via the Preferences panel by clicking the down arrow to the right of the filter field, and then clicking **Show Disabled Messages**.

   The message list now contains all the messages that are disabled in the Preferences panel.

7. Review the message associated with each message ID to understand why it might be suppressed in the code or disabled in Preferences.

Setting Preferences for M-Lint

Use M-Lint preferences to adjust how the M-Lint code analyzer operates. The preferences apply to M-Lint usage with the Editor, the Embedded MATLAB™

Editor (if you have products which use that tool), and the M-Lint Code Check Report, with a few exceptions that are noted. For more information about using M-Lint, see "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 8-119.

This section contains information about the following topics:

- Enabling integrated M-Lint warning and error messages in the Editor
- Restricting M-Lint Underlining in the Editor
- Changing the Color M-Lint Uses to Indicate an Automatic Fix Is Available
- Choosing the Messages to Display in Your Code
- Filtering the Messages Displayed in the Preferences Panel
- Saving Your Preferred Settings
- Using Your Saved Settings
- Using Default Settings
- "Enabling MATLAB Compiler Deployment Messages" on page 8-149

Enable integrated M-Lint warning and error messages

Select this check box if you want the Editor to show M-Lint messages in the M-file. This preference does *not* apply to the M-Lint Code Check Report. When you select this preference, the Editor provides visual cues that alert you to potential errors and problems, as well as opportunities for improvement in your code. These visual cues take the form of underlines and a message indicator bar. From these cues, you can view a message for each line of an M-file that M-Lint indicates might be improved. For example, a common M-Lint message is that a variable is defined but never used in the M-file.

Underlining. Restrict the M-Lint underlining notification using the associated drop-down list. The list is available only when **Enable integrated M-Lint warning and error messages** is selected. Options are

- `Underline warnings and errors`
- `Underline errors only`
- `No underlines`

Errors are underlined in red and warnings are underlined in orange. For all of the underlining options, the Editor provides the marks for errors and warnings in the indicator bar.

You might choose a different option at different stages in your workflow. For example, when first coding, you might prefer no underlines because they would appear as you enter a statement and might be distracting. Later, you might choose to underline only errors to help you debug your file. Finally, when tweaking an existing M-file, you might want to underline warnings and errors because the file is in a state that you can fix any issues you introduce at the time.

Autofix. Click **Adjust autofix highlight color** to open the Colors Preferences dialog box so that you can adjust the color that M-Lint uses to highlight errors and warnings that it can fix automatically. By default, this color is pale orange.

In the Color Preferences dialog box, in the **Other colors** group, use the M-Lint autofix highlight palette to select a color. For more information, see "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 8-119.

Active settings

When you have M-Lint warnings and error messages enabled, use M-Lint settings to show or hide specific messages for your code. If you are new to using M-Lint or MATLAB, use the default settings. After you are familiar with M-Lint and MATLAB, consider suppressing the display of certain M-Lint messages.

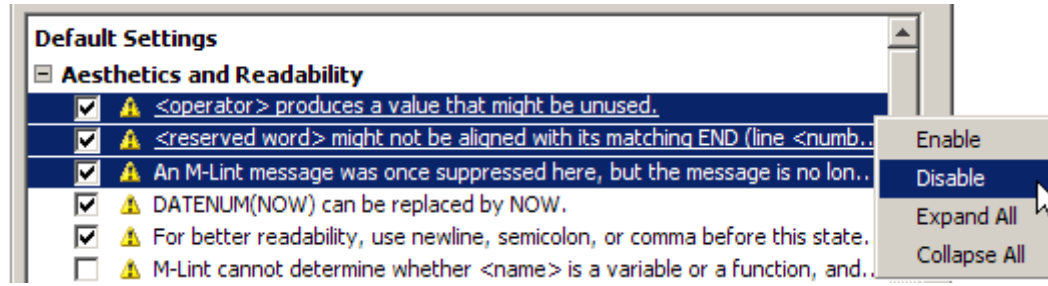
To suppress an M-Lint message on a line-by-line or file-by-file basis, see "Suppressing M-Lint Indicators and Messages" on page 8-132, or the `mlint` function.

To suppress messages in more than one file, it is more convenient to disable the M-Lint preference settings, as described here:

1. Use the filter field to filter the messages displayed in the Active settings table. For details, see "Filtering M-Lint Messages" on page 8-144.

2 Click the link for a given message (if available), to get more information about that message, including an explanation and suggested action.

3 In the Active settings table, select check boxes or clear check boxes to enable or disable messages, respectively. To enable or disable a set of messages simultaneously, highlight the messages in the Active settings table, right-click, and then choose **Enable** or **Disable**.



M-Lint shows enabled messages and hides disabled messages in the Editor.

4 Click **Apply** or **OK** to save the changes.

As with all preferences, MATLAB retains the settings for your next session.

Note The **MATLAB Compiler (deployment) messages** category is the only one that you can enable or disable by category. The M-Lint preferences panel only displays the **MATLAB Compiler (deployment) messages** category if you have MATLAB® Compiler™ installed.

Filtering M-Lint Messages. You can filter the list of messages in the Preferences dialog box to display only those messages that are currently of interest to you. Use any combination of the methods that the following table presents.

Note If you do not have the MATLAB Compiler installed, the M-Lint preferences panel does not display the **MATLAB Compiler (deployment) messages** category.

To see a list of messages that:	Perform this action:	Example Scenario
Contain a given string in the: • Short message • Extended message • Message category • Message ID	Type the string in the filter field.	You recall seeing a message containing a certain string that you want to review, but you cannot remember the exact message text. For example, type `com` in the filter field to display those messages that contain that string in the short message, extended message, or message ID.
Correspond to a given message ID	Type `msgid:` followed by the message ID in the filter field.	You are reviewing the code that someone else wrote and you want to see the message that corresponds to a suppressed one using the `%#ok<AGROW>` pragma. Type `msgid:agrow` in the filter field. The message corresponding to `AGROW` is a link. Click it for more information about the message. Not all M-Lint messages have additional information. These messages do not appear as links.
You can set using M-Lint Preferences	Click the down arrow to the right of the search field, and then click **Show All** .	You want, to see the complete list of messages after you have filtered the messages on a given string or filter menu option.

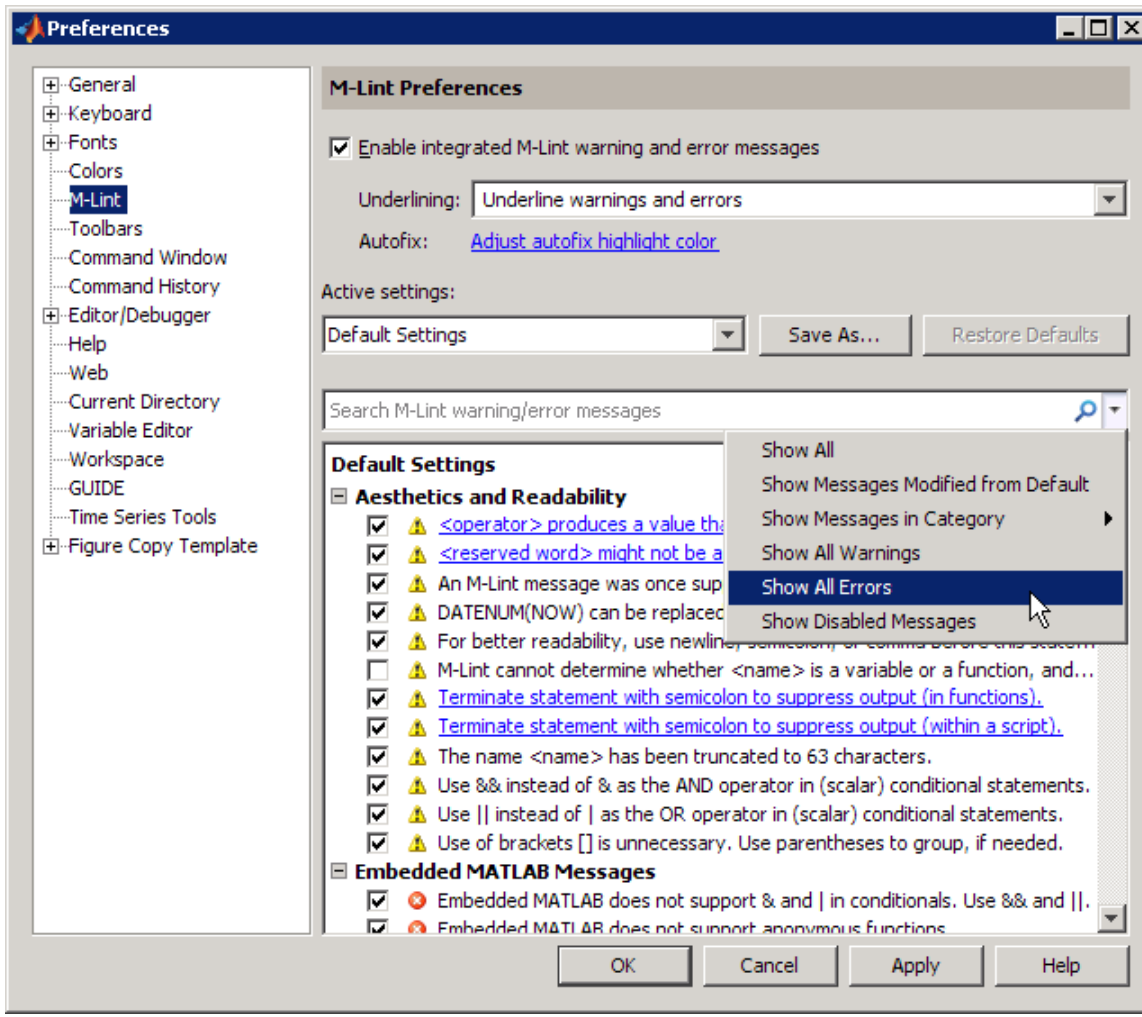
To see a list of messages that:	Perform this action:	Example Scenario
Are different from the default setting (of enabled or disabled)	Click the down arrow to the right of the search field, and then click **Show Messages Modified from Default**. A grey dot precedes a message with a setting different from the default. For example: 	A coworker gave you a settings file and you want to review each message that the coworker changed from its default setting.
Are in a given category	Click the down arrow to the right of the filter field, click **Show Messages in Category**, and then click the category you want.	You want to review M-Lint messages that describe coding practices that make it difficult for others to use your code. Click the down arrow to the right of the filter field, click **Show Messages in Category**, and then click **Aesthetics and Readability**. Click the messages that appear as links for more information. Not all M-Lint messages appear as links.
Are warnings	Click the down arrow to the right of the filter field, and choose **Show All Warnings**. An exclamation point in a yellow triangle indicates a warning message.	You recall previous warnings that your code generated, but you cannot remember enough details to use the filter field to find it. You want to skim all the warning messages to find a particular one of interest.

To see a list of messages that:	Perform this action:	Example Scenario
Are errors	Click the down arrow to the right of the filter field, and choose **Show All Errors**. By default, an X in a red dot indicates an error message,  .	You want to find a message that a script you worked on previously elicited. All you can recall is that it was an error and it involved `parfor`. Click the down arrow to the right of the filter field, and choose **Show All Errors**. Then, type a space and `parfor` in the filter field. The M-Lint preference panel displays only error messages that contain the word `parfor`.
Are disabled	Click the down arrow to the right of the filter field, and then choose **Show Disabled Messages**.	You want to see the messages that M-Lint disables by default or that you have previously disabled.

Example of Filtering Messages. To display error messages that contain the string `comma` and are disabled, do the following:

1. Click the arrow next to the filter field and select **Show All Errors**.

   The filter field contains the string `severity:error`.



2 At the end of the string `severity:error`, press the **Space** key, and then type `comma`.

3 Click the arrow next to the filter field and select **Show Disabled Messages**.

The filter field now contains the string, `severity:error comma enabled:false`. Only the M-Lint messages that fulfill those requirements display in the Preferences panel.

To restore the list of all M-Lint messages, click the  button.

Saving Settings. If you are likely to use different settings at different times, or if you want to make these settings available to other users, click **Save As**. MATLAB prompts you to provide the name of the `txt` file to which it will save the settings. The default location for settings is the MATLAB preferences folder (the folder returned when you run `prefdir`), although you can choose a different folder when saving.

Using Saved Settings. To use settings previously saved, select the settings `txt` file from the **Active settings** drop-down list. Or, select `Browse` to locate the settings file. Then click **Apply** or **OK** to make the settings take effect. You can also access saved settings from within the Editor using **Tools > M-Lint**, or the M-Lint message bar.

After selecting a settings file, you can modify the settings, but your changes automatically modify the `txt` file. If you want to retain the current settings in the `txt` file, create a copy of the settings file in which to make changes. To do so, click **Save As** and save the file to a different name. Then, make changes to the newly named file.

Default Settings. The **Active settings** indicator shows `Default Settings` when you are using the default settings rather than settings from a `txt` file. The term `(modified)` appears when you make changes to the default settings, but have not yet saved the changes to a file. To undo any unsaved changes and return to the default settings, click **Restore Defaults**. If you think you will use the modified default settings in a future session, save the settings as described in "Saving Settings" on page 8-149.

When a `txt` file appears in the Active settings field, return to default settings by using the drop-down list to select `Default Settings`.

Enabling MATLAB Compiler Deployment Messages

You can switch between showing or hiding Compiler deployment messages when you work on a file by changing the M-Lint preference for this category

of messages. Your choice likely depends on whether or not you are working on a file to be deployed. When you change the preference, it also changes the setting in the Editor. The converse is also true—when you change the setting from the Editor, it effectively changes this preference, however you will not see the changes reflected in the **M-Lint Preferences** if Preferences is open at the time you make the change. Whether or not you change the setting from the Editor or from **M-Lint Preferences**, it applies to the Editor and to the M-Lint Code Check Report.

To enable MATLAB Compiler deployment messages, follow these steps:

1. Choose **File > Preferences > M-Lint**.
2. Click the down arrow next to the filter field, and then choose **Messages in Category > MATLAB Compiler (deployment) messages.**
3. Click the **Enable Category** button.
4. Deselect individual message that you do not want M-Lint to display for your code (if any).

The settings `txt` file, which you can create as described in “Saving Settings” on page 8-149, includes the status of this setting.

Debugging Process and Features

In this section...

Ways to Debug M-Files

You can debug the M-files using the Editor, 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 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, the file is automatically saved before it runs. If you run a file with unsaved changes from the Command Window, MATLAB software runs the saved version of the file, so you will not see the results of your changes.

- Add the files to a folder on the search path or put them in the current folder — Be sure the file you run and any files it calls are in folders that are on the search path. If all files to be used are in the same folder, you can instead make that folder be the current folder.

Debugging Example — The Collatz Problem

The debugging process and features are best described using 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 folder or add the folder 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(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
```

Alternatively, you can open the files by issuing the following commands, and then save each file to a local folder:

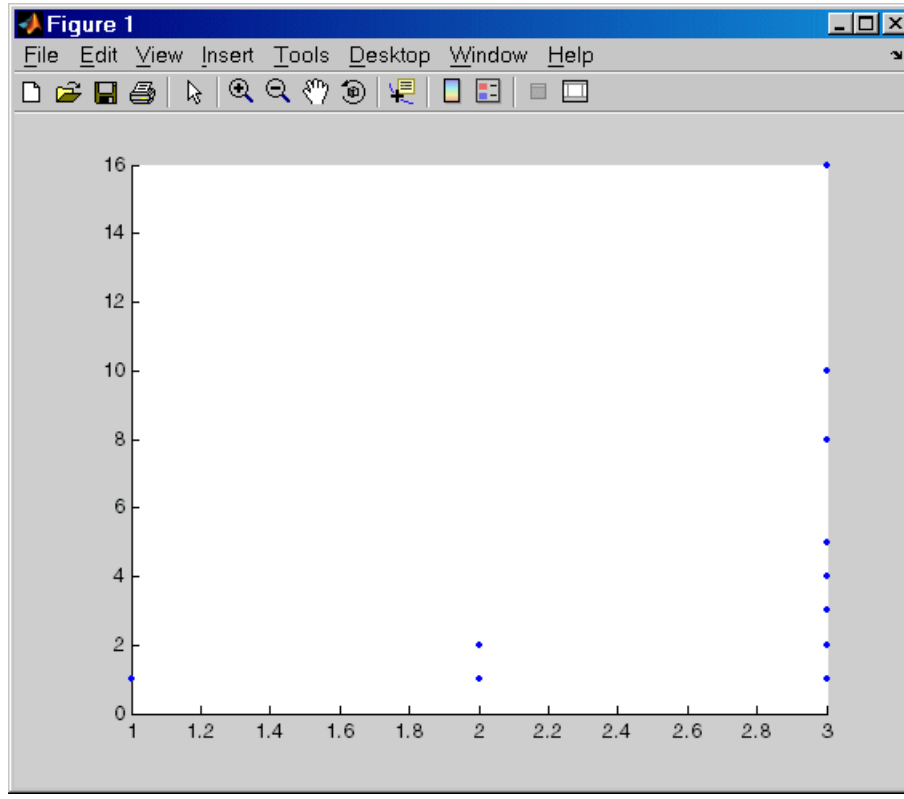
```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...
 'examples','collatz.m'))

open(fullfile(matlabroot,'help','techdoc','matlab_env',...
 'examples','collatzplot.m'))
```

Trial Run for Example. Open the file `collatzplot.m`. Make sure the current folder is the folder 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 can be. You can set breakpoints in the Editor, 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 8-156.
- A conditional breakpoint, which stops at a specified line in an M-file only under specified conditions. For details, see "Conditional Breakpoints" on page 8-176.
- 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 8-180.

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 8-170. 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 folder or in folders on the search path. When you add or remove a breakpoint in a file that is not in a folder on the search path or in the current folder, a dialog box appears. This dialog box presents options that allow you to add or remove the breakpoint. You can either change the current folder to the folder containing the file, or you can add the folder 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:

1. If you have changed the file, save it.
2. Click in the breakpoint alley at an executable line where you want to set the breakpoint.
   - The *breakpoint alley* is the narrow column on the left side of the Editor, to the right of the line number.
   - Executable lines are preceded by a - (dash).

     If you attempt to set breakpoints at lines that are not executable, such as comments or blank lines, MATLAB sets it at the next executable line.

Other ways to set a breakpoint are to:

- Position the cursor in an executable line and then click the Set/clear breakpoint button  on the toolbar.
- From the **Debug** or context menu, select **Set/Clear Breakpoint**.

Setting Breakpoints for the Example. It is unclear whether the problem in the example is in `collatzplot` or `collatz`. To start, follow these steps:

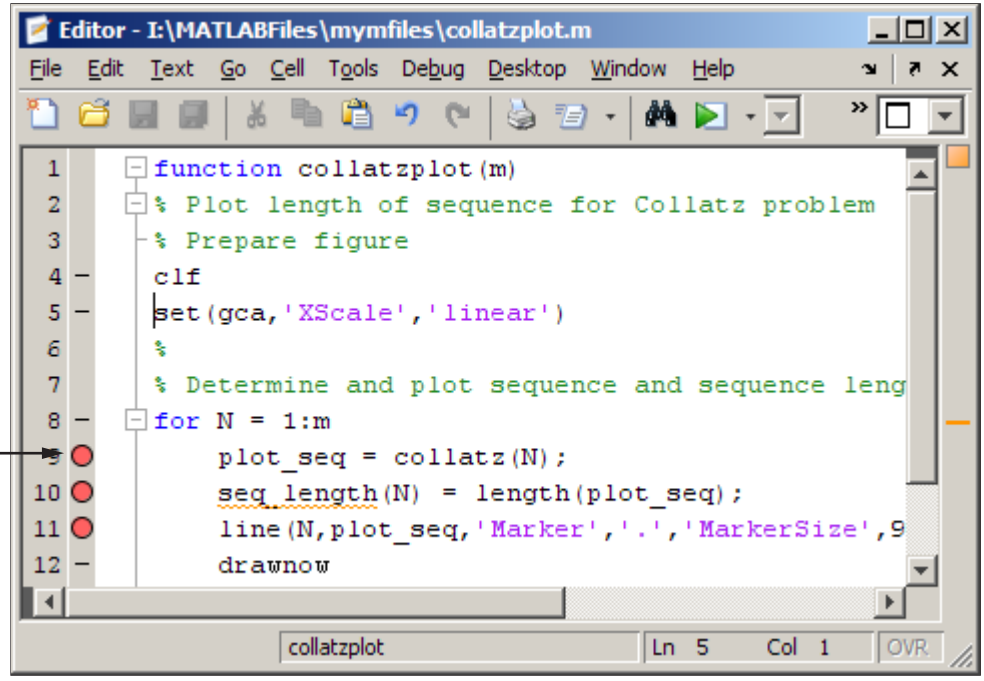
1. Set a breakpoint in `collatzplot.m` at line 9.

   This breakpoint enables you to step into `collatz` to see if the problem is there.
2. Set additional breakpoints at lines 10 and 11.

   These breakpoints stop the program so you can examine the interim results.

Click where there is a dash (-) to set a breakpoint at that line.

A red icon indicates a valid breakpoint is set.

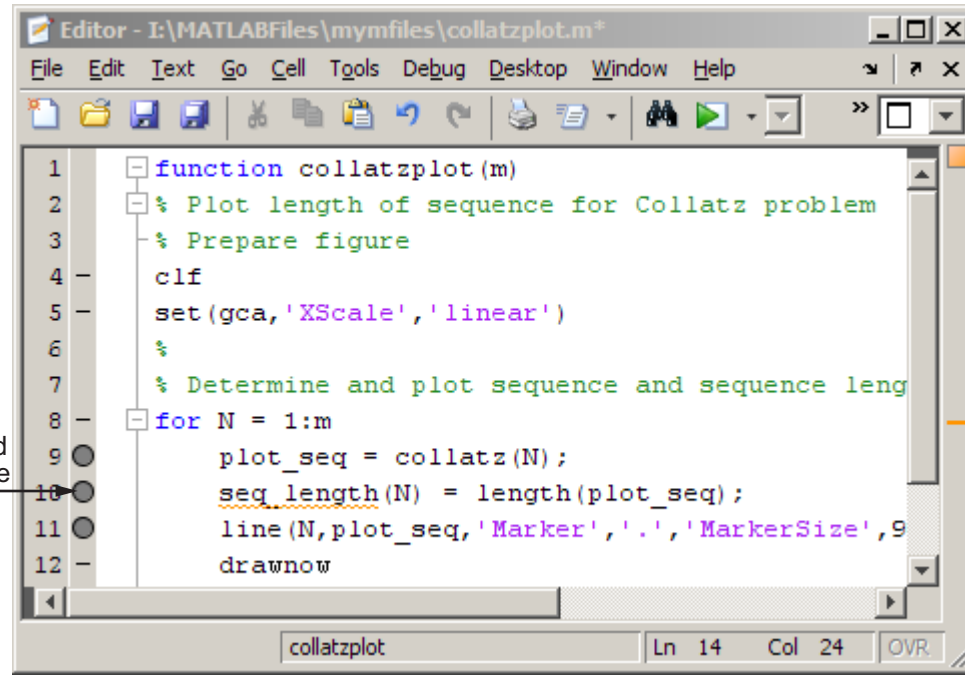


Understanding Valid (Red) and Invalid (Gray) Breakpoints. Red breakpoints indicate valid standard breakpoints. If breakpoints are gray, they are not valid.

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)



Breakpoints are gray for either of these reasons:

- There are unsaved changes in the file. Save the file to make breakpoints valid. The gray breakpoints become red, indicating they are now valid. Any gray breakpoints that you entered at invalid breakpoint lines automatically move to the next valid breakpoint line with a 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 9
dbstop in collatzplot at 10
dbstop in collatzplot at 11
```

Running an M-File with Breakpoints

After setting breakpoints, run the M-file from the Command Window or the Editor.

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 or from a run configuration with a value specified.

Results of Running an M-File Containing Breakpoints

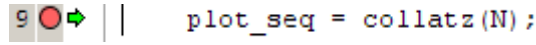
Running the M-file results in the following:

- The prompt in the Command Window changes to

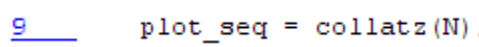
  ```
  K>>
  ```

  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 by the green arrow just to the right of the breakpoint, which in the example, is line 9 of `collatzplot` as shown here.

  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

- 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 folder or a folder on the search path, you are prompted to either add the folder to the path or change the current folder.

In debug mode, you can set breakpoints, step through programs, examine variables, and run other functions.

Note that MATLAB software 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 or desktop, or use the equivalent functions.

Toolbar Button	Debug Menu Item	Description	Function Alternative
	**Run** *m-file* or **Run Configuration for** *m-file*	Commence execution of M-file and run until completion or until a breakpoint is encountered. The **Run Configurations for** *m-file* menu item provides a submenu that enables you to select a particular run configuration or to edit the run configurations for the M-file. If you choose **Run** *m-file*, MATLAB uses the default run configuration.	None

Toolbar Button	Debug Menu Item	Description	Function Alternative
None	**Go Until Cursor**	Continue execution of M-file until the line where the cursor is positioned. Also available on the context menu.	None
	**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`
	**Continue**	Resume execution of M-file until completion or until another breakpoint is encountered.	`dbcont`
	**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 9. 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 9, 10, and 11. Now the program is again paused at the breakpoint at line 9.

Stepping In to Called Function in the Example

Now that `collatzplot` is paused at line 9 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 9 of `collatzplot` goes to line 9 of `collatz`. If `collatz` is not open in the Editor, it automatically opens if you have selected **Debug > Open Files When Debugging**.

The pause indicator at line 9 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:

Many of these methods are used in "Examining Values in the Example" on page 8-167.

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:

- To display the call stack, use `dbstack`.
- To change to a different workspace, use `dbup` and `dbdown`.
- To list the variables in the current workspace, use `who` or `whos`.

Workspace in the Example. At line 9 of `collatzplot`, you stepped in, and the current line is 9 in `collatz`. The **Stack** shows that `collatz` is the current workspace.

Viewing Values as Data Tips in the Editor

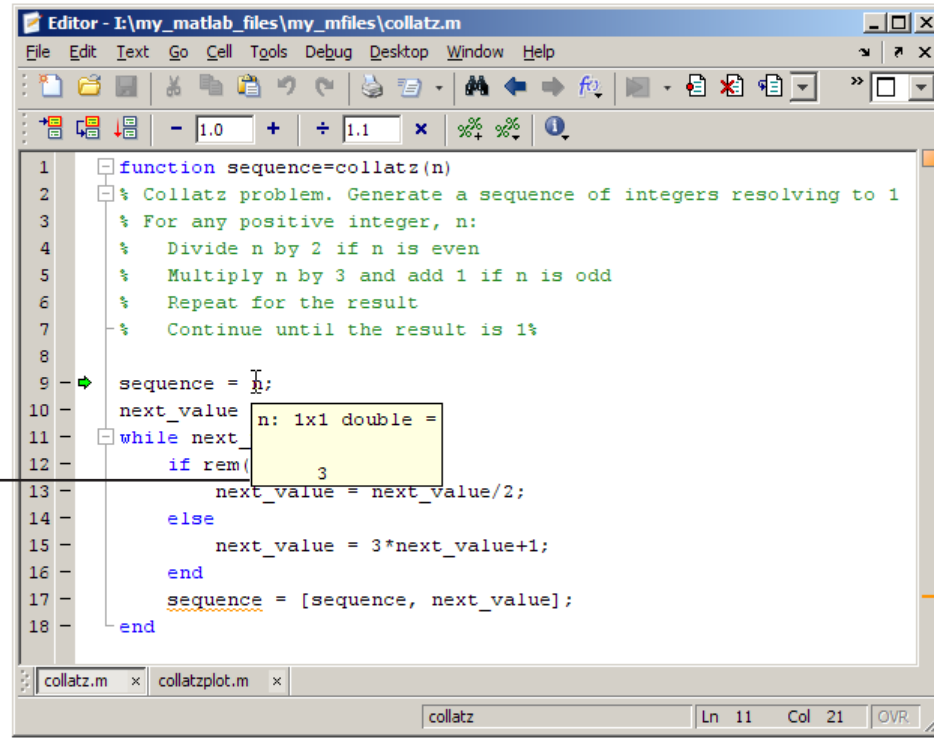
In the Editor, position the mouse pointer to the left of a variable. Its current value appears—this is called a data tip, which is like a Tooltip for data. The data tip stays in view until you move the pointer. If you have trouble getting the data tip to appear, click in the line containing the variable and then move the pointer next to the variable.

A related function is `datatipinfo`.

Data Tips in the Example. Position the mouse pointer over `n` in line 9 of `collatz`. The data tip shows that `n = 2`, as expected.

Hold the mouse pointer over a variable.

Its current value temporarily displays as a data tip.



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 it displays the variable's current value. For the example, to see the value of `n`, type

```
n
```

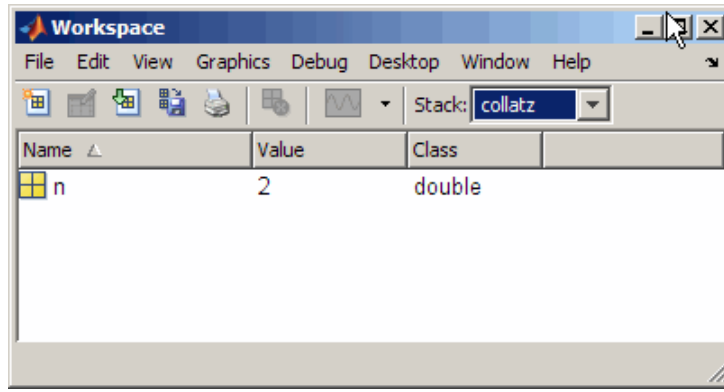
The Command Window displays the expected result

```
n =
 2
```

and displays the debug prompt, `K>>`.

Viewing Values in the Workspace Browser and Variable 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.

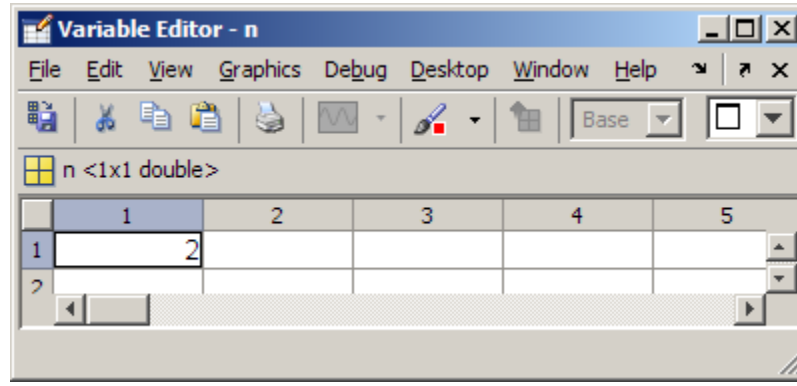


The **Value** column does not show all details for all variables. To see details, double-click a variable in the Workspace browser. The Variable Editor opens, displaying the content for that variable. You can open the Variable Editor directly for a variable using `openvar`.

To see the value of `n` in the Variable Editor for the example, type

```
openvar n
```

and the Variable Editor opens, showing that `n = 2` as expected.



Evaluating a Selection

Select a variable or equation in an M-file in the Editor. Right-click and select **Evaluate Selection** from the context menu (for a single-button mouse, use **Ctrl**+ click). The Command Window displays the value of the variable or equation. 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 9 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 10, and then line 11. The variable `seq_length` in line 10 is a vector with the elements:

```
1  2
```

which is correct.

Finally, step again to advance to line 12. Examining the values in line 11, `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.

Problems Viewing Variable Values from Parent Workspace

Sometimes, if you set a breakpoint in a function, and then attempt to view the value of a variable in the parent workspace using the `dbup` command, the value of the variable is currently under construction. Therefore, the value is not available. This is true whether you view the value by specifying the `dbup` command in the Command Window or by using the **Stack** field on the Editor toolbar.

In such cases, MATLAB returns the following message, where *x* is the variable for which you are trying to examine the value:

```
K>> x
??? Reference to a called function result under construction x.
```

For example, suppose you have code such as the following:

```
x = collatz(x);
```

MATLAB detects that the evaluation of `collatz(x)` replaces the input variable, *x*. To optimize memory use, MATLAB overwrites the memory that *x* currently occupies to hold a new value for *x*. When you request the value of *x*, and it is under construction, its value is not available, and MATLAB displays the error message.

Correcting Problems and Ending Debugging

These are some of the ways to correct problems and end the debugging session:

Many of these features are used in "Completing the Example" on page 8-172.

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 Variable 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 "Understanding Valid (Red) and Invalid (Gray) Breakpoints" on page 8-158 for details.

If you attempt to save an edited M-file while in debug mode, a dialog box opens 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.

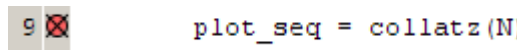
After quitting debugging, pause indicators in the Editor display no longer appear, and the normal prompt `>>` appears in the Command Window instead of the debugging prompt, `K>>`. You can no longer access the call stack.

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 8-176.

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.



When you run `dbstatus`, the resulting message for a disabled breakpoint is

```
Breakpoint on line 9 has conditional expression 'false'.
```

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.

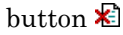
Clearing (Removing) Breakpoints. All breakpoints remain in a file until you clear (remove) them or until they are automatically cleared. Clear a breakpoint after determining that a line of code is not causing a problem.

To clear a breakpoint in the Editor:

- Click anywhere in a line that has a breakpoint and select **Set/Clear Breakpoint** from the **Debug** or context menu.
- Click a standard breakpoint icon, or a disabled conditional breakpoint icon.
- Use the `dbclear in m-file at lineno` command in the Command Window. For the example, clear the breakpoint at line 9 in `collatzplot` by typing:

  ```
  dbclear in collatzplot at 9
  ```

To clear all breakpoints in all files:

- Select **Debug > Clear Breakpoints in All Files** on the toolbar.
- Click the Clear breakpoints in all files button 
- Use `dbclear all` in the Command Window.

  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 "Understanding Valid (Red) and Invalid (Gray) Breakpoints" on page 8-158 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 11, 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
```

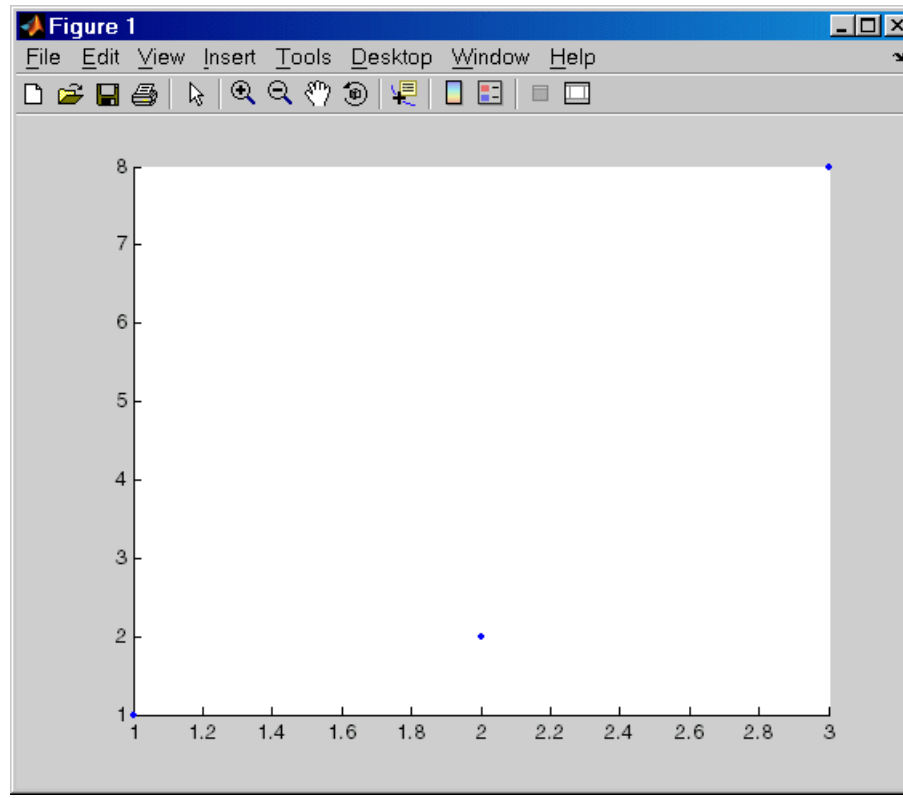
in the Command Window.

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)
```

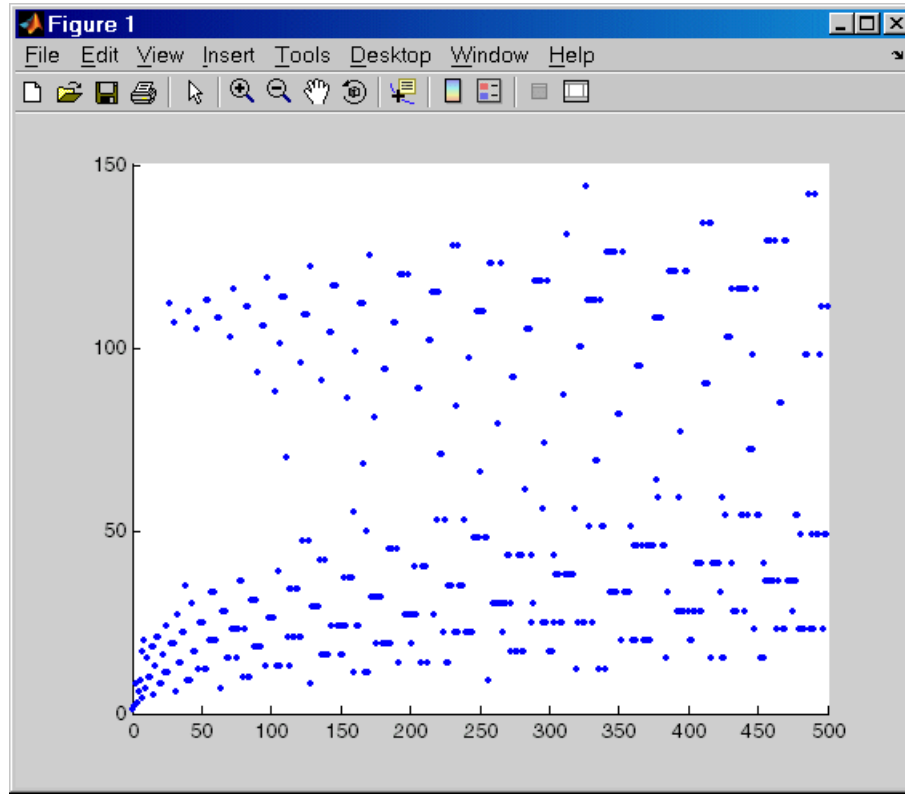
7 To make debugging easier, you ran `collatzplot` for a small value of `m`. Now that you know it works correctly, run `collatzplot` for a larger value to produce more interesting results. Before doing so, you might want to disable output for the line you just added in step 6, line 19 of `collatz.m`, 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

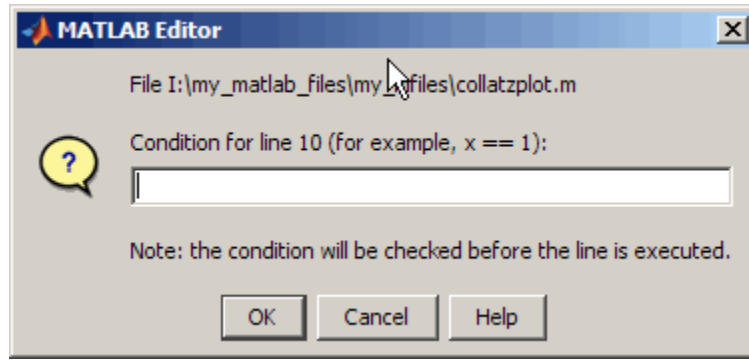
Set conditional breakpoints to cause MATLAB to stop at a specified line in a file only when the specified condition is met. One particularly good use for conditional breakpoints is when you want to examine results after a certain number of iterations in a loop. For example, set a breakpoint at line 10 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

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.

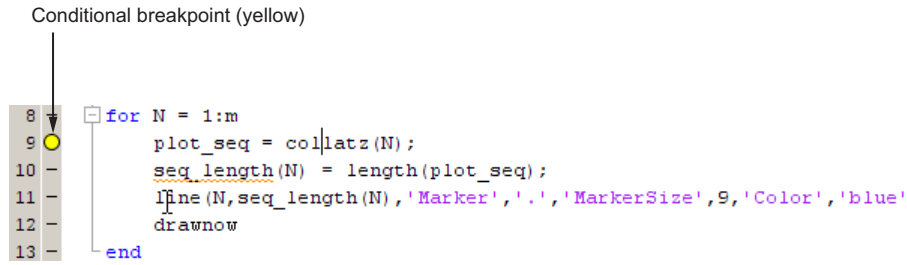
The **MATLAB Editor** conditional breakpoint dialog box opens as shown in this example.



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 9 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.



When you run the file, MATLAB software 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 9 when `N` is `2`. If you continue executing, MATLAB pauses again at line 9 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

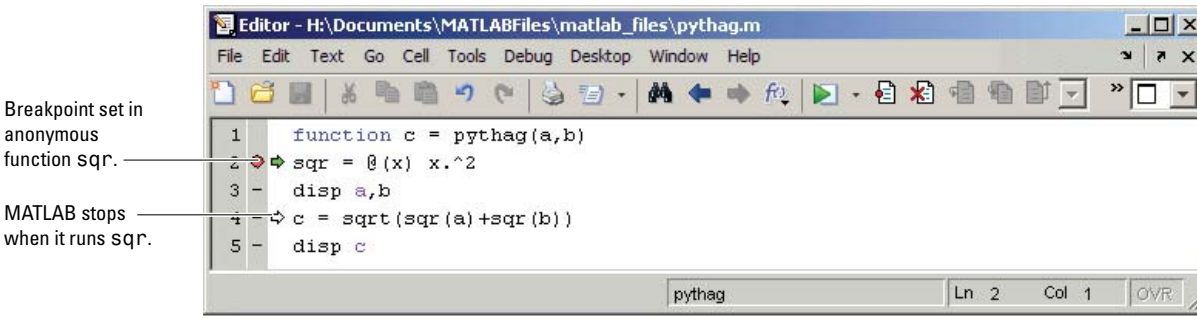
There can be multiple breakpoints in an M-file line that contains anonymous functions. There can be a breakpoint for the line itself (MATLAB software stops at the start of the line), as well as a breakpoint for each anonymous function in that line. When you add a breakpoint to a line containing an anonymous function, the Editor 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 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.



Breakpoints in Methods that Overload Functions

MATLAB functions often call other MATLAB functions and methods to perform their operations. If you set a breakpoint in a class method, and then run a MATLAB function that results in calling that method, execution stops at the breakpoint. This behavior can be confusing if you are unaware that the MATLAB function calls the method containing the breakpoint.

For instance, suppose you do the following:

1. Define a class named `MyClass` that overloads the MATLAB `size` function.
2. Create an instance of `MyClass`.
3. Insert breakpoints within the `MyClass size` method.
4. Call `whos`.

When you call the `whos` function, it calls the `size` function to obtain size information about the variables in the workspace. Under the preceding circumstances, because `MyClass` overloads the `size` function, `whos` calls the `MyClass size` method instead of the default `size` function to determine the size of the `MyClass` object. Execution stops at the breakpoint you set in the `size` method. You can enable the MATLAB function to execute to completion by either stepping or continuing through the method. To prevent this behavior from recurring, remove the breakpoints.

Error Breakpoints

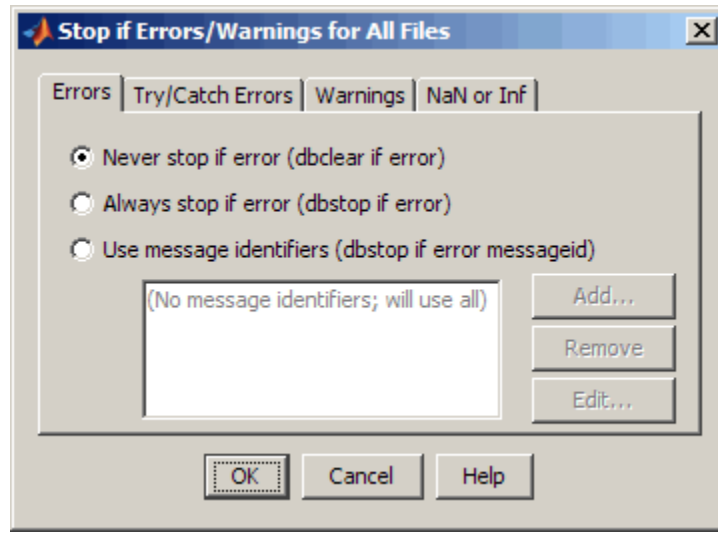
Set error breakpoints to stop program execution and enter debug mode when MATLAB encounters a problem. Unlike standard and conditional breakpoints, you do not set these breakpoints at a specific line in a specific file. Rather, once set, MATLAB stops at any line in any file when the error condition specified by using 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 you select **Debug > Open Files when Debugging**. 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 and Clearing Error Breakpoints

To set error breakpoints:

1. Select **Debug > Stop if Errors/Warnings**.
2. In the **Stop if Errors/Warnings for All Files** dialog box, specify error breakpoints on all appropriate tabs, and then click **OK**.

To clear error breakpoints, select the **Never stop if ...** option for all appropriate tabs, and then click **OK**.



Error Breakpoint Types and Options

As the tabs in the **Stop if Errors/Warnings for All Files** dialog box suggest, there are four basic types of error breakpoints you can set:

- **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 occurs, 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. MATLAB pauses immediately after the line that encountered the value. You can resume execution or use debugging features.

Select options for these error breakpoints, as follows:

- Select the **Never stop if ...** on a tab to clear that type of breakpoint.
- Select **Always stop if ...** on a tab to set that type of breakpoint.
- Select **Use message identifiers ...** on a tab to limit each type of error breakpoint (except NaN or Inf). Execution stops only for the error you specify by the corresponding message identifier.

  This option is not available for the **NaN or Inf** type of error breakpoint. You can add multiple message identifiers, and edit or remove them.

Examples of Setting Warning and Error Breakpoints

Pausing Executing for Warnings. To pause execution when MATLAB produces a warning:

1. Select the **Warnings** tab.
2. Select **Always stop if warning**, and then click **OK**.

Now, when you run an M-file and MATLAB produces a warning, execution pauses andMATLAB enters debug mode. The file opens in the Editor at the line that produced the warning.

Setting Breakpoints for a Specific Error. To stop execution for a specific error add a message identifier:

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, type the message identifier of the error for which you want to stop. The identifier is of the form `component:message` (for example, `MATLAB:nargchk:notEnoughInputs`). Then click **OK**.

   The message identifier you specified appears in the **Stop If Errors/Warnings for All Files** dialog box.

5 Click **OK**.

Obtaining Error Message Identifiers. To obtain an error message identifier generated by a MATLAB function, run the function to produce the error, and then call MExeption.last. For example:

```
surf
MException.last
```

The Command Window displays the MException object, including the error message identifier in the `identifier` field. For this example, it displays:

```
ans =

  MException

  Properties:
    identifier: 'MATLAB:nargchk:notEnoughInputs'
       message: 'Not enough input arguments.'
         cause: {}
         stack: [1x1 struct]

  Methods
```

Obtaining Warning Message Identifiers. To obtain a warning message identifier generated by a MATLAB function, run the function to produce the warning. 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, to the right of each option. For example, the function equivalent for **Always stop if error** is `dbstop if error`. Use these functions in the command Window as follows:

- `dbstop` with appropriate arguments to set error breakpoints
- `dbclear` to clear error breakpoints
- `dbstatus` to view the error breakpoints currently set.

  The `dbstatus` output lists error breakpoints in the `cond` field and message identifiers for breakpoints in the `identifier` field.

Debugging Functions

The debug functions are:

- `dbstop`—Set breakpoint.
- `dbclear`—Remove breakpoint.
- `dbcont`—Resume execution.
- `dbdown`—Change local workspace context.
- `dbmex`—Enable MEX-file debugging.
- `dbstack`—Function call stack.
- `dbstatus`—List breakpoints.
- `dbstep`—Execute one or more lines
- `dbtype`—List M-file with line numbers.
- `dbup`—Change local workspace context.
- `dbquit`—Quit debug mode.

Using Cells for Rapid Code Iteration and Publishing Results

In this section...

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* refers to a section of code. A cell contains the contiguous lines of code that you want to evaluate as a whole in an M-file script. A cell has boundaries to define its start and end. Because cell features operate on cells, it is important to understand how you define boundaries explicitly, how MATLAB defines boundaries implicitly, and how implicitly and explicitly defined cell boundaries interact to create cells, as described in "Defining Cells" on page 8-188

Specifically, MATLAB software uses cells for:

- Rapid code iteration in the Editor — This makes the experimental phase of your work with M-file scripts easier. The next section, "Rapid Code Iteration Overview" on page 8-186, 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 also requires you to define cells. You can make use of the cell navigation and evaluation you specify for rapid code iteration or define and use cells explicitly for publishing. See Chapter 10, "Publishing M-Files" for complete details.

Rapid Code Iteration Overview

When working with an M-file, you often experiment with your code—modifying it, testing it, and updating it—until you have an M-file that does what you want. For example:

- Suppose you have code that plots data. You might break the code into two cells: the code in the first cell creates the basic results, while the code in the second cell labels the plot. The two cells 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. This scenario is presented in "Example of Evaluating Cells" on page 8-211.
- Suppose you have an two images that you want to add together, and then display the results. As part of this algorithm, you want to adjust the brightness of the second image before adding it to the first image. You can read in that images, tweak the second image's brightness, read it in again, adjust its brightness, and so on using the cell mode toolbar buttons until you see the brightness you want. There is no need to save the M-file between adjustments. If you have an active Internet connection, you can watch the Rapid Code Iteration Using Cells video demo that illustrates this example and more, including an overview of the major rapid code iteration features.

Use the MATLAB Editor cell features with M-file scripts to facilitate this process. You also can use cell features with function M-files, but there are some restrictions—see "Using Cells in Function M-Files" on page 8-210.

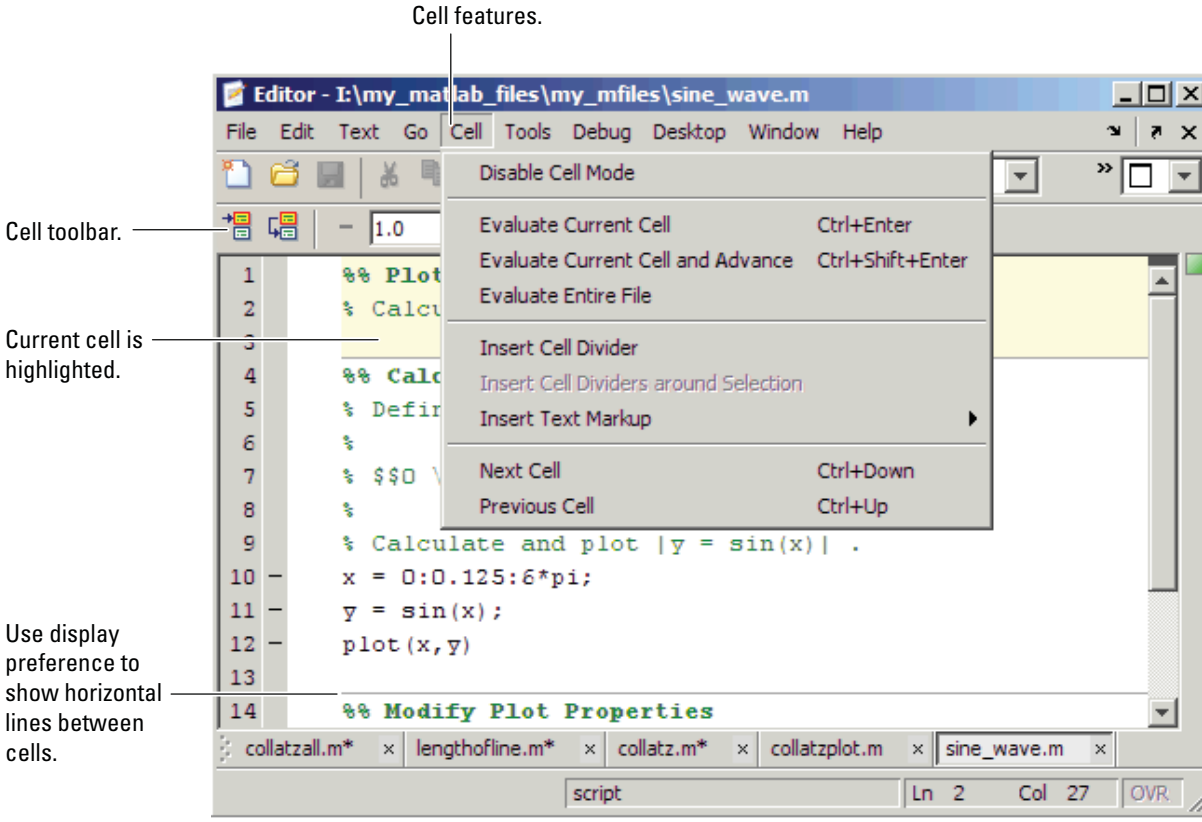
Note Cell mode is supported for use with M-files only. It is not intended for use with plain text files. When used with plain text files, results are unpredictable.

This is the overall process of using cells for rapid code iteration:

1. In the MATLAB Editor, select **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 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 8-188.

3 After 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 "Evaluating M-File Cells" on page 8-208.



Defining Cells

You define cell boundaries explicitly by inserting a line that begins with a cell break (also referred to as a cell divider), which is two percent sign characters (%%). White space can precede these two characters, and text can follow them, as long as there is white space between the %% characters and the text. For details, see "Defining Cell Boundaries Explicitly" on page 8-189.

MATLAB defines implicit cell boundaries in a code block only when you specify one or more explicit cell breaks within that code block. MATLAB defines implicit cell breaks as follows:

- MATLAB defines implicit cell breaks at the top and bottom of the file, to create an implicit cell that contains the entire file. However, the Editor does not highlight the resulting cell, which encloses the entire file, unless you add one or more explicit cell breaks to the file.
- If you define an explicit cell break in a function, MATLAB defines implicit cell breaks at the function declaration and at the function `end` statement.

  The resulting cells are nested within the full file cell. Note that if you do not end the function with an explicit `end` statement, MATLAB behaves as though the end of the function occurs immediately before the start of the next function.
- If you define an explicit cell break within a language construct (such as an if statement, a while statement, and so on), MATLAB defines implicit cell breaks at the lines containing the start and end of the language construct.

  The resulting cells are nested within the full file cell, and the function in which the code block occurs, if any.

If an implicit cell break and an explicit cell break occur on the same line, they collapse into one explicit cell break. For more information on nested cells, see "Understanding Nested Cells" on page 8-197.

This section includes the following topics:

Defining Cell Boundaries Explicitly

To define cell boundaries explicitly, you must insert cell breaks. Follow these steps:

1. Ensure that cell mode is enabled. (See "Rapid Code Iteration Overview" on page 8-186.)

2. Optionally, to help you distinguish cells from each other, do one or both of the following:

   - Include a faint gray horizontal line (rule) above each cell to help you distinguish the cells from each other.

     Select **File > Preferences > Editor/Debugger > Display**, and then in **Cell display options**, select **Show lines between cells**.

     The horizontal lines do not appear in the M-file when you print it.

   - Set a color to indicate the current cell.

     Select **File > Preferences > Editor/Debugger > Display**. Then, in **Cell display options**, select **Highlight cells**, and then select the color that you want. By default, MATLAB highlights the current cell in yellow.

     The current cell is the cell where you have placed the cursor. Like the lines between cells, highlighting helps you distinguish the cells from each other.

3. Do one of the following to insert the cell breaks:

   - Position the cursor just before the line at which you want to start the cell and select **Cell > Insert Cell Break** .
   - Click the Insert cell break button .
   - Enter two percent signs (`%%`) at the start of the line where you want to begin the new cell.

- Select the lines of code you want in a cell, and then select **Cell > Insert Cell Breaks Around Selection**

Note Program control statements, such as `if ... end`, must be contained within a single cell. You cannot insert a cell break between the `if` and the `end` statements.

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 cell boundaries, and moving lines of code.

Creating Titles for Cells

The Editor 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*. Including text in cell titles is optional, however, it improves the readability of the file and is used for cell publishing features.

To create a cell title, after the `%%` characters that specify a cell break, type a space, followed by a description of the cell.

Highlighting Cells

When the cursor is positioned in any line within a cell, the Editor highlights the entire cell that contains that line with a yellow background, by default. This identifies it as the *current cell*. For example, it is used when you select the **Evaluate Current Cell** option on the **Cell** menu.

Turning Off Cell Highlighting.

1. Select **File > Preferences > Editor/Debugger > Display**.
2. Under**Cell display options**, deselect **Highlight cells**.

Setting a Color for Cell Highlighting.

1. Select **File > Preferences > Editor/Debugger > Display**.
2. Under **Cell display options**, select **Highlight cells**.

3 Click the down arrow next to the color block beside **Highlight cells**, and then select the color with which you want to highlight cells.

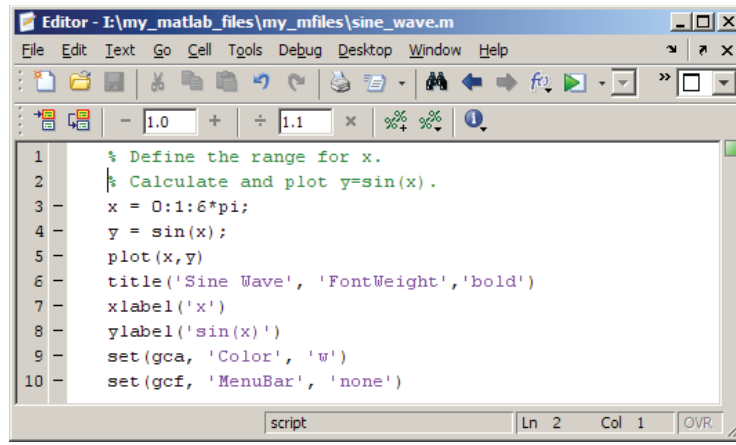
Example of Defining Cells

This example defines two cells for a simple M-file called `sine_wave.m`, shown in the following code and figure.

The steps that follow insert a cell breaks into the code to create two cells. The code in the first cell creates the basic results, while the second labels the plot. The two cells 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 Break**.

   The Editor inserts `%%` as the first line and moves the rest of the file down one line. All lines appear highlighted in yellow, indicating that the entire file is a single cell, assuming you have that display preference for cells selected.

3. After the `%%`, type a space, and then enter a cell title.

   ```
   %% Calculate and Plot Sine Wave
   ```

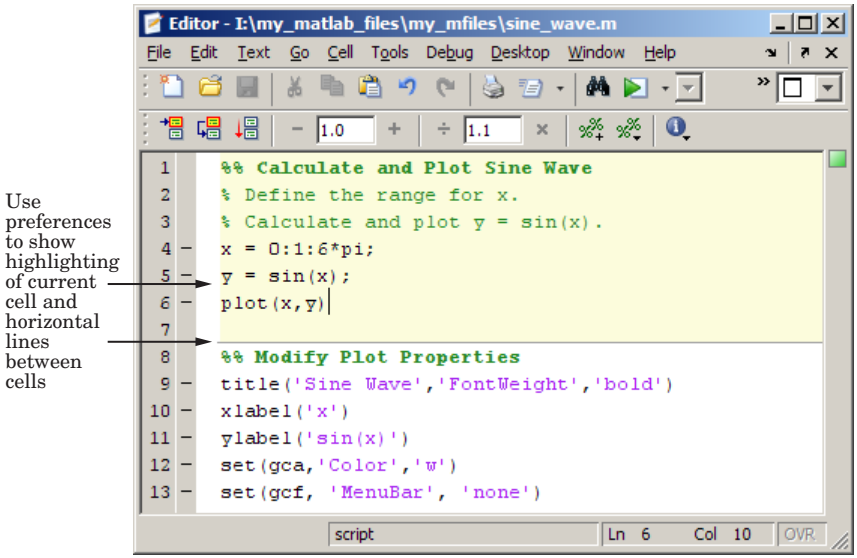
4. Position the cursor at the start of line 7, `title...`, and then select **Cell > Insert Cell Break**.

   The Editor inserts a line containing only `%%` at line 7 and moves the remaining lines down one line. A horizontal line that helps you distinguish the two cells appears above the `%%` line, assuming you have that display preference for cells selected. Lines 7 through 12 appear highlighted in yellow, indicating they comprise the current cell.

5. On line 7, type a space after the `%%`, and then enter a cell title for the new cell.

   ```
   %% Modify Plot Properties
   ```

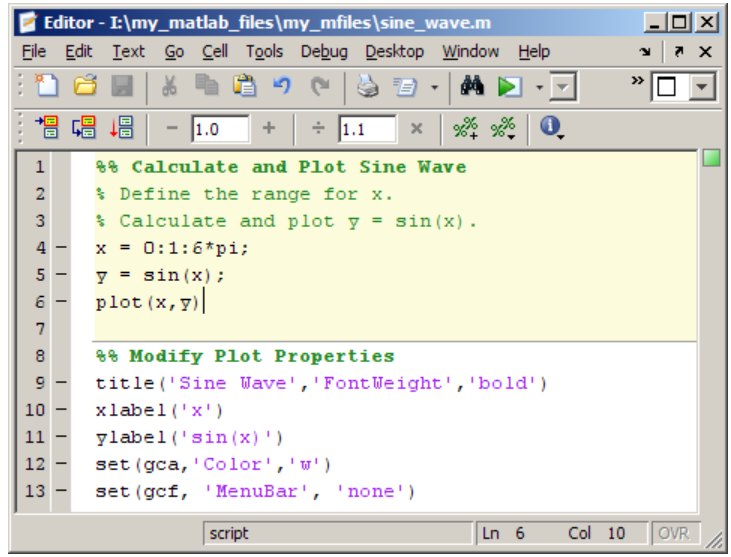
   Save the file. The file appears as shown in this figure.



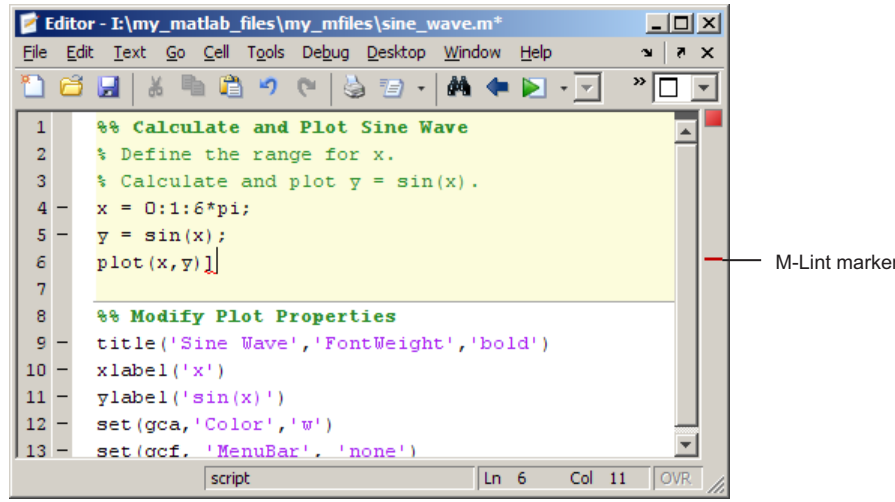
Fixing Cell Highlighting Problems

If you introduce an error into a file, such as a syntax error, cell highlighting and dividers may not appear as you expect. Although dividers and highlighting for existing cells remain in place, cells you insert after you have introduced the syntax error do not appear highlighted. In addition, if you close and reopen the file, then all cell dividers are gone and none of the cells appears highlighted.

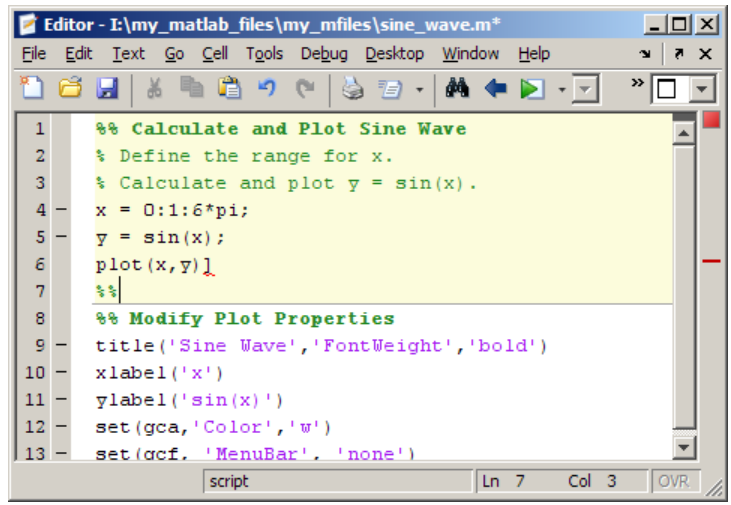
For example, suppose your code currently appears as specified in the “Example of Defining Cells” on page 8-191, and as shown here.



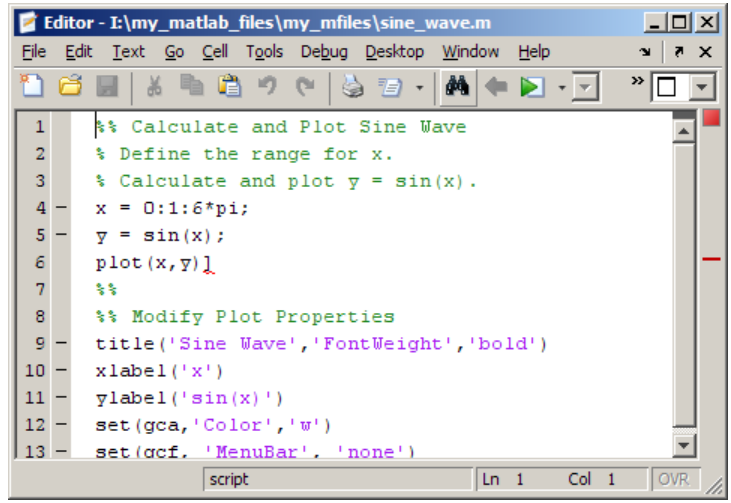
If you accidentally insert a syntax error (a closing bracket at the end of line 6), the error is evident by the M-Lint marker. The cell highlighting remains as-is.



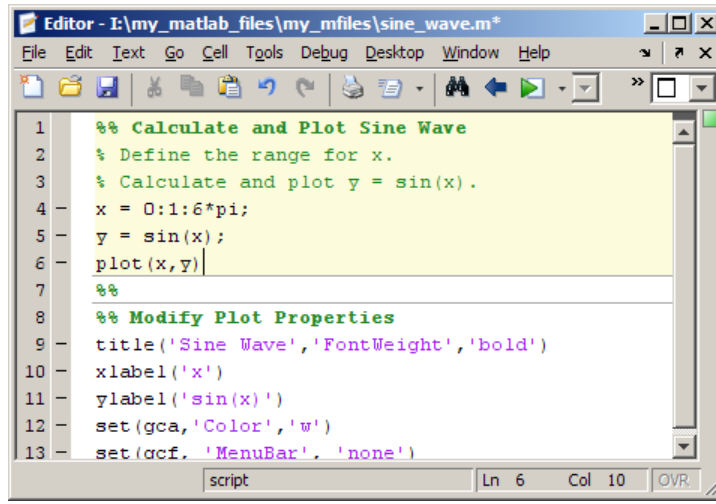
However, if you attempt to introduce a new cell at line 7, a cell divider does not appear above line 7 and the highlighting does not indicate a new cell, as you might expect.



In addition, if you save, close, and then reopen the file, all cell dividers and highlighting are gone.



To fix the problem, correct the syntax error. The cell dividers and highlighting reappear.



Removing Cells

To remove a cell, do one of the following:

- Delete one of the percent signs (%) from the line that starts the cell.

  This changes the line from a cell break to a standard comment.

- Delete the entire line that contains the %% characters.

In both cases, because you remove the cell break, MATLAB merges the two cells that were previously separated by the cell break.

Summary of Cell Mode and Cell Requirements

The following list summarizes facts to keep in mind when using cell mode and defining cells:

- Cell mode is supported for use with M-files only. It is not intended for use with plain text files.

- MATLAB does not execute the code in lines beginning with cell break characters, `%%`.
- MATLAB considers the entire file to be a single cell; therefore, the first line in a file does not have to begin with `%%`.
- 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.
- You can set preferences for cell display options by selecting **File > Preferences > Editor/Debugger > Display**, and then making choices for **Cell Display options**.
- If M-Lint finds errors in a file, cell dividers and highlighting may not appear as you expect. For details, see "Fixing Cell Highlighting Problems" on page 8-193.

For more information on M-Lint, see "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 8-119.

Understanding Nested Cells

You can insert cells within nested code, which results in nested cells. The following sections illustrate how inserting explicit cell breaks interacts with the implicit cell breaks that MATLAB inserts within an M-file.

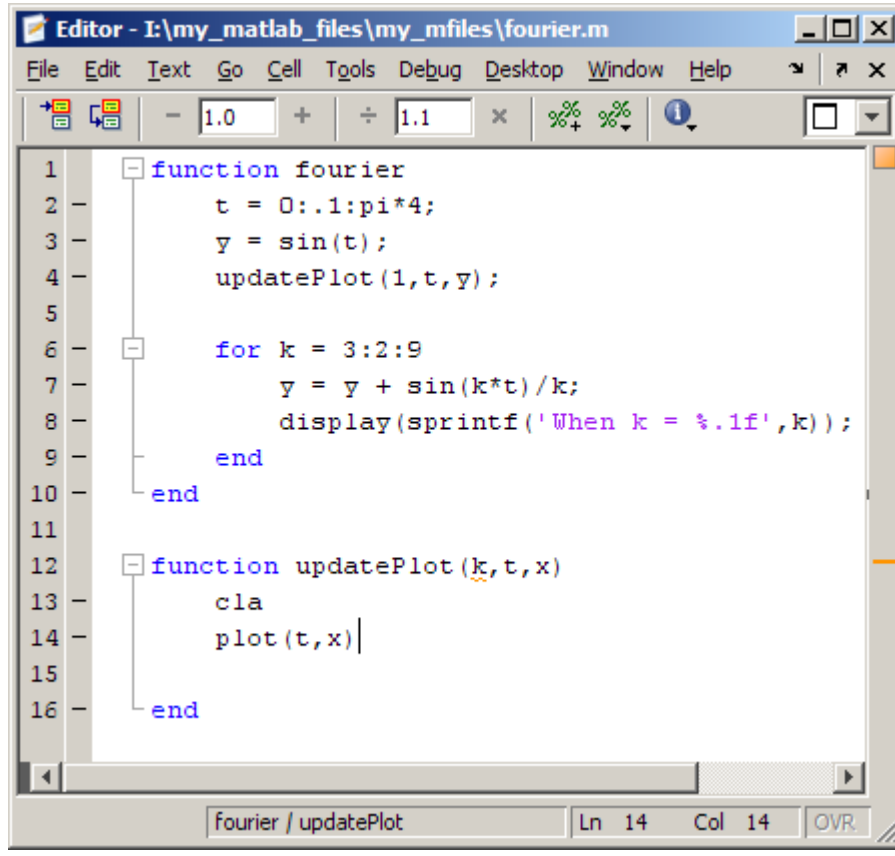
M-File Without Explicit Cell Breaks

The following code when viewed in an M-file displays no cells or highlighting. It is a single, implicit cell, defined by MATLAB.

```
function fourier
    t = 0:.1:pi*4;
    y = sin(t);
    updatePlot(1,t,y);
```

```
    for k = 3:2:9
        y = y + sin(k*t)/k;
        display(sprintf('When k = %.1f',k));
    end
end

function updatePlot(k,t,x)
    cla
    plot(t,x)

end
```

This code appears as shown in the following image when you view it in the Editor.



How Nesting Cell Breaks Result in Cells

Suppose you insert two cell breaks into `fourier.m` as follows:

1. One within the `fourier` function, at line 5.
2. One within the `for` loop, at line 7.

This results in the following cells, which are illustrated in “Example M-File with Nested Cell Breaks” on page 8-200:

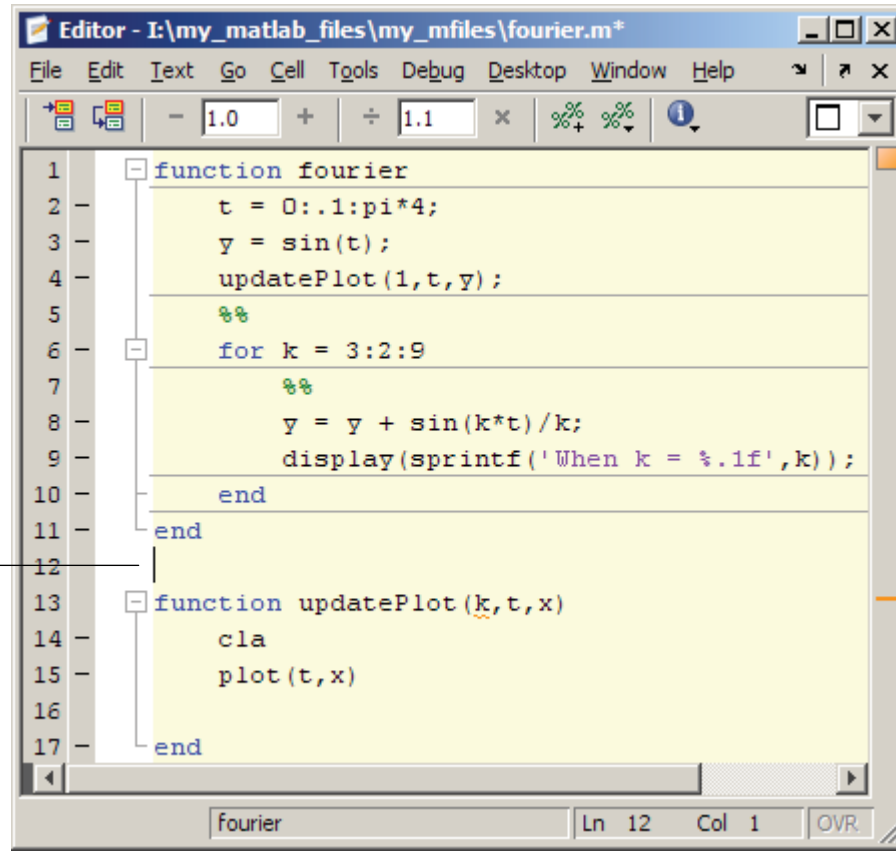
- One cell at the outermost level, from the top to the bottom of the file.

- Two cells at the second level, within the `fourier` function:
  - One from the implicit break at line 2 to the explicit break at line 5.
  - One from the explicit break at line 5 to the implicit break before line 11 (end of the function).
- One cell at the third level, within the `for` loop from the explicit line break at line 7 to the implicit line break before line 10.

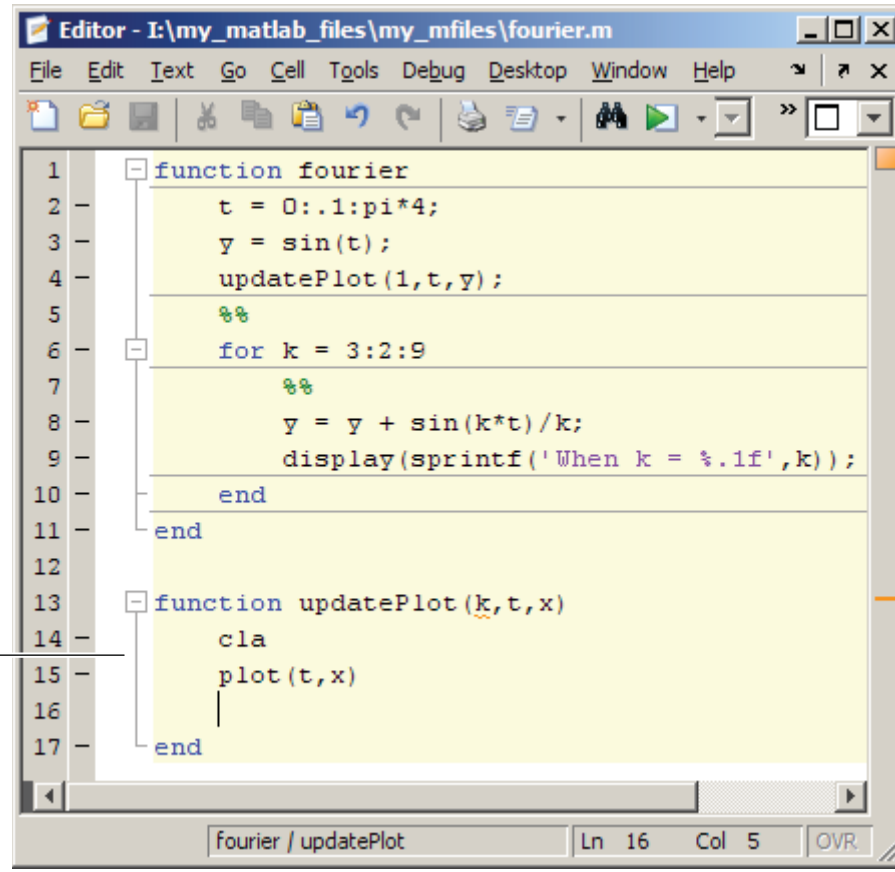
Example M-File with Nested Cell Breaks

The following images illustrate how inserting explicit cell breaks, as described in"How Nesting Cell Breaks Result in Cells" on page 8-199, affect the appearance of the M-file:

- **First level of nesting** — When you place the cursor outside a function, at the outermost level, the entire file appears highlighted, showing that it comprises a cell at this level of nesting.

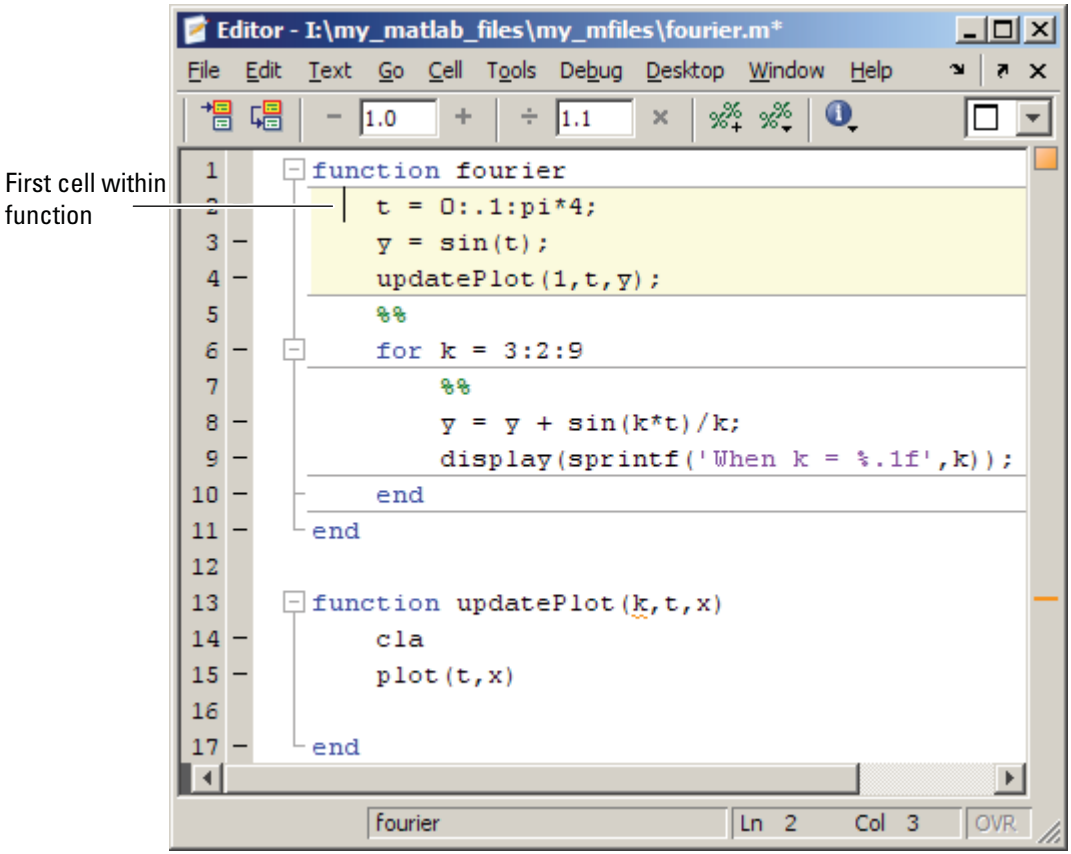


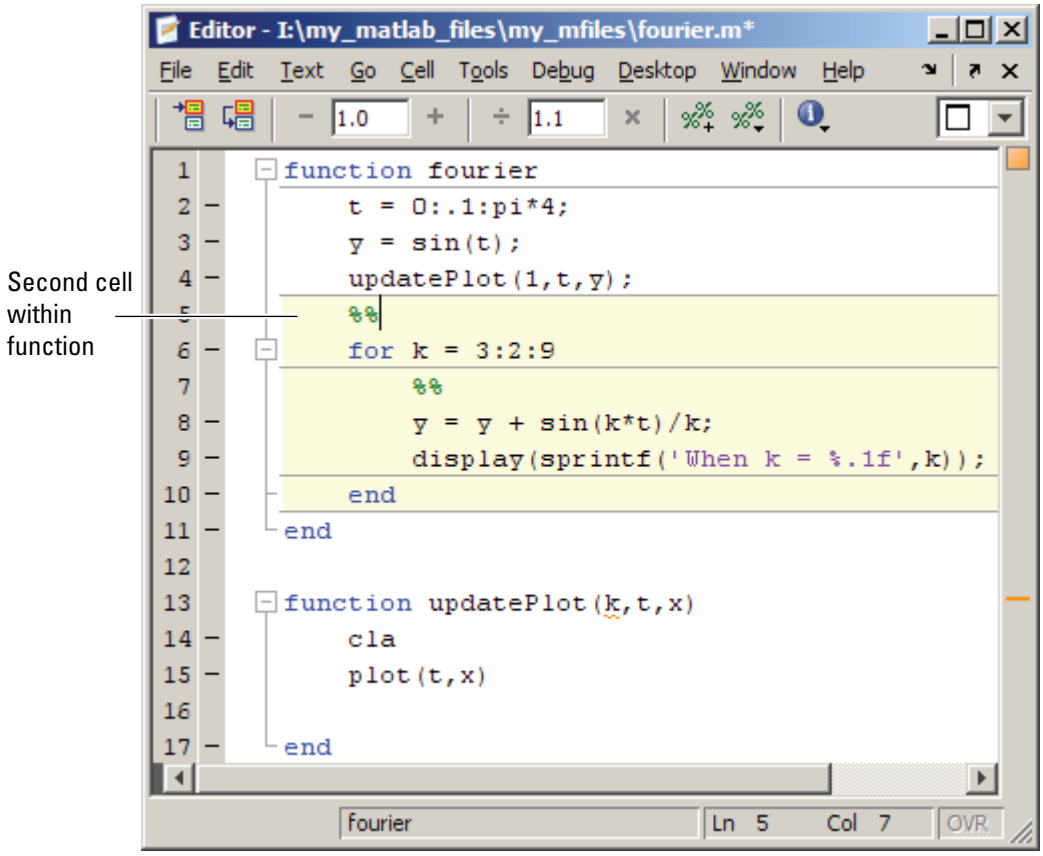
MATLAB only defines implicit cell breaks in a code block if you specify an explicit cell break within that code block. Therefore, because function `updatePlot` in this example has no explicit (and therefore, no implicit) cell breaks defined for it, when you place the cursor within that function, MATLAB considers the cursor to be within the cell that encloses the whole file.



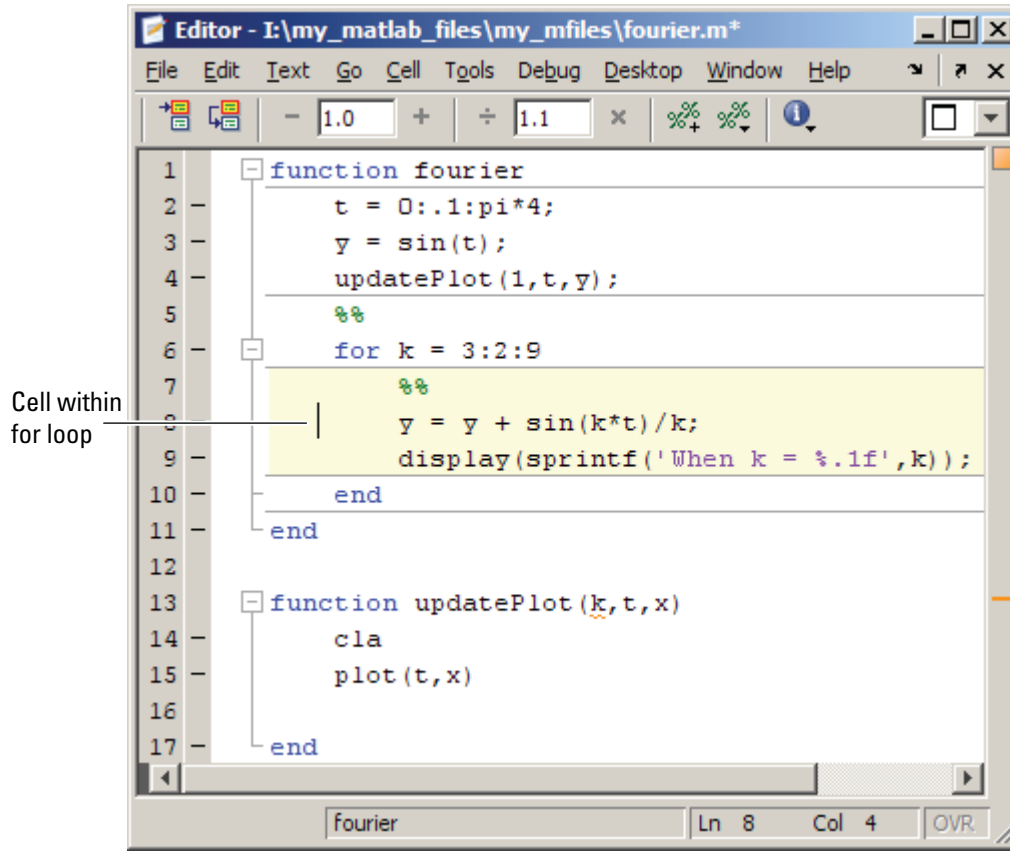
Cursor in function with no explict, and therefore no implicit, cells defined.

- **Second level of nesting** — When you place the cursor within the function (but outside the for loop), either the first or second cell at this level of nesting appears highlighted, depending on where the cursor is located.


First cell within function
Editor - I:\my_matlab_files\my_mfiles\fourier.m*
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
function fourier
t = 0:.1:pi*4;
y = sin(t);
updatePlot(1,t,y);
%%
for k = 3:2:9
%%
y = y + sin(k*t)/k;
display(sprintf('When k = %.1f',k));
end
end
function updatePlot(k,t,x)
cla
plot(t,x)
end
fourier
Ln 2 Col 3
OVR

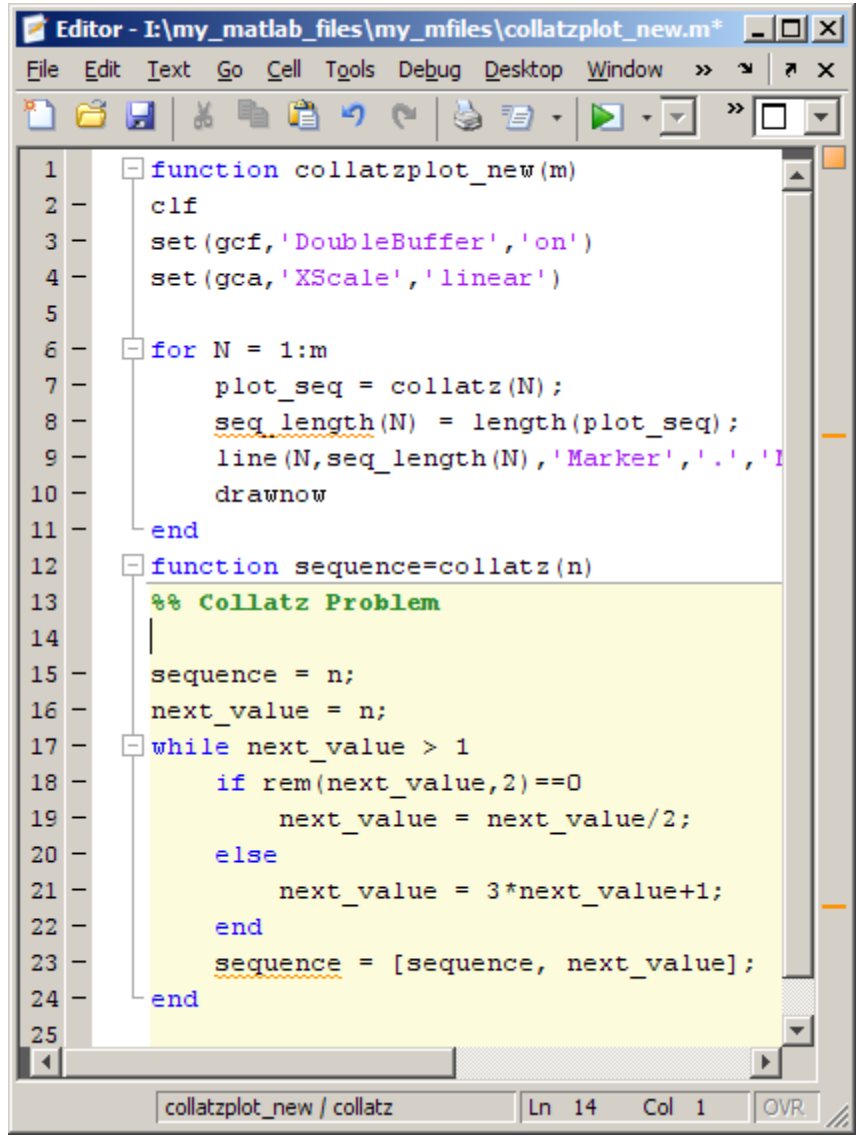


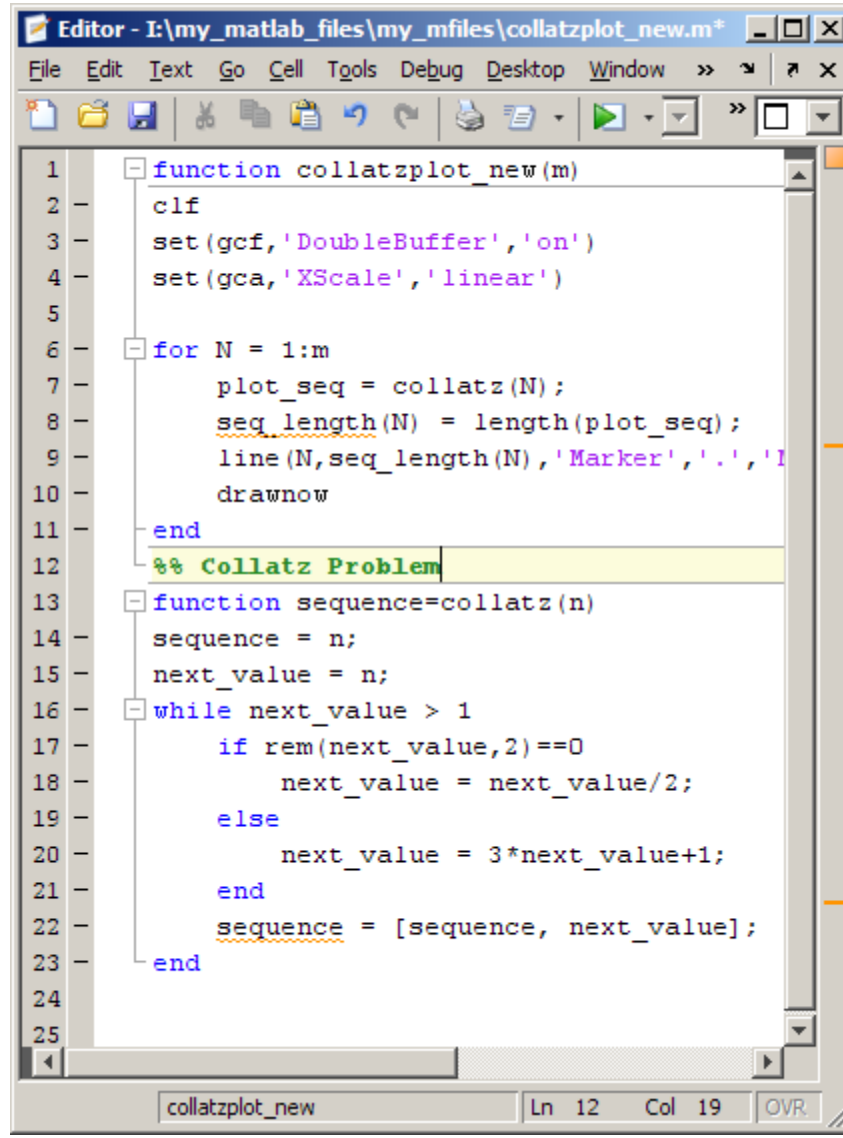
- **Third level of nesting** — When you place the cursor within the `for` loop, the cell within this loop is highlighted.



Associating Cell Breaks with Subfunctions

Be aware that if you want a cell break to be associated with a subfunction, you should place it within the subfunction, rather than above the subfunction declaration. Otherwise, it creates a single cell within the code block that precedes the subfunction. The following two images demonstrate this using `collatzplot_new.m`.


Editor - I:\my_matlab_files\my_mfiles\collatzplot_new.m*
File Edit Text Go Cell Tools Debug Desktop Window
1 function collatzplot_new(m)
2 clf
3 set(gcf,'DoubleBuffer','on')
4 set(gca,'XScale','linear')
5
6 for N = 1:m
7 plot_seq = collatz(N);
8 seq_length(N) = length(plot_seq);
9 line(N,seq_length(N),'Marker','.','I
10 drawnow
11 end
12 function sequence=collatz(n)
13 %% Collatz Problem
14
15 sequence = n;
16 next_value = n;
17 while next_value > 1
18 if rem(next_value,2)==0
19 next_value = next_value/2;
20 else
21 next_value = 3*next_value+1;
22 end
23 sequence = [sequence, next_value];
24 end
25
collatzplot_new / collatz
Ln 14 Col 1
OVR


Editor - I:\my_matlab_files\my_mfiles\collatzplot_new.m*
File Edit Text Go Cell Tools Debug Desktop Window
function collatzplot_new(m)
clf
set(gcf,'DoubleBuffer','on')
set(gca,'XScale','linear')
for N = 1:m
plot_seq = collatz(N);
seq_length(N) = length(plot_seq);
line(N,seq_length(N),'Marker','.','I
drawnow
end
%% Collatz Problem
function sequence=collatz(n)
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
collatzplot_new
Ln 12 Col 19
OVR

Evaluating M-File Cells

As you develop an M-file, you can use the Editor cell features to evaluate the M-File cell-by-cell. This method helps you to experiment with, debug, and fine-tune your code. You can navigate from cell to cell, and evaluate each cell individually. See the following topics for details:

Navigating Among Cells in an M-File

Operation	Instructions
Move to the next cell.	Select **Cell > Next Cell**.
Move to previous cell.	Select **Cell > Previous Cell**.
Move to a specific cell.	Do either of the following: • Use the Editor **Cell Mode** toolbar, as follows: **1** Click the Show cell titles button . **2** Select the cell title to which you want to move. • Use the **Go** menu, as follows: **1** Select **Go > Go To**. The Go To dialog box opens. **2** Select **Function or cell title**. **3** Select the cell title to which you want to move. **4** Click **OK**.

Evaluating Cells in an M-File

The cell evaluation features run the cell code currently shown in the Editor, 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 exist in the MATLAB workspace.

To run the code in a cell, use the **Cell** menu evaluation items or equivalent buttons in the cell mode toolbar. When you evaluate a cell, the results display in the Command Window, figure window, or elsewhere, depending on the code evaluated.

Operation	Instructions
Run the code in the current cell.	Select **Cell > Evaluate Current Cell** or click the Evaluate cell button .
Run the code in the current cell, and then move to the next cell.	Select **Cell > Evaluate Current Cell and Advance** or click the Evaluate cell and advance button .
Run all the code in the file.	Select **Cell > Evaluate Entire File** or click the Evaluate entire file button . By default, the Evaluate entire file button is not on the Editor toolbar. See "Setting Toolbars Preferences for Desktop Tools" on page 2-156 for information on how to add it. **Note** A beep indicates there is an error. See the Command Window for the error message.

Processing Considerations When Evaluating Cells

This section describes processing considerations that you should take into account when you evaluate cells in M-files.

Setting Breakpoints. 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.

Using Cells in Function M-Files. You can define and evaluate cells in function M-files as long as the variables referenced in the cell are in your workspace. 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.

Using Function Names as Variable Names in Cells. If you use a MATLAB function name as a variable name within a cell, you might receive an unexpected error when you evaluate the cell. The precedence rules that MATLAB typically follows do not apply when it evaluates a cell. Typically, MATLAB evaluates variables before functions. However, when you evaluate cells, MATLAB parses all the cell code and loads it into memory before evaluating it. Therefore, functions might be evaluated before variables under some circumstances, as illustrated by the following example:

Suppose you create a MAT file, `mydata.mat`, using the following commands:

```
clear all
info=5;
save mydata.mat
clear all
```

When you enter the following commands in the Command Window, `b` evaluates to 5, as expected:

```
load mydata
b=info
```

However, when you evaluate the same commands in an M-File cell, `b` evaluates to the MATLAB `info` function, thus the Command Window displays the following error:

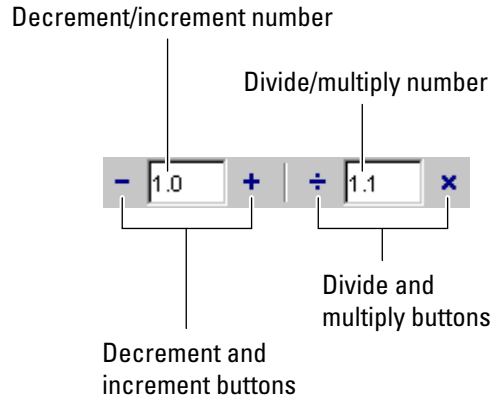
```
??? Error using ==> info
Too many output arguments.
```

For this reason, you may want to avoid using function names as variable names within M-file cells.

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.

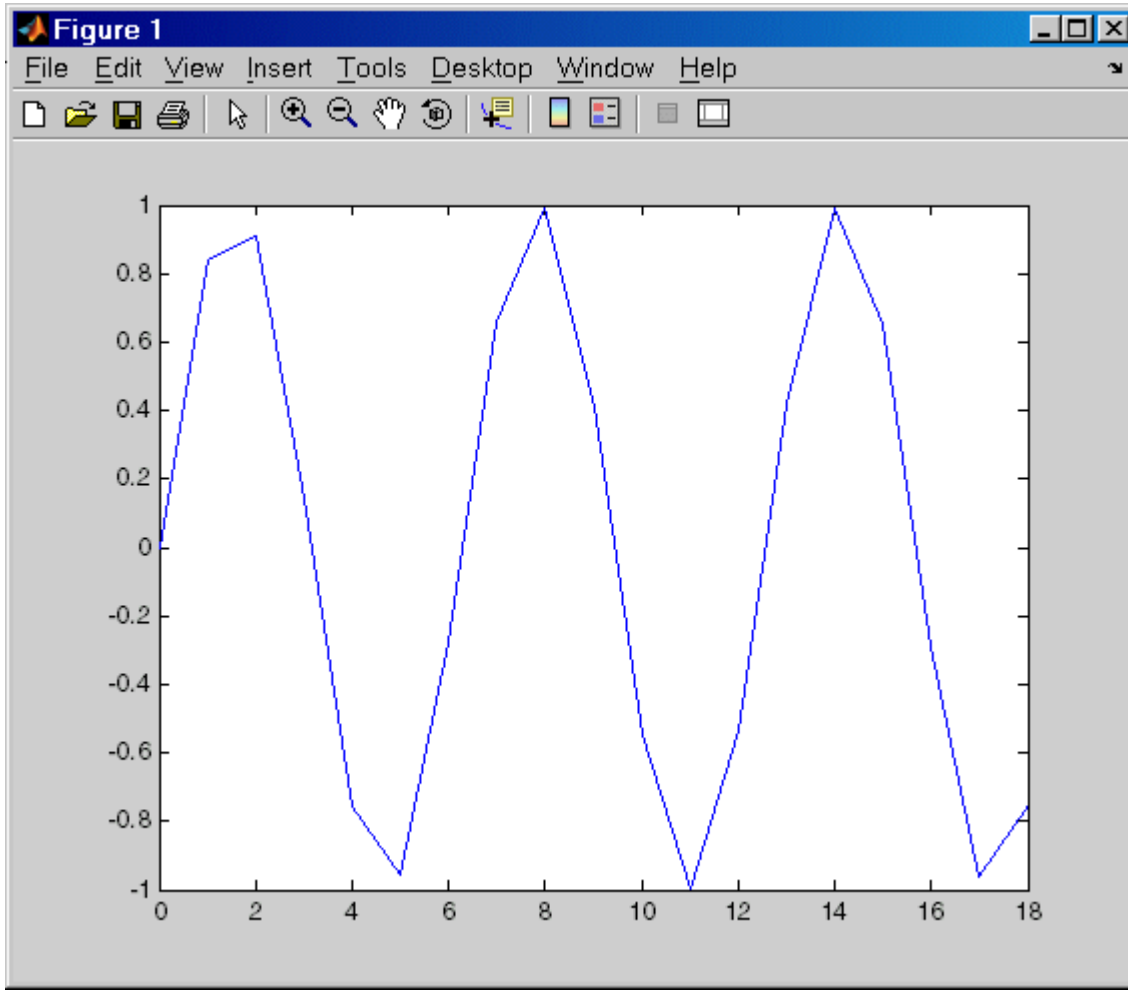
Note MATLAB software does not automatically save changes you make to values using the cell toolbar. To save changes, select **File > Save**.

Example of Evaluating 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.

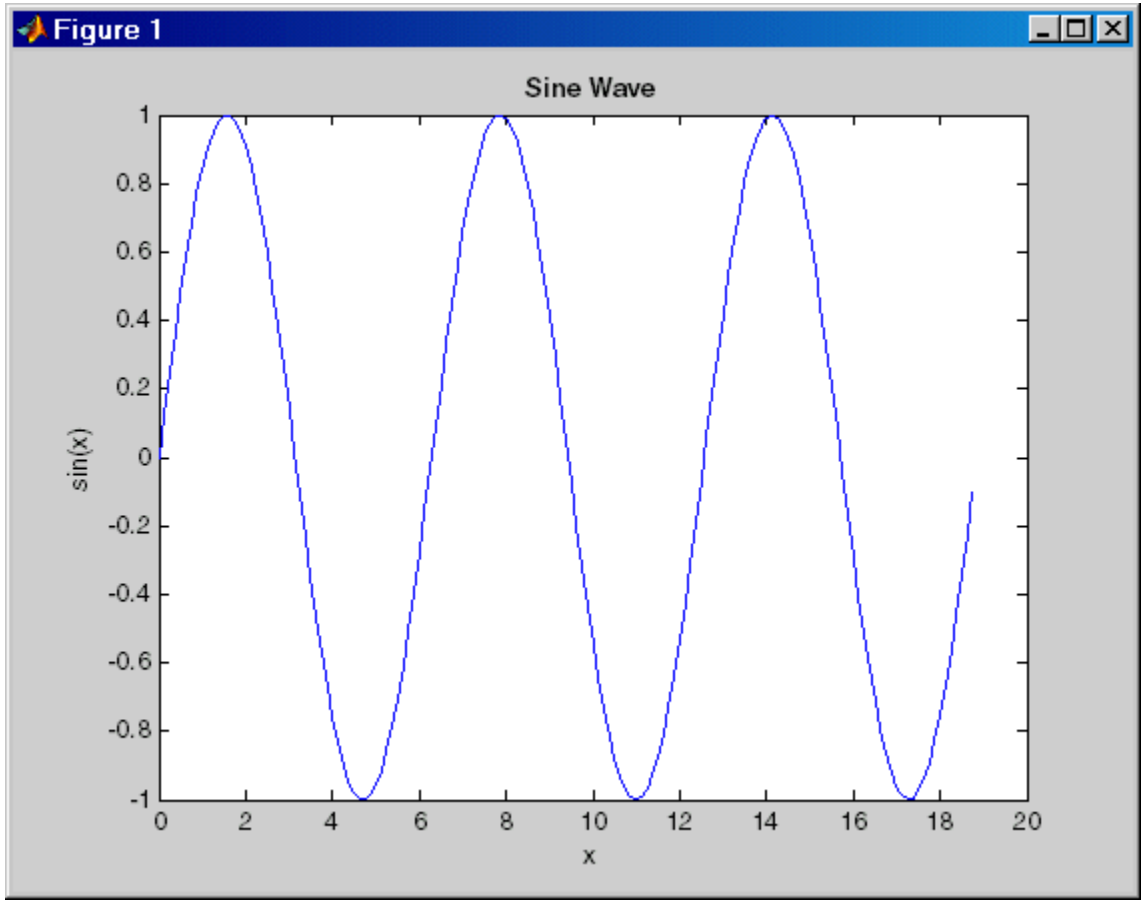
4. In the cell toolbar, click the Multiply button once. The increment for `x` as shown in line 4 changes from `0.0625` to `0.125`.

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 choose **Cell > Evaluate Current Cell** to evaluate the current cell.

MATLAB updates the figure.



Debugging Functions

Instead of, or in addition to using Editor features to debug files, you can use the following debug functions:

- `dbstop`—Sets breakpoint
- `dbclear`—Clears breakpoint
- `dbcont`—Resumes execution from breakpoint, while in debug mode
- `dbdown`—Reverses workspace shift performed by `dbup`, while in debug mode
- `dbmex`—Enables MEX-file debugging on UNIX platforms
- `dbstack`—Function calls stack, while in debug mode
- `dbstatus`—Lists breakpoints
- `dbstep`—Executes one or more lines from current breakpoint
- `dbtype`—Lists M-file with line numbers
- `dbup`—Shifts current workspace to workspace of caller, while in debug mode
- `dbquit`—Quits debug mode

9

Tuning and Managing M-Files

This set of tools provides useful information about the M-files in a folder 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.

Using M-File Reports

In this section...

See also "M-Lint Code Check Report" on page 9-22, and the File and Folder Comparisons tool.

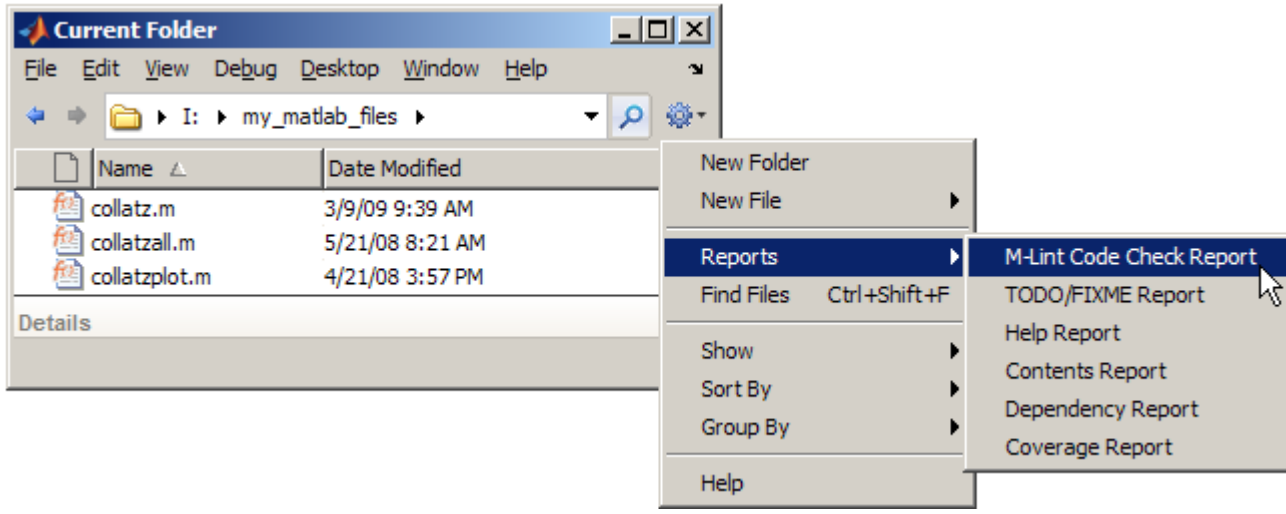
Refining and Improving M-Files Using Reports

Reports help you refine the M-files in a folder and improve their performance. They are also useful for checking the quality of files before you distribute them for use by others, such as for a finished project, to share on MATLAB Central, or for a toolbox. (A toolbox is a collection of files for use with MATLAB and related products.)

Accessing Reports

Access reports from the MATLAB Current Folder browser, as follows:

1. Select **Desktop > Current Folder**.
2. Navigate to the folder containing the M-files for which you want to produce reports.
3. On the Current Folder browser toolbar, click the **Actions** button  , and then select the type of report you want to run for all the M-files in the current folder.



The report you select appears as an HTML document in the MATLAB Web Browser.

Note You cannot run reports when the path is a UNC (Universal Naming Convention) path, that is, starts with \\. Instead, use an actual hard drive on your system, or a mapped network drive.

Using Reports

All M-File reports contain various links that enable you to access additional information, as described in the table that follows:

To	Do this
Open a file in the Editor to view it or make changes to it.	Click a file name in the report.
Open a file at the line listed in a report.	Click the line number.
Update a report after making changes to the report options.	Click **Rerun This Report.**

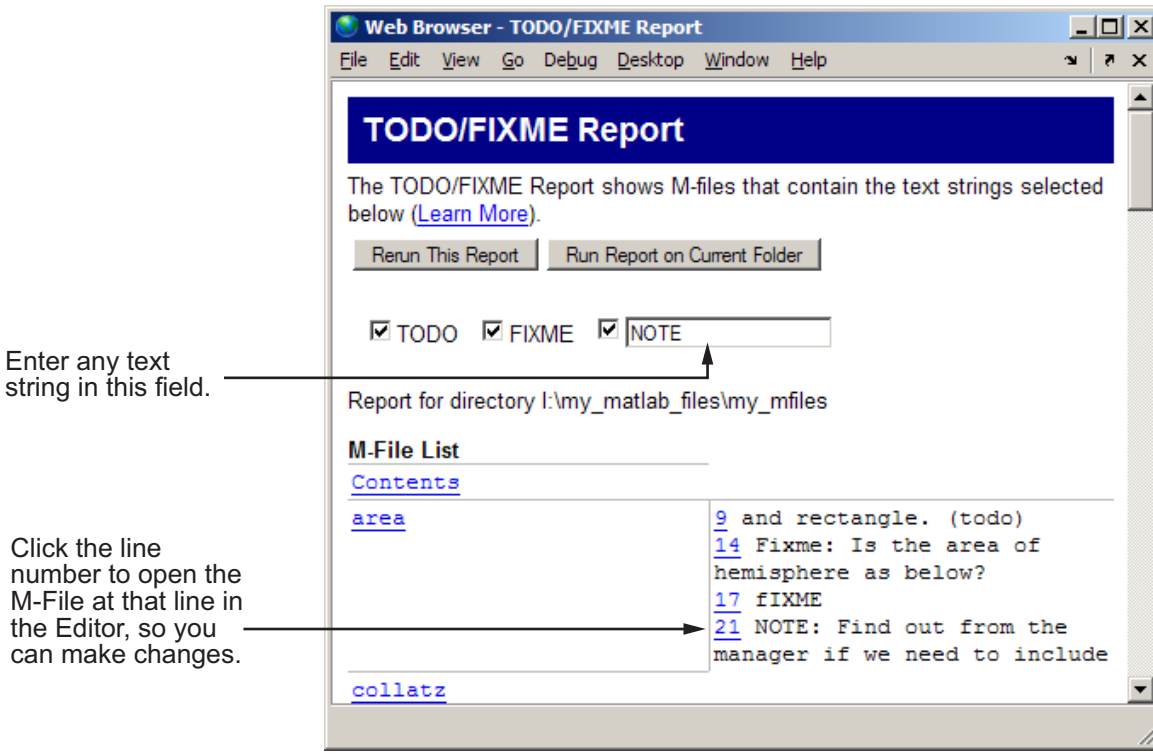
To	Do this
Update a report after changing any files in the folder.	Click **Rerun This Report.**
Generate the same type of report for a different folder.	**1** Keep the current report open. **2** Change the MATLAB current folder. **3** Click **Run Report on Current Folder**.

Note Clicking **Rerun This Report** reruns the report for the folder shown in the report, not for the MATLAB current folder.

Identifying M-Files with Reminder Annotations

The TODO/FIXME Report identifies all the M-files in a given folder that you have annotated by adding comments with the text `TODO`, `FIXME`, or a string of your choosing. This enables you mark, and then find later, areas in an M-File with annotations to indicate that you intend to improve, complete, or perform some other update in the future. The TODO/FIXME Report presents a list of files containing the annotations in a Web browser.

This sample TODO/FIXME Report shows files containing the strings TODO, FIXME, and NOTE. The search is case insensitive.



Working with TODO/FIXME Reports

1. Select **Desktop > Current Folder** and navigate to the folder containing the M-files for which you want to produce a TODO/FIXME report.

2. On the Current Folder browser toolbar, click the **Actions** button, and then select **Reports > TODO/FIXME Report**.

   The TODO/FIXME Report opens in the MATLAB Web browser.

3. In the TODO/FIXME Report window, select one or more of the following to specify the lines that you want the report to include:

   - `TODO`
   - `FIXME`

- The text field check box

  You can then, enter any text string in this field, including a regular expression. For example, you might enter `NOTE`, `tbd`, or `re.*check`.

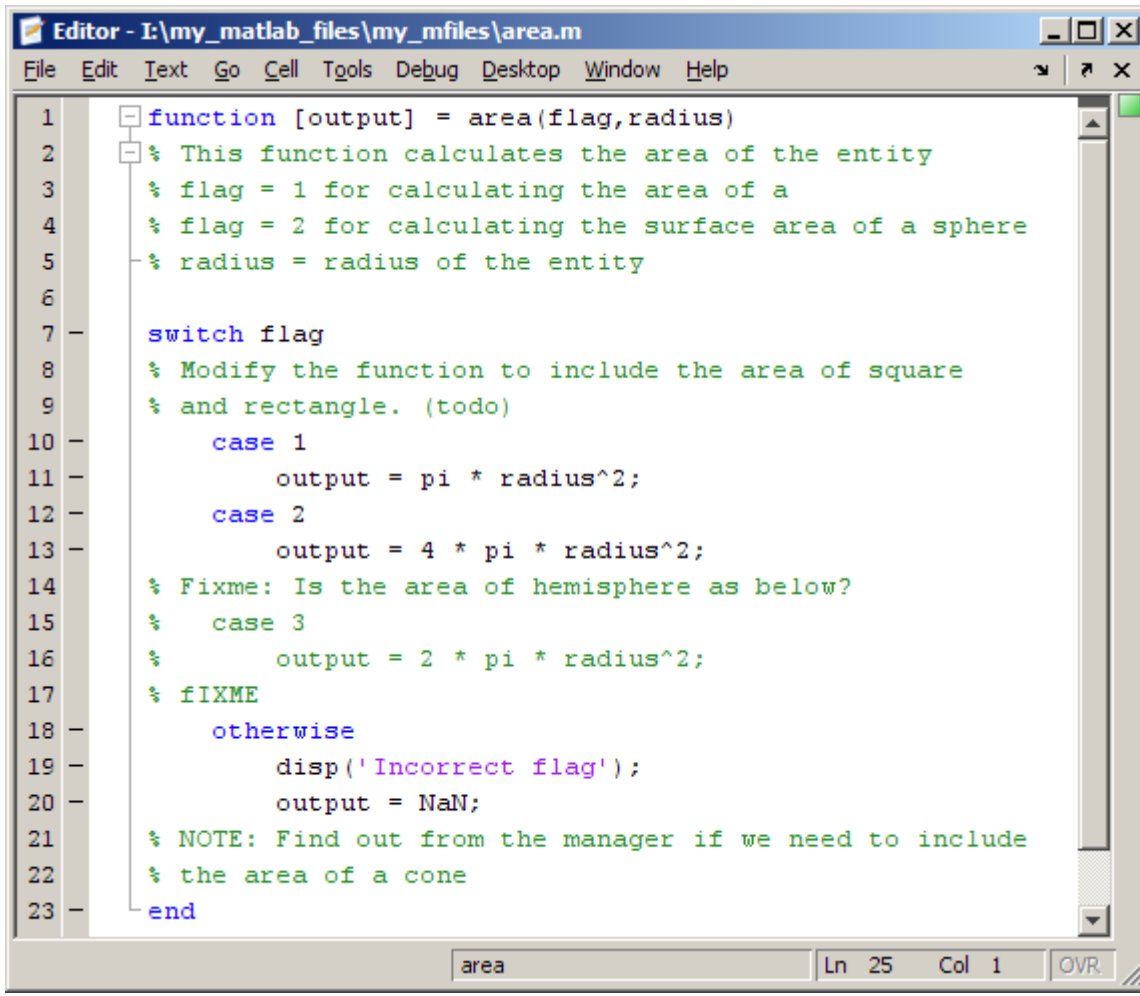
4. Run the report on the M-Files in the current folder, by clicking **Rerun This Report**

   The Window refreshes with a list of all the lines in the M-File programs within the specified folder that contain the strings you selected in step 1. Matches are not case-sensitive.

   If you want to run the report on a folder other than the one currently specified in the TODO/FIXME Report window, change the current folder to the one where you want to run the report, and then click **Run Report on Current Folder**.

To open an M-File in the Editor at a specific line, click the line number in the report. Then you can make changes, as needed.

Suppose you have an M-File, `area.m`, in the current folder. The code for `area.m` is shown in the image that follows.



When you run the TODO/FIXME report on the folder containing `area.m`, with the `TODO` and `FIXME` strings selected and the string `NOTE` specified and selected, the report lists:

```
9 and rectangle.  (todo)
14 Fixme: Is the area of hemisphere as below?
17 fIXME
21 NOTE: Find out from the manager if we need to include
```

area	9 and rectangle. (todo) 14 Fixme: Is the area of hemisphere as below? 17 fIXME 21 NOTE: Find out from the manager if we need to include

Notice the report includes the following:

- Line 9 as a match for the `TODO` string. The report includes lines that have a selected string regardless of its placement within a comment.
- Lines 14 and 17 as a match for the `FIXME` string. The report matches selected strings in the M-File regardless of their casing.
- Line 21 as a match for the `NOTE` string. The report includes lines that have a string specified in the text field, assuming you select the text field.

Generating a Summary View of the Help Components in M-Files

A Help Report presents a summary view of the help component of your M-files. Use this information to assist you in identifying files of interest or files that lack a help component. It is a good practice to provide help for your files not only to assist you in recalling their purpose, but to assist others who use the 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.

Working with Help Reports

1. Select **Desktop > Current Folder** and navigate to the folder containing the M-files for which you want to produce a Help Report.
2. On the Current Folder browser toolbar, click the **Actions** button  , and then select **Reports > Help Report**.

The Help report opens in the MATLAB Web Browser.

3 Select one or more options, described in the following list, to have the Help Report display the specified help information:

- **Show subfunctions** to have the Help Report display help information for all subfunctions called by each function. Help information for subfunctions is highlighted in gray.
- **Description** to have the Help Report display 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, then **No description line**, highlighted in pink, appears instead.
- **Examples** have the Help Report display 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** have the Help Report display 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** have the Help Report display 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 the MATLAB Command Window 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 folder on the search path. You might want to move that file to a folder 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 folder to the path or make its folder become the current folder.

- **Copyright** have the Help Report display 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).

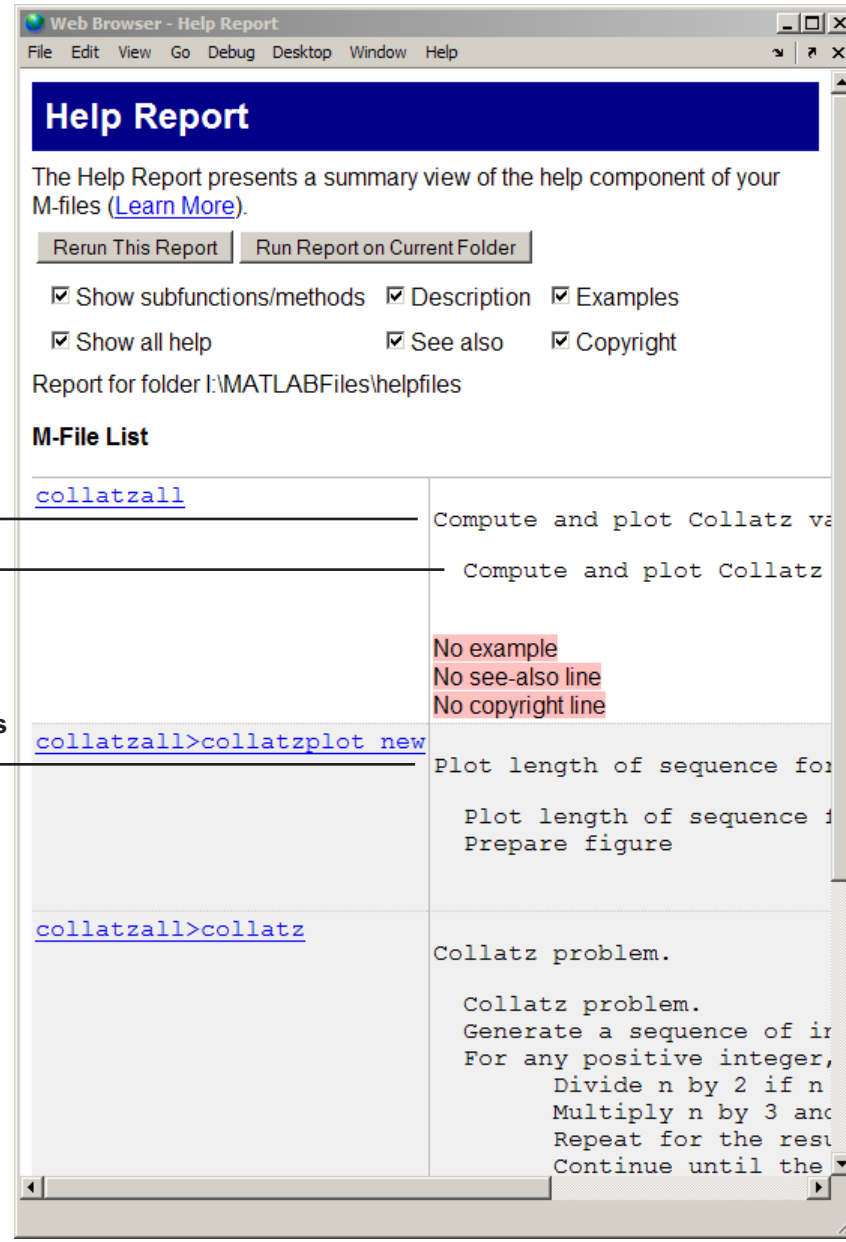
  It is a good practice to include a copyright line in the help for your M-files, that notes the year you created the file and the current year. For example, for an M-file you created in 2001, include this line

  ```
  % Copyright 2001-2008
  ```

  If the report:

  - Does not find a copyright line in the M-file help, **No copyright line**, highlighted in pink, appears.
  - Finds that the end of the date range is not the current year, **Copyright year is not current**, highlighted in pink, appears.

4 Click **Rerun This Report**. Your report resembles the following image.



Check **Description** to see the first help line.

Check **Show all help** to see the rest of the help.

Check **Show subfunctions** to see help information for subfunctions.

Displaying and Updating a Report on the Contents of a Folder

The Contents Report displays information about the integrity of the `Contents.m` file for a given folder. A `Contents.m` file includes the file name and a brief description of each M-file in the folder. 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 folder.

When you type `help` followed by the folder name, such as `help mydemos`, The MATLAB Command window displays the information contained within the `mydemos/Contents.m` file. For more information, see "Providing Help for Your Program" in the MATLAB Programming documentation.

Working with Contents Reports

1. Select **Desktop > Current Folder** and navigate to the folder containing the M-files for which you want to produce a Contents report.

2. On the Current Folder browser toolbar, click the **Actions** button, and then select **Reports > Contents Report**.

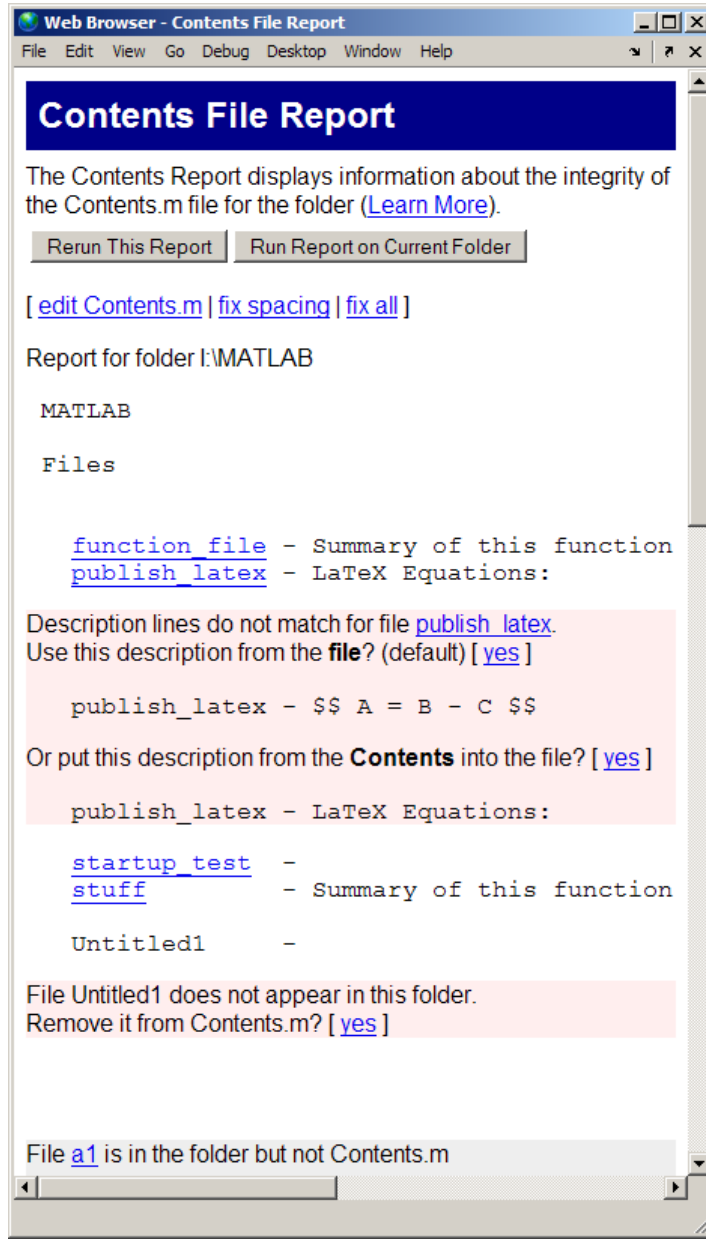
   The Contents Report opens in the MATLAB Web browser. If there is no `Contents.m` file for the folder, the report tells you the `Contents.m` file does not exist and asks if you want to create one. Click **yes** to automatically create the `Contents.m` file. You can edit the `Contents.m` file in the Editor 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 folder, such as files for internal use.

3. Update the `Contents.m` file to reflect changes you make to files in the folder. For example, when you remove a file from a folder, remove its entry from the `Contents.m` file.

   The following options are available for updating the contents:

   - **edit Contents.m**— Opens the `Contents.m` file in the Editor.
   - **fix spacing**—Automatically align the file names and descriptions in the `Contents.m` file.
   - **fix all** makes all of the suggested changes at once.

- To make changes on a case-by-case basis, read each question in the Contents Report, and then click **yes** if you want to make the suggested change.


Web Browser - Contents File Report
File Edit View Go Debug Desktop Window Help
Contents File Report
The Contents Report displays information about the integrity of the Contents.m file for the folder (Learn More).
Rerun This Report
Run Report on Current Folder
[edit Contents.m | fix spacing | fix all]
Report for folder I:\MATLAB
MATLAB
Files
function_file - Summary of this function
publish_latex - LaTeX Equations:
Description lines do not match for file publish_latex.
Use this description from the file? (default) [yes]
publish_latex - $$ A = B - C $$
Or put this description from the Contents into the file? [yes]
publish_latex - LaTeX Equations:
startup_test -
stuff - Summary of this function
Untitled1 -
File Untitled1 does not appear in this folder.
Remove it from Contents.m? [yes]
File a1 is in the folder but not Contents.m

Messages in the Contents File Report

No Contents File. This message appears if there is no `Contents.m` file in the folder. Click **yes** to automatically create a `Contents.m` file, which contains the file names and descriptions for all M-files in the folder.

```
No Contents.m file. Make one? [ yes ]
```

File Not Found. This message appears when a file included in `Contents.m` is not in the folder. These messages are highlighted in pink. For example, a message such as

```
File helloworld does not appear in this folder.
Remove it from Contents.m? [ yes ]
```

means the `Contents.m` file includes an entry for `helloworld`, but that file is not in the folder. This might be because:

- You removed the file `helloworld`.
- You manually added `helloworld` to `Contents.m` because you planned to create the file, but have not as yet.
- 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 7
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 folder is not in `Contents.m`. These messages are highlighted in gray. Click **yes** to add the file name and its description line from the M-file help to the `Contents.m` file.

```
collatzall is in the folder but not Contents.m
 collatzall - Plot length of sequence for Collatz problem
Add the line shown above? [ yes ]
```

Creating a New Contents.m File to Reflect All Files in the Current Folder

If you always want the `Contents.m` file to reflect all files in the current folder, you can automatically generate a new `Contents.m` file rather than changing the file based on the Contents Report, as follows:

1. Delete the existing `Contents.m` file.
2. Run the Contents Report.
3. Click `yes` when prompted for MATLAB to automatically create a Contents Report.

Displaying Dependencies Among M-Files

The Dependency Report shows dependencies among M-files in a folder. This is useful for determining:

- Which files in the folder are required by other files in the folder
- If any files in the current folder will fail if you delete a file
- If any called files are missing from the current folder

The report does not list M-files in the `toolbox/matlab` folder as dependencies because every MATLAB user has those files.

To provide meaningful results, the dependency report requires that the following be true:

- The search path when you run the report is the same as when you run the files in the folder. (That is, the current folder is at the top of the search path.)
- The files in the folder for which you are running the report do not change the search path or otherwise manipulate it.
- The files in the folder do not load variables, or otherwise create name clashes that result in different program elements with the same name.

Working with Dependency Reports

1. Select **Desktop > Current Folder** and navigate to the folder containing the M-files for which you want to produce a Dependency Report.
2. On the Current Folder browser toolbar, click the **Actions** button, and then select **Reports > Dependency Report**.

   The Dependency report opens in the MATLAB Web Browser.
3. If you want, select one or more options within the report, as follows:
   - To see a list of all M-files (children) called by each M-file in the folder (parent), select **Show child functions**.

     The report indicates where each child function resides, for example, in a specified toolbox. If a child function's location is listed as unknown, it might be because the child function is not on the search path or in the current folder, or the file may have been moved or deleted.
   - To list the M-files that call each M-file, select **Show parent functions**.

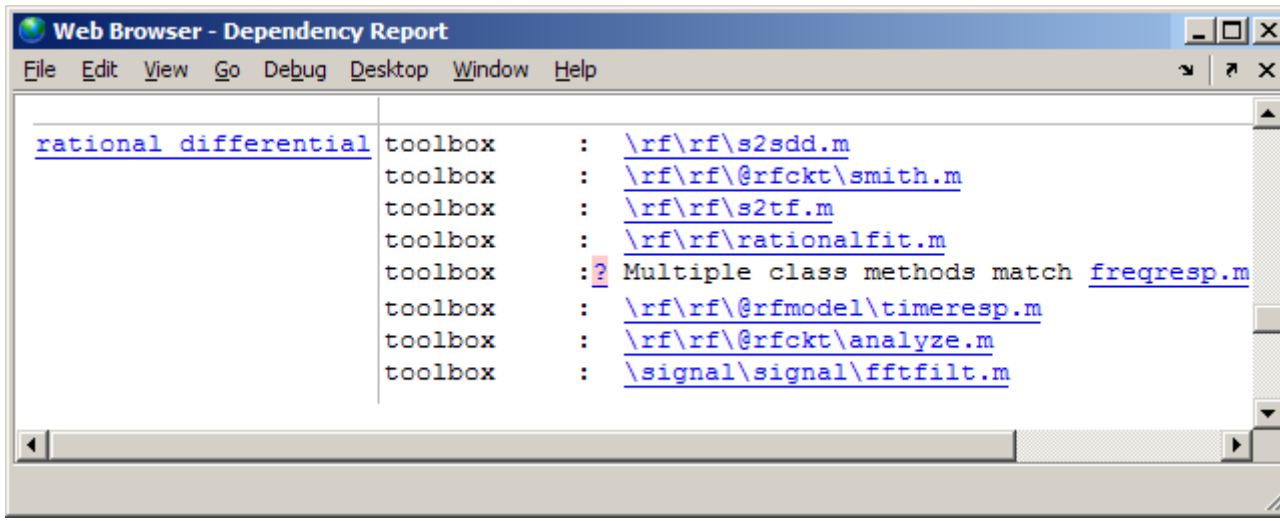
     The report limits the parent (calling) functions to those in the current folder.
   - To include subfunctions in the report, select **Show subfunctions**. Subfunctions are listed directly after the main function and are highlighted in gray.
4. Click **Run Report on Current Folder**.

chirpy.m calls two files in the Signal Processing Toolbox and one in the Image Processing Toolbox.

go.m calls mobius.m, located in the current folder.

Because the report is a static analysis, it cannot determine run-time data types and, therefore, cannot identify the particular class methods required

by an M-file. If multiple class methods match a referenced method, the Dependency Report inserts a question mark link next to the file name, as shown in the following image.



Click the question mark link to see a list of the class methods with the specified name that the MATLAB software might use. MATLAB lists *almost all* the method files on the search path that match the specified method file (in this case, `freqresp.m`). Do not be concerned if the list includes methods of classes and MATLAB built-in functions that are unfamiliar to you.

It is not necessary for you to determine which file will be used. MATLAB determines which method to use using the object that the program calls at run time. This list is provided as an indication of the possible toolboxes and method files used.

The following image shows the contents of the right side of the Web browser after you click the question mark link.

```
toolbox :   \rf\rf\s2sdd.m
toolbox :   \rf\rf\@rfckt\smith.m
toolbox :   \rf\rf\s2tf.m
toolbox :   \rf\rf\rationalfit.m
toolbox :? Multiple class methods match freqresp.m

   Unable to determine which of the following files will run: (Learn More)
   \control\ctrlobsolete\freqresp.m
   W:\bat\Akernel\perfect\matlab\toolbox\control\control\@lti\freqresp.m
   W:\bat\Akernel\perfect\matlab\toolbox\ident\ident\@idproc\freqresp.m
   W:\bat\Akernel\perfect\matlab\toolbox\ident\ident\@idpoly\freqresp.m
   W:\bat\Akernel\perfect\matlab\toolbox\ident\ident\@idmodel\freqresp.m
   W:\bat\Akernel\perfect\matlab\toolbox\ident\ident\@idfrd\freqresp.m
   W:\bat\Akernel\perfect\matlab\toolbox\rf\rf\@rfmodel\freqresp.m
```

The Dependency Report is similar to running the `depfun` function, although the two do not provide identical results. For performance purposes, the Dependency Report limits the functions considered. Therefore, do not use the Dependency Report to determine which M-files you need to provide when you tell someone to run a particular M-file; instead use the `depfun` function.

Identifying How Much of an M-File Ran When Profiled

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 view coverage details in the Profiler detail report, or by following these steps:

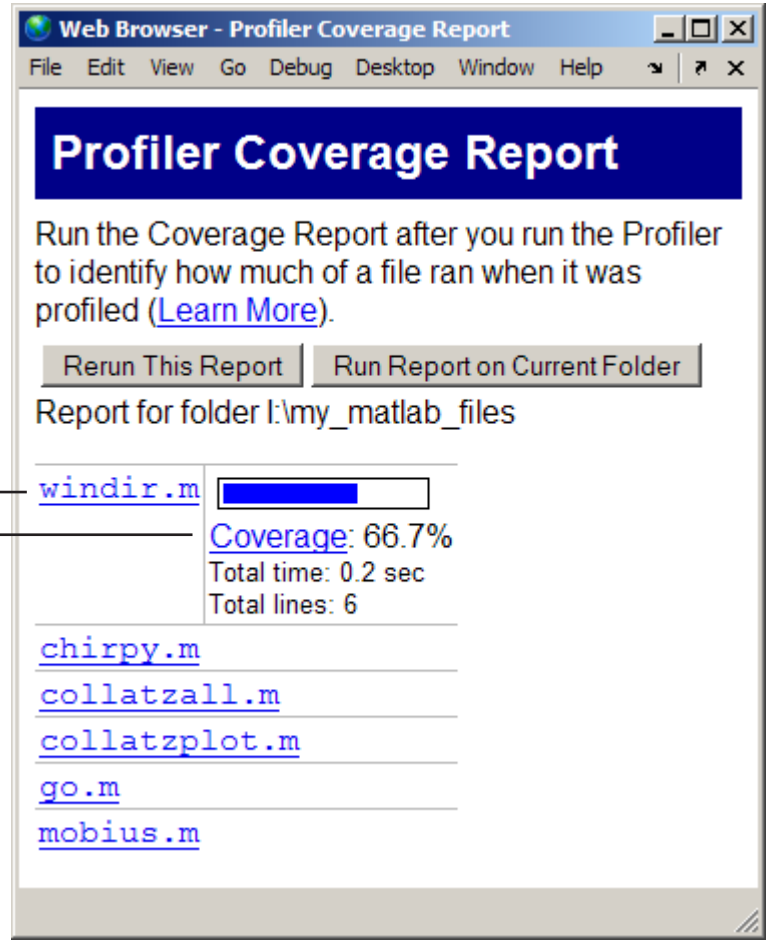
1. On the MATLAB desktop, select **Desktop > Profiler**. Profile an M-file in the Profiler. For detailed instructions, see “Profiling for Improving Performance” on page 9-27.

2. Select **Desktop > Current Folder** and navigate to the folder containing the M-file for which you ran the Profiler.

3 On the Current Folder browser toolbar, click the **Actions** button, and then select **Reports > Coverage Report**.

The **Coverage Report** appears, providing a summary of coverage for the M-file you profiled.

4 Click the **Coverage** link to see the Profile Detail Report for the file.



The Coverage Report shows the percentage of a file that ran when it was profiled.

Click the Coverage link to see the Profile Detail Report.

M-Lint Code Check Report

In this section...

Running the M-Lint Code Check Folder 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. 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 folder report, follow these steps:

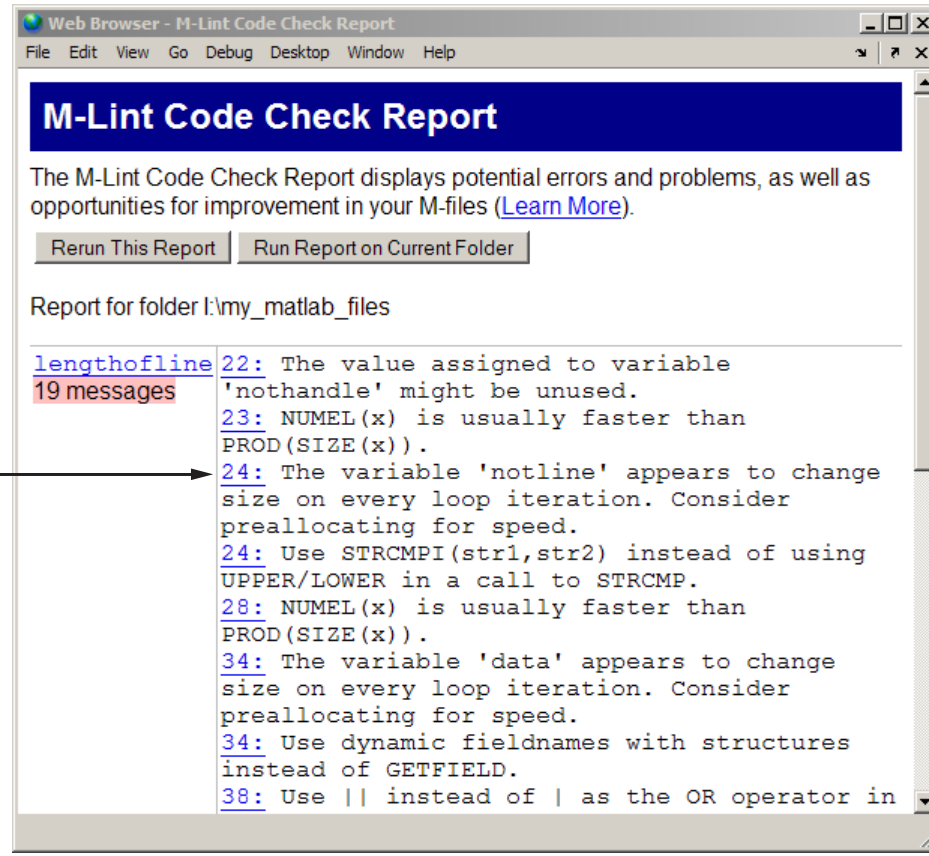
1. In the Current Folder browser, navigate to the folder that contains the M-files you want to check with M-Lint. To use the example shown in this documentation, `lengthofline.m`, you can change the current folder by running

   ```
   cd(fullfile(matlabroot,'help','techdoc','matlab_env','examples'))
   ```

2. If you plan to modify the example, save the file to a folder for which you have write access, and then make that folder the current MATLAB folder. In this example, the file is saved to `I:\my_matlab_files`.

3. In the Current Folder browser toolbar, click the **Actions** button  , and then select **Reports > M-Lint Code Check Report**.

   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 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 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, click on the link in the M-Lint message if one is provided, as described in "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 8-119.
- If the M-Lint message does not contain a link and 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 9-24.

- 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 to help you identify the changes you made to the file. For more information, see "Comparing Two Text Files" on page 8-81.

6. Run and debug the file(s) again to be sure you have not introduced any inadvertent errors.

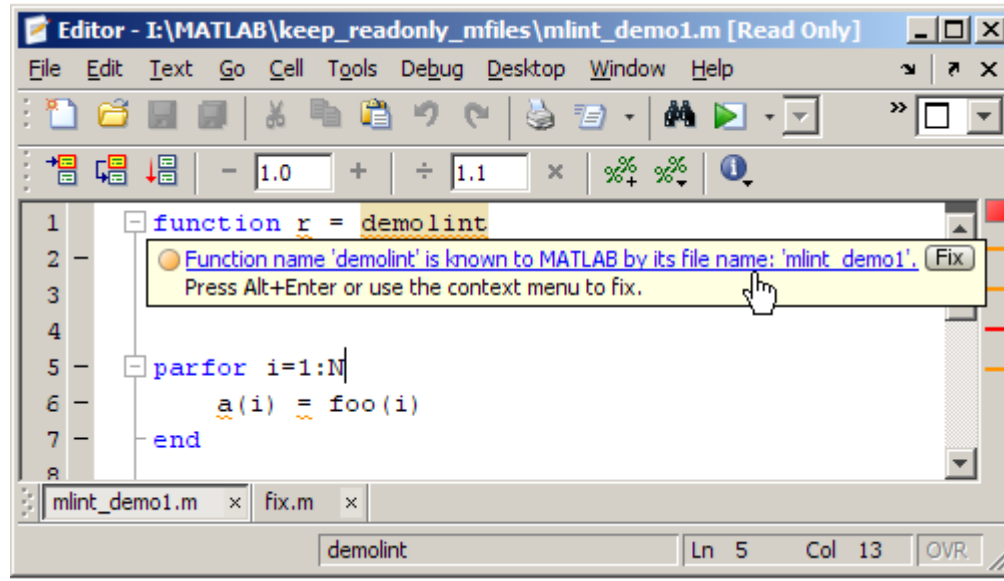
7. If the M-Lint Code Check Report is already displayed, click **Rerun This Report** to update the report based on the changes you made to the file. Ensure the M-Lint messages are gone, based on the changes you made to the M-files.

Making Changes Based on M-Lint Messages

For information on how to correct the potential problems presented by M-Lint, use the following resources:

- If available, click the link for the extended message in the M-Lint message tooltip, as shown in the image following this list. Not all M-Lint messages have extended messages.
- Look for relevant topics in the Programming Fundamentals and "Programming Tips" documentation.
- Use the Help browser **Search** and **Index** panes to find documentation about terms presented in the M-Lint messages.

The following image shows an M-Lint tooltip with a link to an extended M-Lint message. The orange *underline* indicates that an extended M-Lint message will be presented in a tooltip when you hover over the word `demolint` in the code. (The orange *highlighting* indicates that an automatic fix is available.)



Other techniques to help you identify problems in and improve your M-files are in these topics:

- "Syntax Highlighting" on page 8-52 in the Command Window and the Editor
- "Examining Errors" on page 3-7 generated when you run the M-file

- "Finding Errors, Debugging, and Correcting M-Files" on page 8-116, namely the Editor and debugging functions
- "Profiling for Improving Performance" on page 9-27 for improving performance

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 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—see "Checking M-File Code for Problems Using the M-Lint Code Analyzer" on page 8-119.

Profiling for Improving Performance

In this section...

What Is Profiling?

Profiling is a way to measure where a program spends time. To assist you in profiling, MATLAB software 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.

Note When using the Parallel Computing Toolbox™ software, you can use the parallel profiler to profile parallel jobs. See "Using the Parallel Profiler" for details.

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 9-35 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 9-37 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

For information about the reports generated by the Profiler, see "Profile Summary Report" on page 9-35 and "Profile Detail Report" on page 9-37.

Opening the Profiler

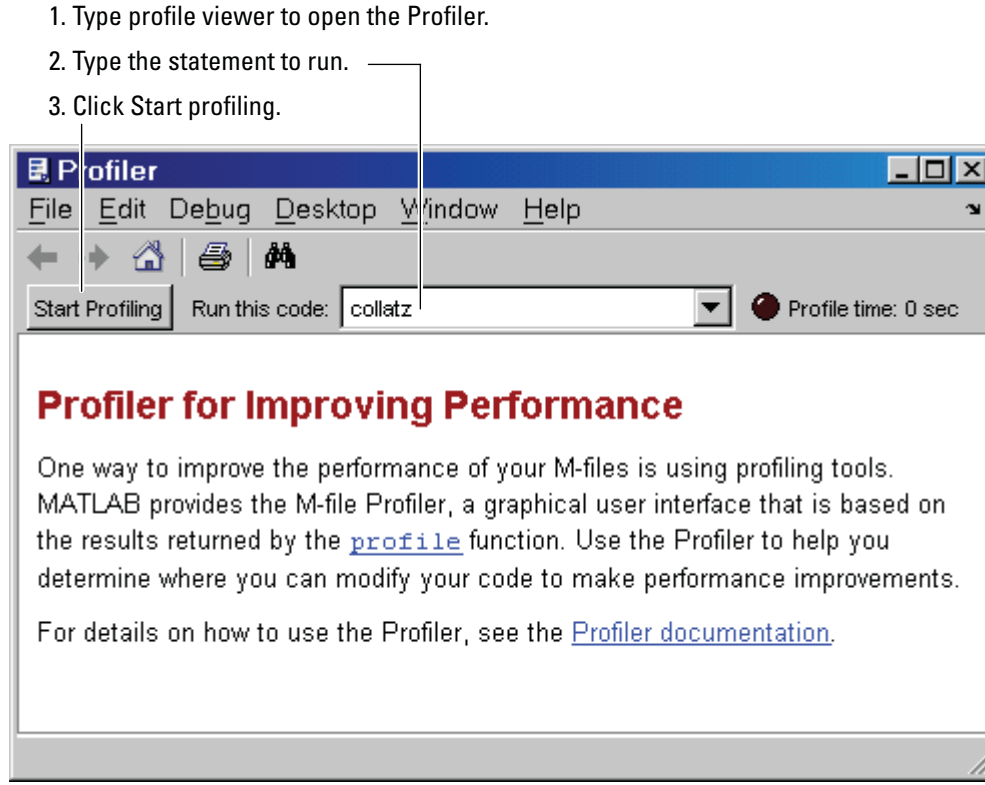
You can use any of the following methods to open the Profiler:

- Select **Desktop > Profiler** from the MATLAB desktop.
- Click the Profiler button  in the MATLAB desktop toolbar.
- With a file open in the MATLAB Editor, 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 typical steps for profiling.



To profile an M-file or a line of code, follow these steps:

1 If your system uses Intel® multi-core chips, you may want to restrict the active number of CPUs to one.

See “Intel Multi-Core Processors — Setting for Most Accurate Profiling on Windows Systems” on page 9-33 for details.

2 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.

3. 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.

   When the profiling is finished, the **Profile time** indicator becomes dark red and shows the length of time the Profiler ran. The statements you profiled are shown as having been executed in the Command Window.

   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 time also does not match the time reported in Profiler Summary report statistics, which is based on `cpu` time by default, not wall clock time. To view profile statistics based on wall clock time, use the `profile` function with the `-timer real` option as shown in "Using the profile Function to Change the Time Type Used by the Profiler" on page 9-48.

4. 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 9-35.

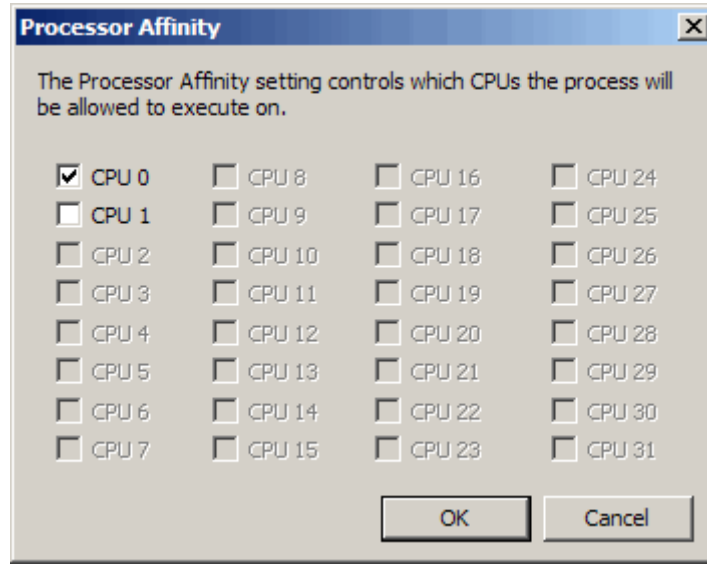
5. If you restricted the number of active CPUs in Step 1, reset the number of active CPUs to the original setting.

Intel Multi-Core Processors — Setting for Most Accurate Profiling on Windows Systems. If your system uses Intel multicore chips, and you plan to profile using CPU time, set the number of active CPUs to one before you start profiling. This results in the most accurate and efficient profiling.

1. Open Windows Task Manager.
2. On the **Processes** tab, right-click `MATLAB.exe` and then click **Set Affinity**.

   The Processor Affinity dialog box opens.
3. In the Processor Affinity dialog box, note the current settings, and then clear all the CPUs except one.

   Your Processor Affinity dialog box should appear similar to the following image:



4. Click **OK**.
5. Reset the state of the Profiler so that it recognizes the processor affinity changes you made. The easiest way to do this is to change the Profiler

timer setting from `real` and then back to `cpu`, by issuing the following in the Command Window:

```
profile -timer real
profile -timer cpu
```

Remember to set the number of CPUs back to their original settings when you finish profiling by rerunning the preceding steps, and restoring the original selections in the Processor Affinity dialog box in Step 3.

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. If MATLAB can determine the amount of time spent for profiling overhead, MATLAB excludes it from the self time also. (MATLAB excludes profiling overhead from the total time and the time for individual lines in the Profile Detail Report as well.)

The bottom of the Profiler page contains a message similar to one of the following, depending on whether MATLAB can determine the profiling overhead:

- Self time is the time spent in a function excluding both the time spent in its child functions and most of the overhead resulting from the process of profiling. In the present run, self time excludes 0.240 seconds of profiling overhead. The amount of remaining overhead reflected in self time cannot be determined, and therefore is not excluded.
- Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes some overhead resulting from the process of profiling.

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.

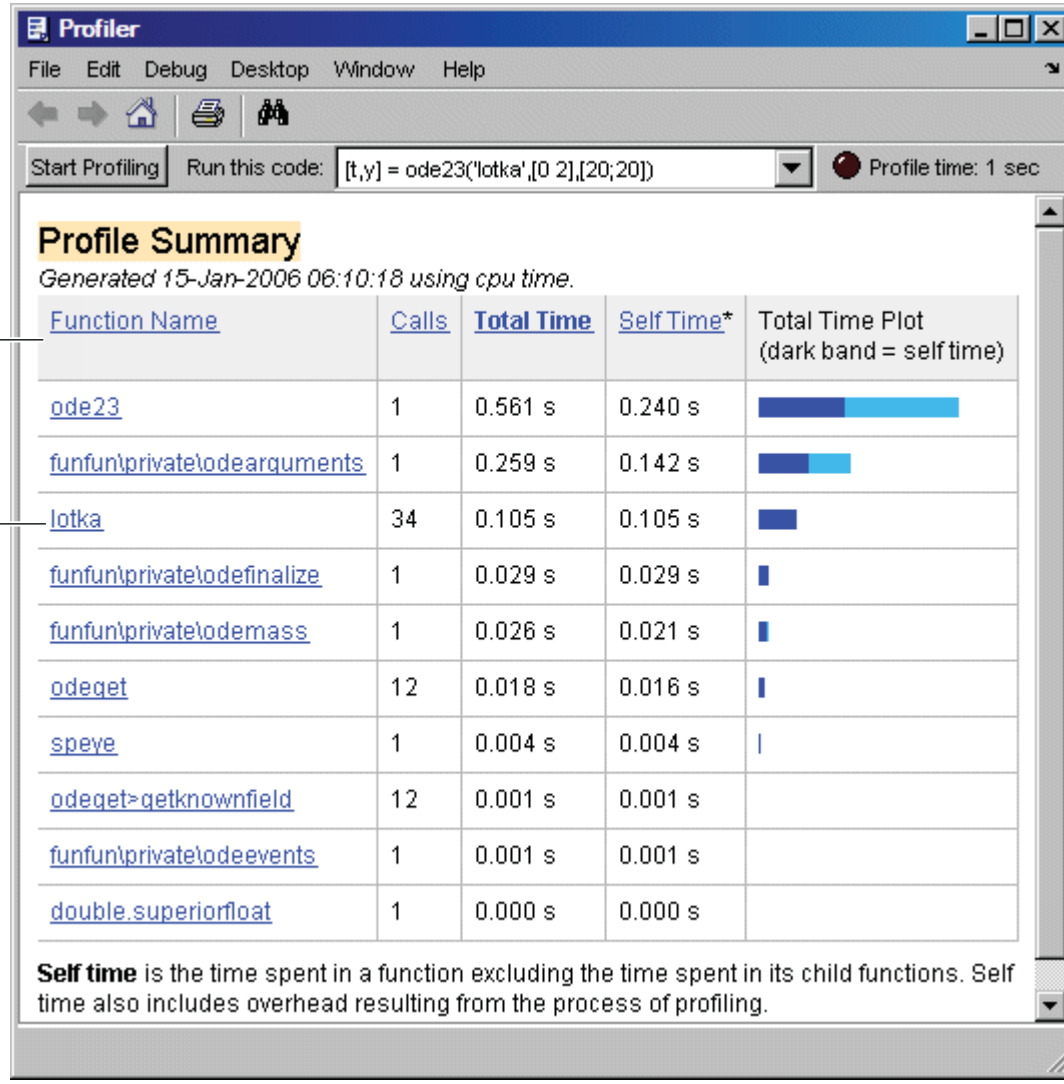
Following is the summary report for the Lotka-Volterra model described in "Example: Using the profile Function" on page 9-45.

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 9-37 for more information.

Click any column label to sort by that column.

Click any function name to display the detailed report.



Profile Detail Report

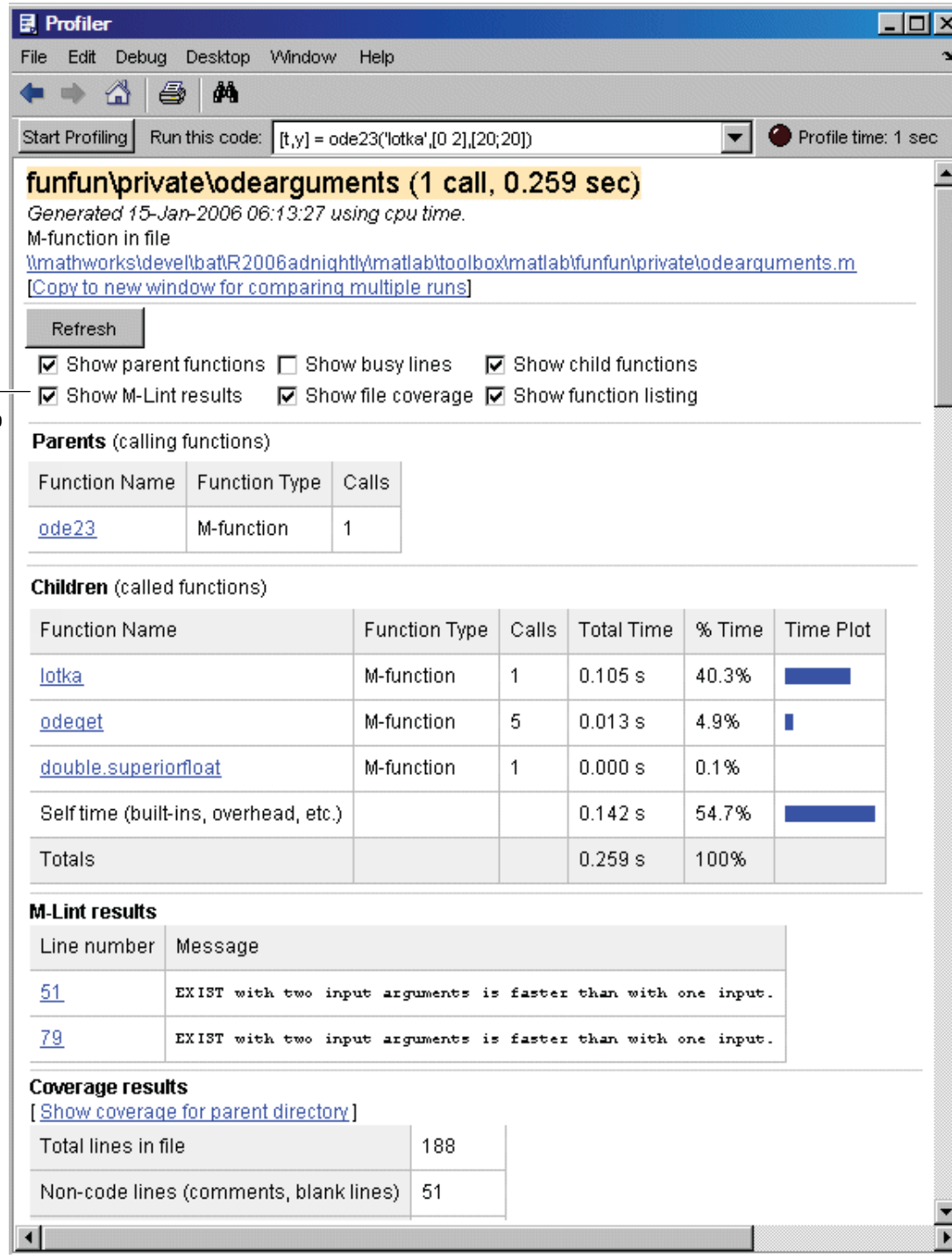
The Profile Detail report shows profiling results for a selected function that was called during profiling. A Profile Detail report is made up of seven sections, summarized below. By default, the Profile Detail report includes

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.

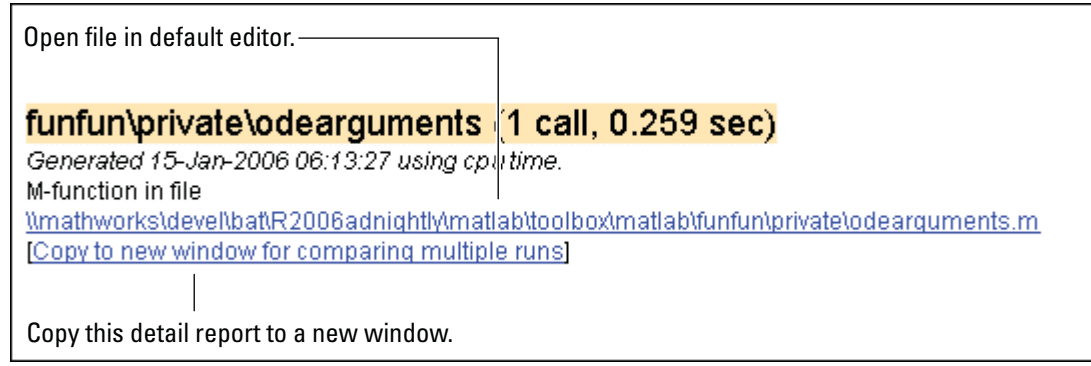


Profile Detail Report Header

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.

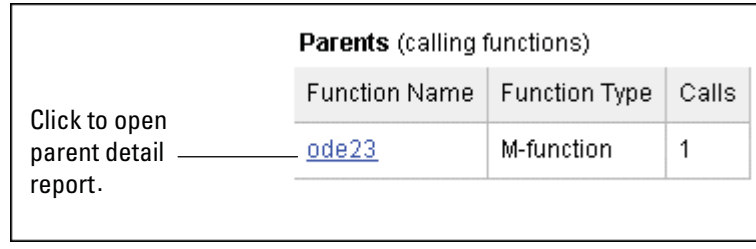
The header includes a link that opens the function in your default text editor.

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 products from The MathWorks, that is, files in `matlabroot/toolbox` folders.



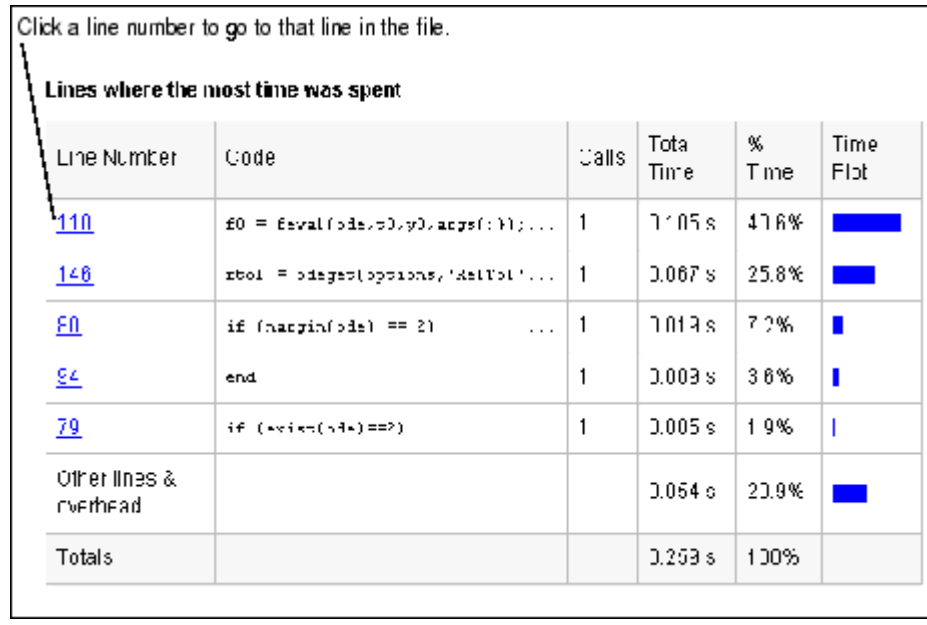
Parent Functions

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.



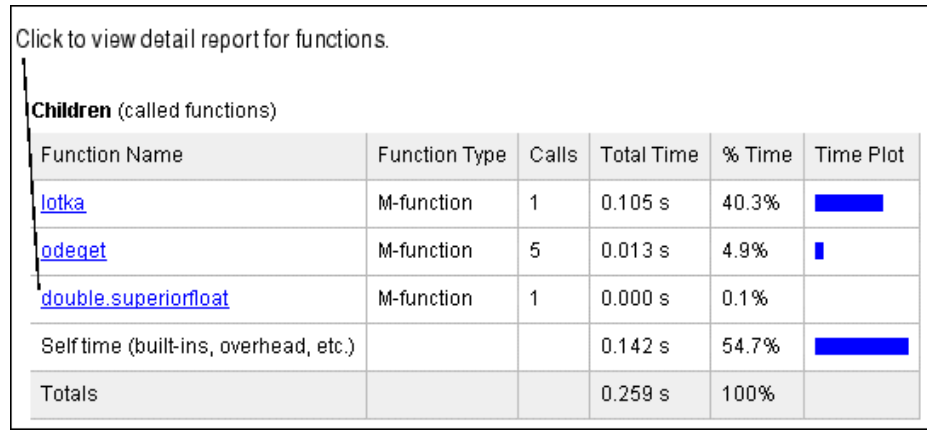
Busy Lines

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.



Child Functions

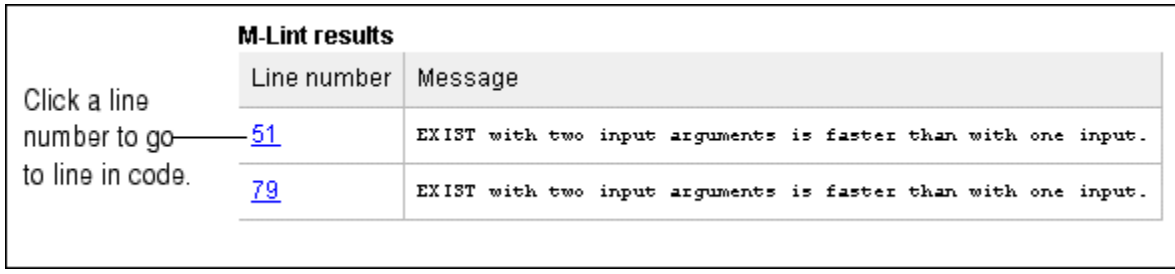
To include the **Children** section of the detail report, select the **Show child functions** check box. This section of the report lists all the functions called by the profiled function. If the called function is an M-file, you can view the source code for the function by clicking its name.



Function Name	Function Type	Calls	Total Time	% Time	Time Plot
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

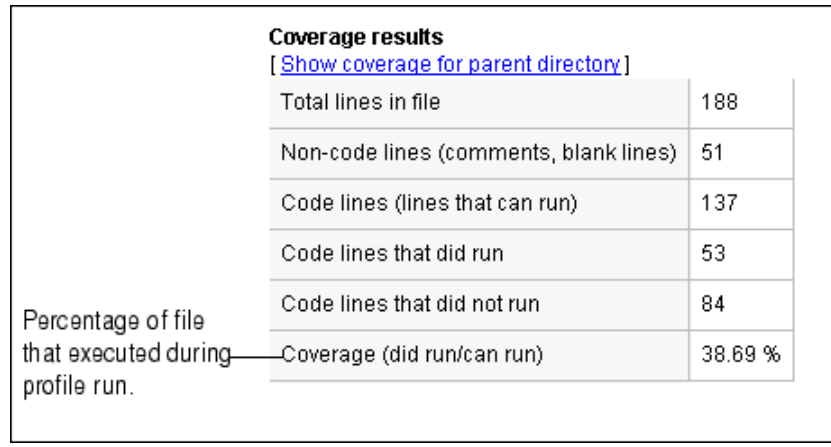
To include the **M-Lint results** section in the detail report display, select the **Show M-Lint results** check box. This section of the report provides information about problems and potential improvements, generated by M-Lint about the function. For more information about M-Lint, see "M-Lint Code Check Report" on page 9-22.



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.

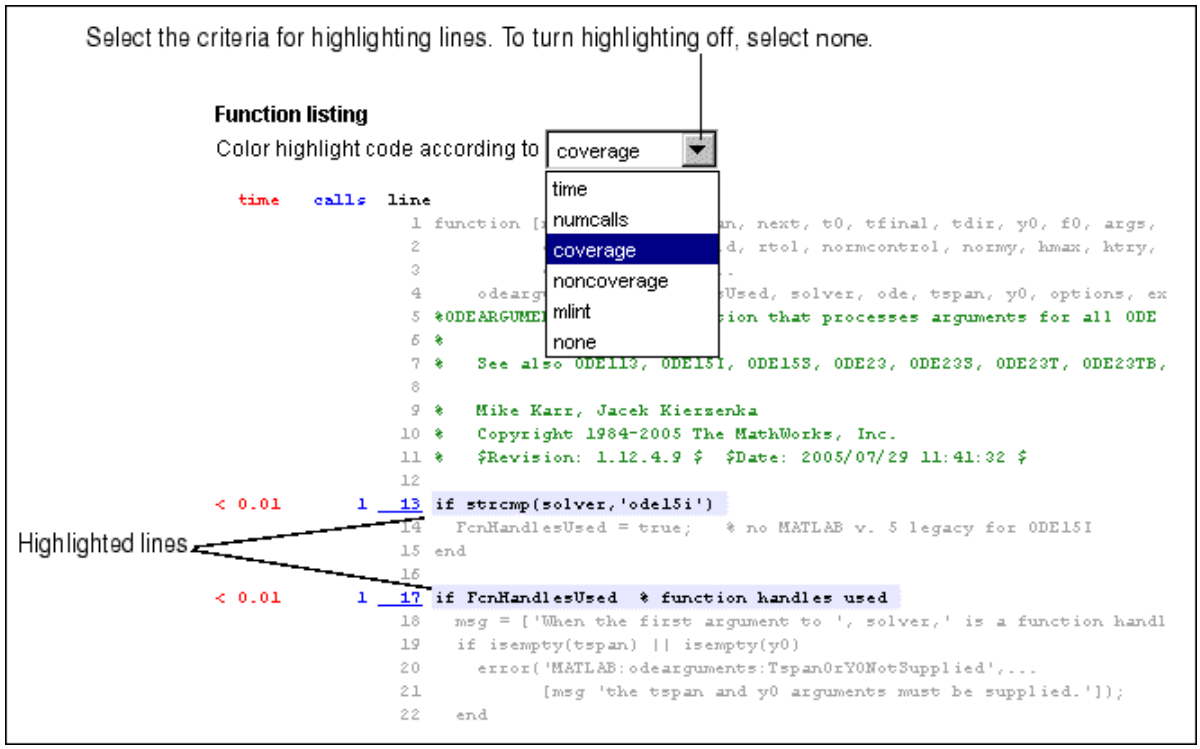


Function Listing

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.



The profile Function

The Profiler is based on the results returned by the `profile` function. The `profile` function provides some features that are not available in the GUI. For example, using the function, you can specify that the statistics display the time it takes for statements to run as clock time instead of CPU time.

This section includes the following topics with respect to the `profile` function:

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 subfolder of the current folder named `profile_results`, and displays the summary report in your system browser. You can specify another folder name as an optional second argument to `profsave`.

Accessing profile Function Results

The `profile` function returns results in a structure. This example illustrates how you can access these results:

1 To start `profile`, specifying the `history` option, type in the Command Window.

```
profile on -history
```

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 Function 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)
```

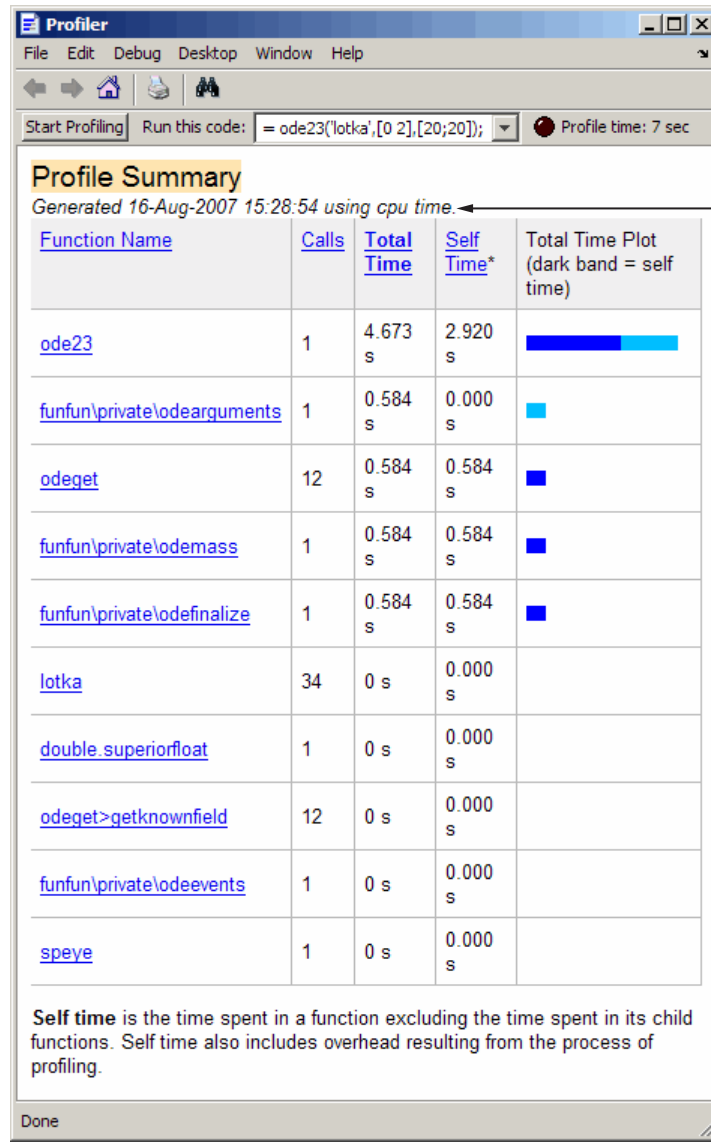
By default, `profsave` puts these HTML files in a subfolder of the current folder named `profile_results`. You can specify another folder name as an optional second argument to `profsave`.

```
profsave(stats,'mydir')
```

Using the profile Function to Change the Time Type Used by the Profiler

By default, MATLAB generates the Profiler Summary report using CPU time, as opposed to real (wall clock) time. This example illustrates how you can direct MATLAB to use real time instead.

The following image shows the Profiler Summary report as it appears by default, using CPU time:



Specify that the Profiler use real time instead, by using the `profile` function with the `-timer real` option, as shown in this example:

1. If the Profiler is currently open, close the Profiler, and if prompted, stop profiling.

2. Set the timer to real time by typing the following in the Command Window:

   ```
   profile on -timer real
   ```

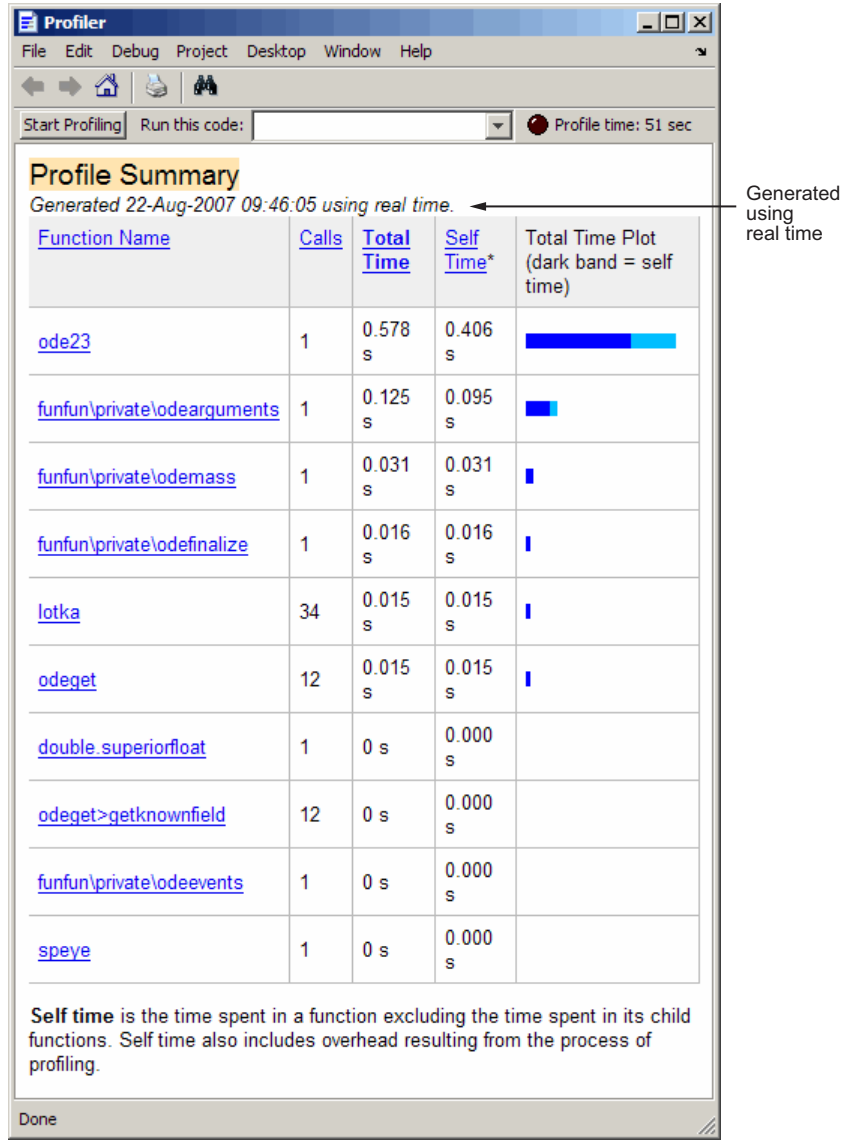
3. Run the M-file that you want to profile. This example runs the Lotka-Volterra predator-prey population model.

   ```
   [t,y] = ode23('lotka',[0 2],[20;20]);
   ```

4. Open the Profiler by typing the following in the Command Window:

   ```
   profile viewer
   ```

   The Profiler opens and indicates that real time is being used, as shown in the following image:



5 To change the timer back to using CPU time:

a Close the Profiler, and if prompted, stop profiling.

b Type the following in the Command Window:

```
profile on -timer cpu
```

c Type the following in the Command Window to reopen the Profiler:

```
profile viewer
```

10

Publishing M-Files

MATLAB software lets you to format M-files and publish them to various output file formats.

Overview of Publishing M-Files

In this section...

What Is Meant by Publishing M-Files?

The MATLAB product allows you to quickly publish your M-file code to enable you to describe and share your code with others who may or may not have MATLAB software. You can include the following within the published M-file:

- Formatted commentary on the code, including bulleted and numbered lists, bold and monospace font, preformatted text, TeX equations, and so on
- M-file code
- Results of running the code, including output to the Command Window and figures created or modified by the code

You can publish in various formats, including HTML, XML, and LaTeX. If Microsoft Word or Microsoft® PowerPoint® applications are on your Microsoft Windows system, you can publish to their formats as well.

If you have an active Internet connection, you can watch the Publishing M Code from the Editor video demo for an overview of the major publishing features using cells with text markup.

Using Cells

After you write and debug an M-file and are ready to share it with others, you can insert cells and commentary using text markup features available in MATLAB. This enables you to publish a formatted M-file. A cell is a section of M-file code (see "What Are Cells?" on page 8-185). For the purposes of publishing, a cell can be a section of the code that you want to present as a

titled subsection within the published document, or a portion of code for which you want the results of code evaluation to display as it occurs (for example, each iteration of a `for` loop), or both.

Any cell features that you use for evaluating and improving your code, as described in "Using Cells for Rapid Code Iteration and Publishing Results" on page 8-185, you also can use for publishing purposes. However, to have formatted comments in the output document, those comments must appear at the start of a cell, before any executable code. This requirement might mean that you need to change cells that you inserted for rapid code iteration. If you do so, be aware that this changes the cells for evaluation purposes, as well.

"Example of a Published M-File" on page 10-4 shows how the cells and formatted comments appear when an M-file is published.

Although you typically include the text markup after you write and debug the code, you can also include text markup as you write the code, or a combination of the two.

Note Cell mode is supported for use with M-files only. It is not intended for use with plain text files.

Process for Publishing M-Files

The overall process to publish an M-file using cell features in the Editor is as follows:

1. Open your M-file in the Editor.
2. Select **Cell > Insert Text Markup** as described in "Formatting M-File Comments for Publishing" on page 10-18.

   This enables you to specify how M-file comments appear in the published document. For example, you can specify that comments appear as bold or monospaced text in the published document.
3. To publish the M-file, do one of the following, as described in "Producing Published Output from M-Files" on page 10-68:

- To publish the M-file with default publishing properties, select **File > Publish *`file name`***. When you use this method, MATLAB publishes the M-file to HTML in an `/html` subfolder of the folder that contains the M-file you are publishing. However, if you previously specified custom property values, as described in the next list item, the last configuration you specified is used and the output file formats and folder may be different.
- To specify custom publishing properties, select **File > Publish Configuration for *`file name`* > Edit Publish Configurations for *`file name`***, adjust properties, and then click **Publish**. You can, for example, choose to include or exclude the executable code from the published document.

Example of a Published M-File

This section provides an example to demonstrate how an M-file appears when published. It shows how the M-file appears before and after text markup is added to cells to achieve the formatted results. This section contains the following topics:

For detailed information on inserting text markup, see "Formatting M-File Comments for Publishing" on page 10-18.

Sample M-File Before Formatting

```
function fourier_demo
    t = 0:.1:pi*4;
    y = sin(t);
    updatePlot(t,y);

    % In each iteration of the for loop add an odd
    % harmonic to y. As "k" increases, the output
    % approximates a square wave with increasing accuracy.

    for k = 3:2:9
```

```
        % Perform the following mathematical operation
        % at each iteration:
        y = y + sin(k*t)/k;

        display(sprintf('When k = %.1f',k));
        display('Then the plot is:');
        updatePlot (t,y)
    end

end

% Even though the approximations are constantly
% improving, they will never be exact because of the
% Gibbs phenomenon, or ringing.

function updatePlot(t,x)
% Subfunction to update the plot
    cla
    plot(t,x)
end
```

Published Sample M-File Before Formatting

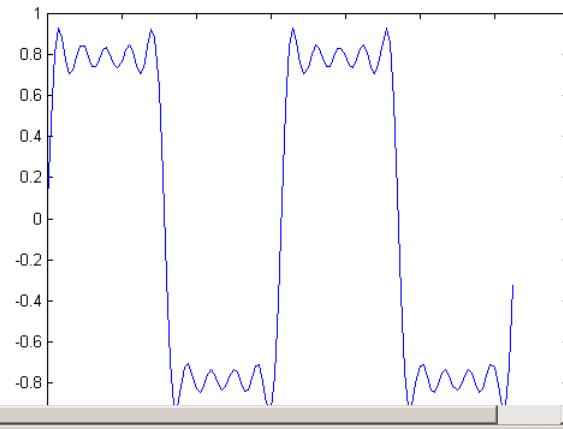
Before you add text markup and cell breaks, publishing `fourier_demo.m` includes the last plot generated by the `for` loop, but otherwise, has little effect. For example, if you select **File > Publish fourier_demo.m**, the published results, as shown in the following figure, are of limited use.

fourier_demo

File Edit View Go Debug Desktop Window Help

```
function fourier_demo
    t = 0:.1:pi*4;
    y = sin(t);
    updatePlot(t,y);

    % In each iteration of the for loop add an odd
    % harmonic to y. As "k" increases, the output
    % approximates a square wave with increasing accuracy.

    for k = 3:2:9
        % Perform the following mathematical operation
        % at each iteration:
        y = y + sin(k*t)/k;

        % Update the plot at each iteration using the
        % updatePlot subfunction.
        display(sprintf('When k = %.1f',k));
        display('Then the plot is:');
        updatePlot (t,y)
    end

end

% Even though the approximations are constantly
% improving, they will never be exact because of the
% Gibbs phenomenon, or ringing.

function updatePlot(t,x)
% Subfunction to update the plot
    cla
    plot(t,x)
end
```

```
When k = 3.0
Then the plot is:
When k = 5.0
Then the plot is:
When k = 7.0
Then the plot is:
When k = 9.0
Then the plot is:
```

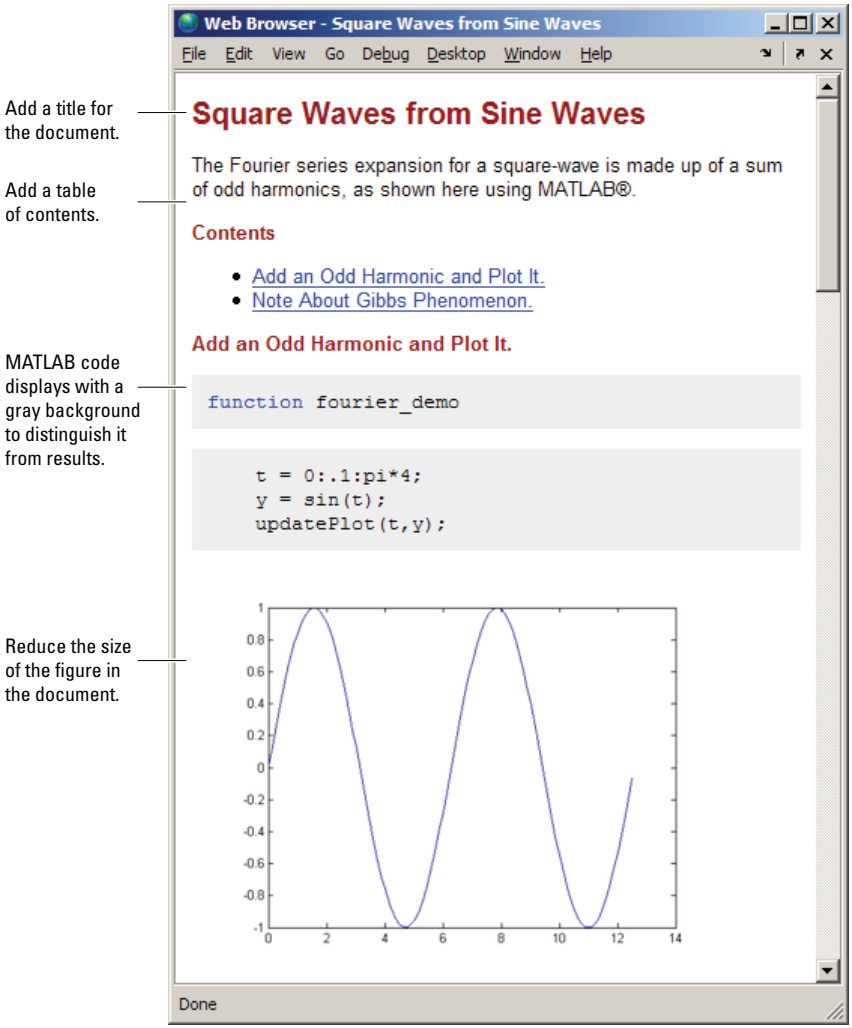


Done

Published Sample M-File After Formatting

If you add a few comments for clarity, apply text markup, and insert cell breaks, as described in “Producing the Formatting for the Example” on page 10-11, the published M-file is transformed as shown in the following three figures:

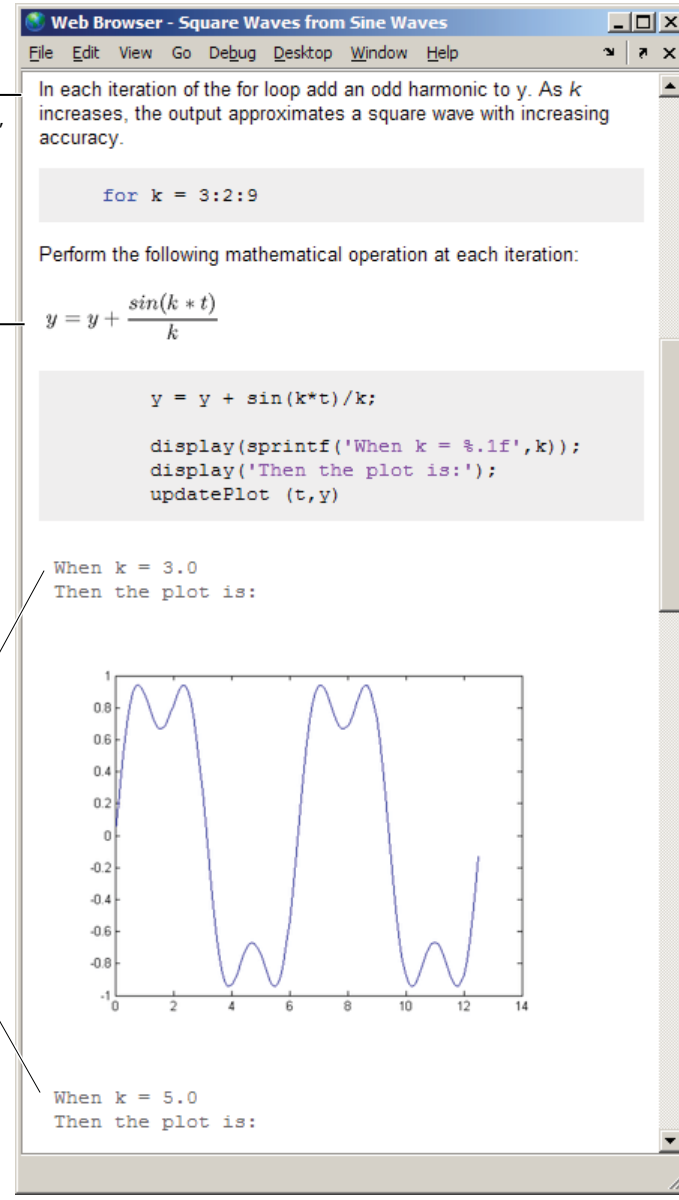
- The first figure shows the top of the published document.
- The second figure shows the middle of the published document.
- The third figure shows the bottom of the document.

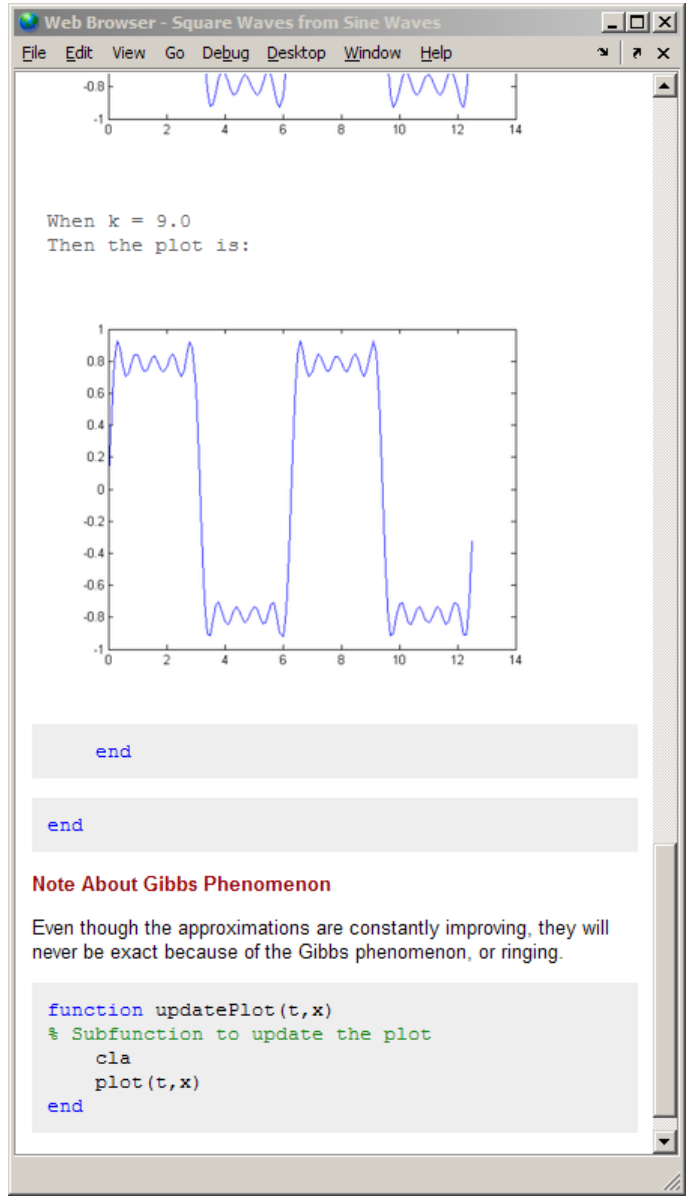

Add a title for the document.
Add a table of contents.
MATLAB code displays with a gray background to distinguish it from results.
Reduce the size of the figure in the document.
Web Browser - Square Waves from Sine Waves
File Edit View Go Debug Desktop Window Help
Square Waves from Sine Waves
The Fourier series expansion for a square-wave is made up of a sum of odd harmonics, as shown here using MATLAB®.
Contents
Add an Odd Harmonic and Plot It.
Note About Gibbs Phenomenon.
Add an Odd Harmonic and Plot It.
function fourier_demo
t = 0:.1:pi*4;
y = sin(t);
updatePlot(t,y);
Done

Make selected comment text, such as the k here, appear in italic.

Publish an equation using TeX format.

Include each iteration of the for loop in the document.




Web Browser - Square Waves from Sine Waves
File Edit View Go Debug Desktop Window Help
When k = 9.0
Then the plot is:
end
end
Note About Gibbs Phenomenon
Even though the approximations are constantly improving, they will never be exact because of the Gibbs phenomenon, or ringing.
function updatePlot(t,x)
% Subfunction to update the plot
cla
plot(t,x)
end

Producing the Formatting for the Example

The following steps apply text markup to the `fourier_demo.m` file. When published to HTML, the results appear as shown in "Published Sample M-File After Formatting" on page 10-7.

For detailed information about each **Cell** menu option, see "Formatting M-File Comments for Publishing" on page 10-18.

1. Open `fourier_demo.m` by running the following command:

   ```
   edit(fullfile(matlabroot,'help','techdoc',...
   'matlab_env','examples','fourier_demo.m'))
   ```

   To work with `fourier_demo.m` on your system, save the file to a folder for which you have write permission. In the example, the file is saved to `I:\my_matlab_files\my_mfiles\fourier_demo.m`.

2. Enable cell mode by selecting **Cell > Enable Cell Mode.**

3. 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 double percent signs (`%%`) indicate the start of a new cell. A single percent sign indicates the beginning of a comment line.

   b. Replace **`DOCUMENT TITLE`** with **`Square Waves from Sine Waves`**.

   c. Replace `% INTRODUCTORY TEXT` with one or more comments about the overall file, for example:

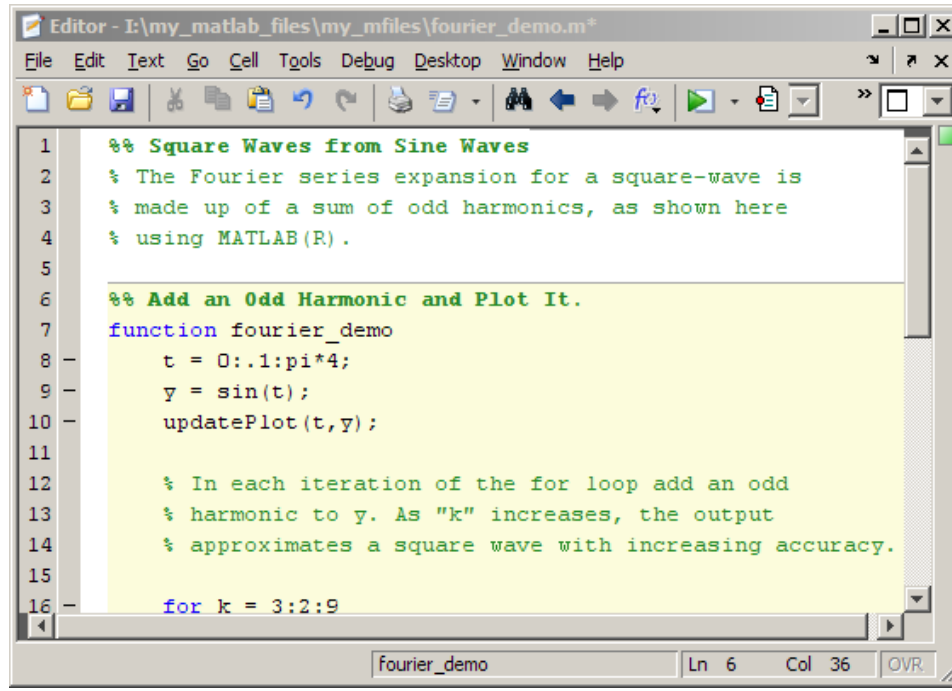
      ```
      % The Fourier series expansion for a square-wave is
      % made up of a sum of odd harmonics, as shown here
      % using MATLAB(R).
      ```

      The string "(R)" will appear as a registered trademark symbol in the published document.

d On line 5, insert a blank line for better readability. Notice that the file now contains two cells. The first cell extends from line 6 to the top of the file; the second cells extends from line 6 to the bottom of the file. The cell break at line 6, splits the file into two cells.

4 On line 6, where the second cell begins (as indicated by %%), type a title for the cell: `Add an Odd Harmonic and Plot It`.

Notice that when you move from one cell to the next in the file, the highlighting in the M-file indicates which cell the cursor is currently in.



5 To display the text that describes the purpose of the loop in the published document as explanatory text, rather than M-file code, insert a cell break before the explanation. That is:

a Place the cursor at line 12.

b Select **Cell > Insert Cell Break**.

Note that the cell that begins on line 12, continues to the end of the `fourier_demo` function. If you insert a cell break anywhere within the code block, MATLAB inserts an implicit cell break at the end of a code block. A *code block* is the body of any programming control statement or function.

6 Remove the quotation marks around the `k` at line 14 and present it in italic instead:

a Delete the quotation marks.

b Select the letter `k`.

c Select **Cell > Insert Text Markup > Italic Text**.

Instead of being enclosed in quotation marks, the letter now appears as `_k_`.

d To see the effect, click Publish  .

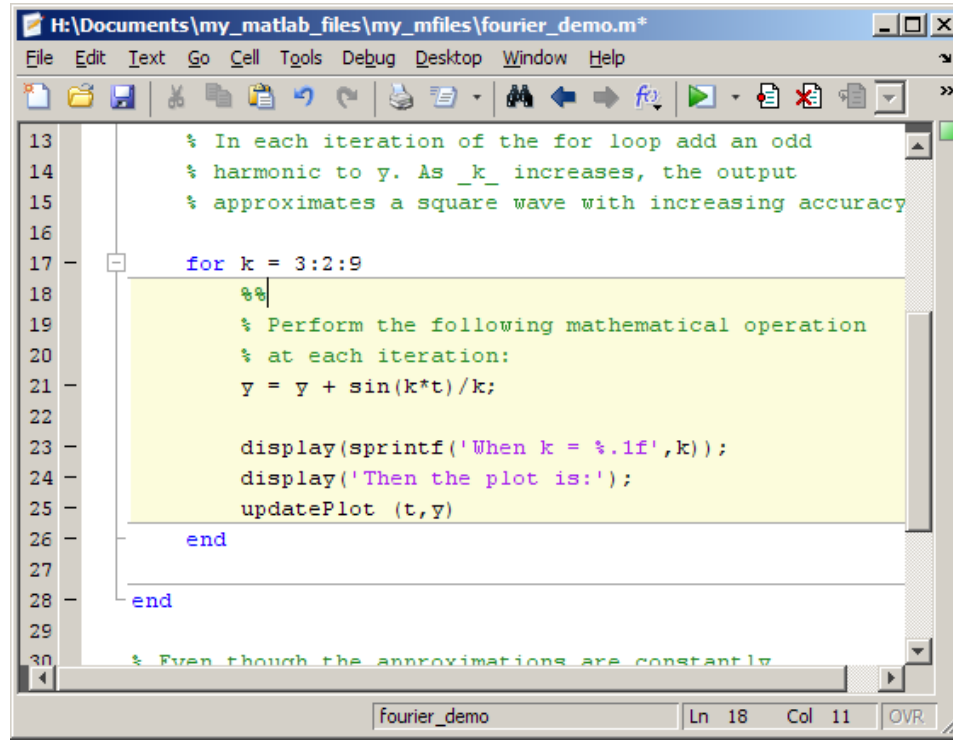
7 Because MATLAB publishes output generated by code immediately after the end of the cell that contains the code, the current cell would cause MATLAB to include the phrase `When k = n Then the plot is:` four times in succession in the published document. In addition, only the final plot generated by the `for` loop would be in the published document.

To have MATLAB include every plot generated by the `for` loop in the published document, each preceded by the phrase `When k = n ...`, create a cell within the `for` loop, as follows:

a Place the cursor at the end of line 17, after `for k = 3:2:9.`

b Select **Cell > Insert Cell Break**.

Now the current cell includes only the body of the `for` loop.



c To see the effect, click Publish .

8 Display equations with symbols and Greek characters (such as pi) using the TeX format. In this example, to create a comment containing a nicely formatted form of the equation, `y = y + sin(k*t)/k`, in the published document, use text markup as follows:

a Position the cursor at the end of the comment on line 20, `% at each iteration.`

b Select **Cell > Insert Text Markup > TeX Equation**.

MATLAB inserts the following lines; the second line is a sample equation with text markup applied:

```
%
% $$e^{\pi i} + 1 = 0$$
%
```

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:

```
y = y + \frac{sin(k*t)}{k}
```

The three lines that display the TeX equation now appear as follows in the M-file:

```
%
% $$y = y + \frac{sin(k*t)}{k} $$
%
```

d To see the effect, click Publish  .

9 Reduce the size of the published figures by editing the publish configuration for the file:

a Select **File > Publish Configuration for fourier_demo > Edit Publish Configurations for fourier_demo.m**.

The Edit M-File Configurations dialog box opens.

b In the column to the right of **Max image width (pixels)**, double-click **Inf**, and type the value `350`.

c In the column to the right of **Max image height (pixels)**, double-click **Inf**, and type the value `350`.

d Click **Save As**. The Save Publish Settings dialog box opens.

e In the **Settings name** field, type `small_images`, and then click **Save**.

f Click **Close**.

g To see the effect, click Publish  .

10 To create a section header without including a cell break, follow these steps:

a Position the cursor at the beginning of line 33, where the comment `% Even though the approximations are constantly` appears.

b Select **Cell > Insert Text Markup > Section Title without Cell Break.**

c Replace `SECTION TITLE` with `Note About Gibbs Phenomenon`.

d Delete line 34, where the comment `% DESCRIPTIVE TEXT` appears.

11 Select **File > Publish fourier_demo**.

The published document, an HTML file, appears in the MATLAB Web Browser, as shown in "Published Sample M-File After Formatting" on page 10-7.

By default, MATLAB stores the HTML document, `fourier_demo.html`, and the associated image files in an `/html` subfolder within the folder containing the source M-file.

See "M-File Code After Text Markup" on page 10-16 for the resulting M-file code.

M-File Code After Text Markup

After adding text markup, the `fourier_demo.m` M-file appears as follows. When you publish the file to HTML, it appears as shown in "Published Sample M-File After Formatting" on page 10-7.

```
%% Square Waves from Sine Waves
% The Fourier series expansion for a square-wave is
% made up of a sum of odd harmonics, as shown here
% using MATLAB(R).

%% Add an Odd Harmonic and Plot It
function fourier_demo
    t = 0:.1:pi*4;
    y = sin(t);
    updatePlot(t,y);

    %%
    % In each iteration of the for loop add an odd
    % harmonic to y. As _k_ increases, the output
    % approximates a square wave with increasing accuracy.

    for k = 3:2:9
        %%
        % Perform the following mathematical operation
        % at each iteration:
```

```
        %
        % $$ y = y + \frac{sin(k*t)}{k} $$
        %
        y = y + sin(k*t)/k;

        display(sprintf('When k = %.1f',k));
        display('Then the plot is:');
        updatePlot (t,y)
    end

end

%% Note About Gibbs Phenomenon
% Even though the approximations are constantly
% improving, they will never be exact because of the
% Gibbs phenomenon, or ringing.

function updatePlot(t,x)
% Subfunction to update the plot
    cla
    plot(t,x)
end
```

To open the formatted file in the Editor, instead of following the steps in "Producing the Formatting for the Example" on page 10-11 run the following command:

```
edit(fullfile(matlabroot,'help','techdoc',...
'matlab_env','examples','fourier_demo2.m'))
```

To work with `fourier_demo2.m` on your system, save the file to a folder for which you have write permission.

Formatting M-File Comments for Publishing

In this section...

Note Many examples in this section show the effects of publishing to HTML. For information on how to publish to HTML, see "Producing Published Output from M-Files" on page 10-68.

Overview of Formatting M-File Comments for Publishing

This section describes the types of text formatting you can specify for M-files, so that your published output appears like a polished document, rather than a text file of code. This enables you to single-source your M-file code with documentation that describes what the code is doing.

You can format M-file comments in either of the following ways to specify how the published results will appear:

- Use **Cell > Insert Text Markup** menu options to format the comments. This automatically inserts the text markup symbols for you.
- Type the markup symbols directly in the comments; the markup symbols you enter are the same as the text markup that results when you use the equivalent menu item. See "Summary of Markup for Formatting M-Files for Publishing" on page 10-61 for details.

You can use text markup as you create a new file, to mark up an existing M-file, or a combination of the two. When you use the **Cell** menu options, the Editor may insert more comment lines and other markup that you want. See "Cleaning Up Text Markup Before Publishing M-Files" on page 10-58 for information on how you can adjust the inserted text when you are done formatting a file.

Several examples in the formatting sections that follow use this M-file, `sine_wave.m`:

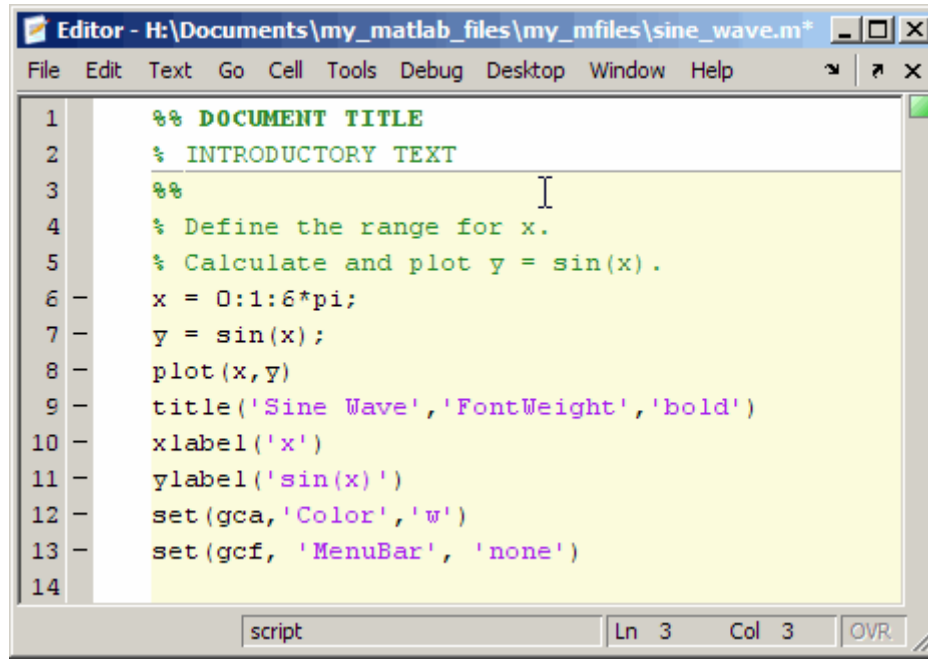
```
% Define the range for x.
% Calculate and plot y = sin(x).
% Display plot in published document.
x = 0:1:6*pi;
y = sin(x);
plot(x,y)
title('Sine Wave', 'FontWeight','bold')
xlabel('x')
ylabel('sin(x)')
set(gca, 'Color', 'w')
set(gcf, 'MenuBar', 'none')
```

Creating Document Titles and Introductory Text for Publishing an Existing M-File

To specify a document title and introductory text for an M-file, follow these steps:

1 In the Editor, position the cursor anywhere in an M-file.

2 Select **Cell > Insert Text Markup > Document Title and Introduction**. The first three lines of the file appear as shown in the following image.

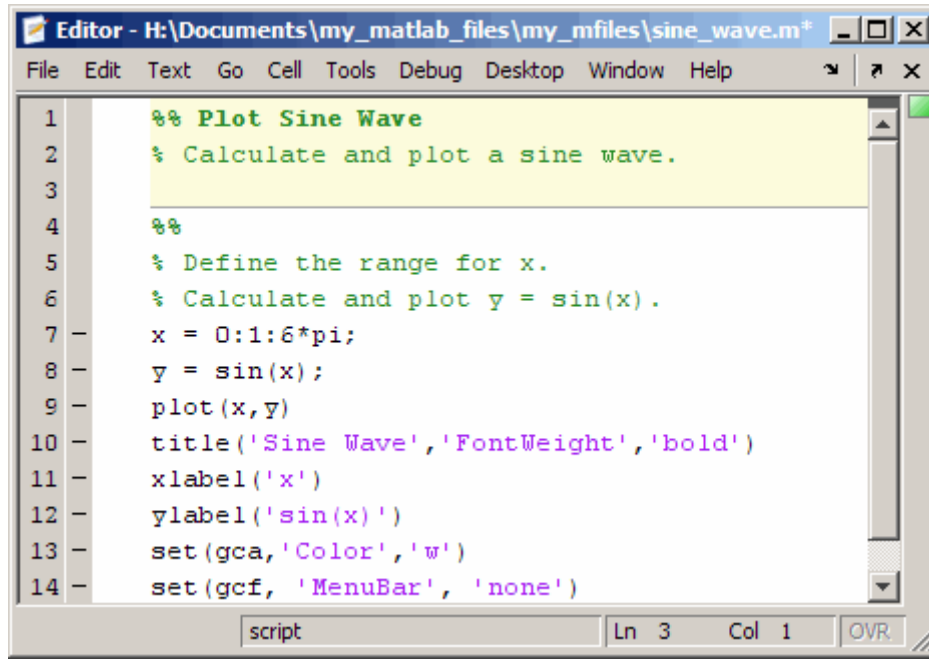


3 Replace **`DOCUMENT TITLE`** with the cell heading that you want to use; for example, **`Plot Sine Wave`**.

4 Replace `INTRODUCTORY TEXT` with text that introduces the M-file; for example, `Calculate and plot a sine wave`.

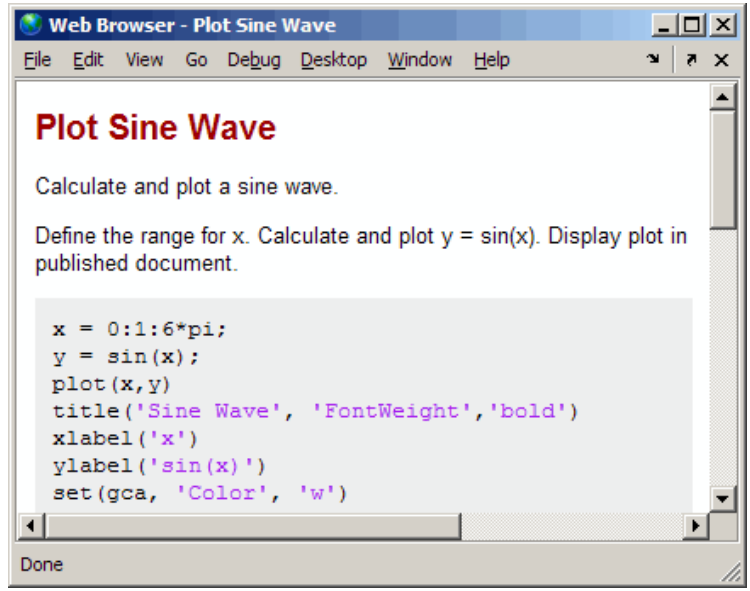
5 Insert a blank comment line to increase readability.

If you specify the example text suggested in the previous list, then the first four lines of the resulting M-file appear as follows.



Notice that a horizontal rule, which indicates a cell break, ends the title and introductory text. When you insert a document title and introduction, the Editor also adds a cell break in preparation for the first section within the M-file.

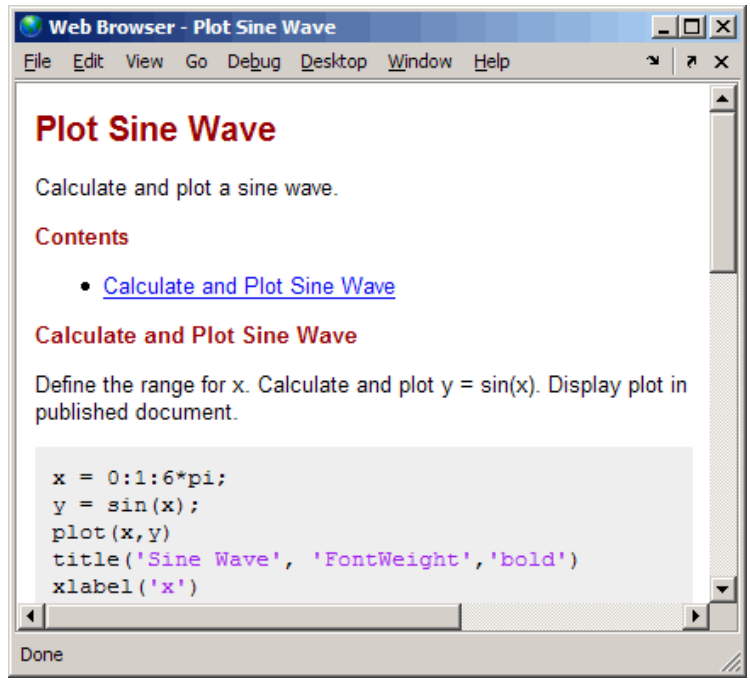
When you publish the M-file, the document title is formatted as a top-level heading (`h1` in HTML), using a large size, bold font; the introductory text appears as formatted text. The following figure shows the M-file published to HTML and presented in the MATLAB Web Browser.



Specifying a Title for the New Section that the Editor Inserts with the Document Title

When you follow the steps in the previous section, "Creating Document Titles and Introductory Text for Publishing an Existing M-File" on page 10-20, the first cell, demarcated in the Editor with the horizontal rule followed by a line with double percent signs (%%), is not evident in the published file because the section does not have a title.

To provide a title for the section, insert text after the double percent signs—for example, `Calculate and Plot Sine Wave`. When you republish the file to HTML, it appears in the MATLAB Web Browser as shown in the following image. Notice that MATLAB automatically inserts the Contents heading and the link to the section when you publish the file to HTML.



The file now has a document title, introductory text, and a first section. You can add more sections, as described in “Creating New Section Titles” on page 10-23.

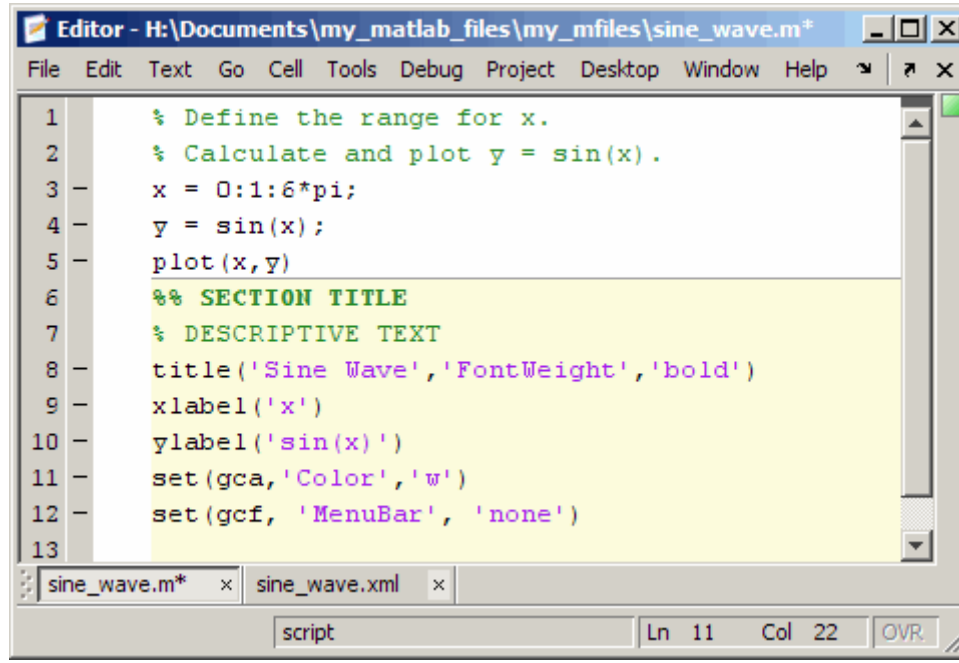
Note You can add any comments in the lines immediately following the title. However, if you want the title to appear as the overall document title, you cannot add any other text before the next cell (a line starting with `%%`) .

Creating New Section Titles

To insert a new section title and descriptive text within an M-file, follow these steps:

1. Position the cursor where you want to insert a new cell—before the `title` function shown in the previous example, for instance.

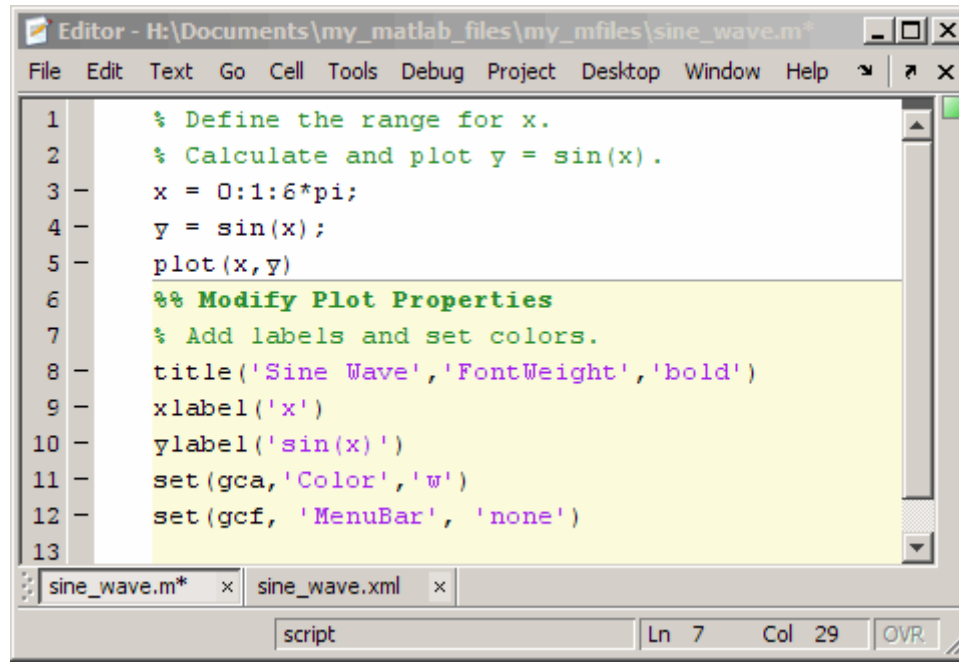
2 Select **Cell > Insert Text Markup > Section Title with Cell Break.** The file appears as follows.



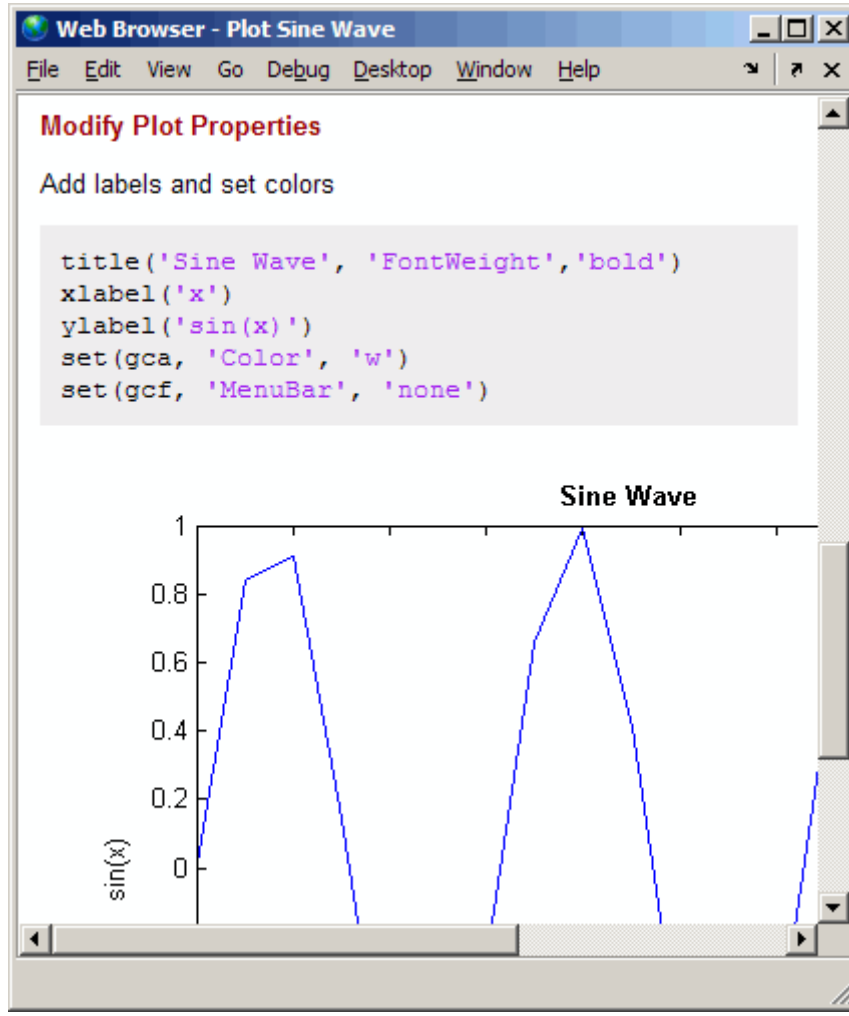
3 Replace **`SECTION TITLE`** with your title—**`Modify Plot Properties`**, for example.

4 Replace `DESCRIPTIVE TEXT` with text that describes the cell—`Add labels and set colors.`, for example.

If you specify the example text suggested in the previous list and "Specifying a Title for the New Section that the Editor Inserts with the Document Title" on page 10-22, then the resulting M-file appears as follows.



When you publish the M-file to HTML, the section title appears as a heading, using a medium size, bold font. Comments appear as formatted text in the published output. The following figure shows the results when you publish the updated `sine_wave.m` file to HTML output. Note that the `Modify Plot Properties` heading is formatted as an `h2`.



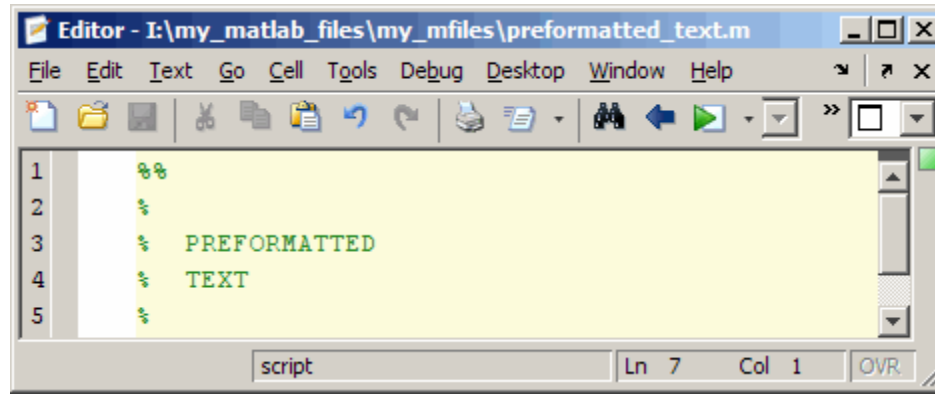
Specifying Preformatted Text in M-Files for Publishing

MATLAB software enables you to specify preformatted text in an M-file. Preformatted text appears in monospace font, maintains the white space that you specify and does not wrap long lines.

To insert preformatted text, follow these steps:

1. Position the cursor within the M-file where you want to insert preformatted text.

2. Select **Cell > Insert Text Markup > Preformatted Text**. Five lines of text are inserted, as shown.



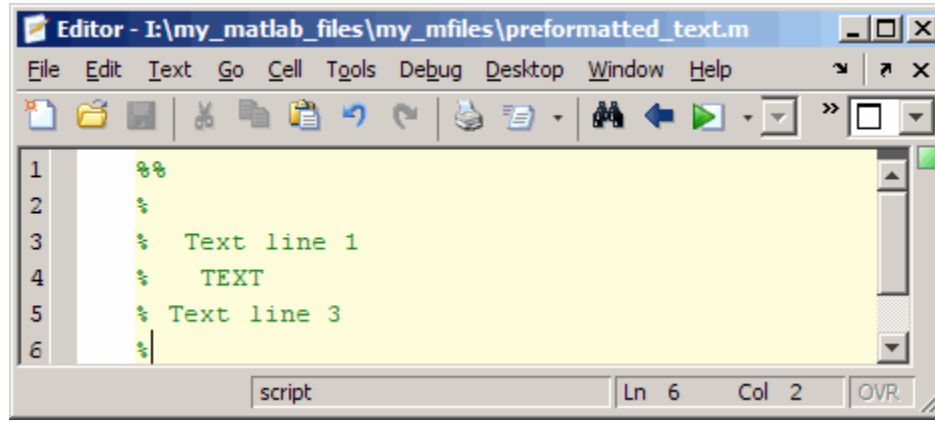
3. Being careful not to delete the two blank spaces before the word PREFORMATTED, replace the words PREFORMATTED and TEXT with your text, including tabs, spaces, and additional comment lines. For example:

   a. Replace `PREFORMATTED` with `Text line 1`.

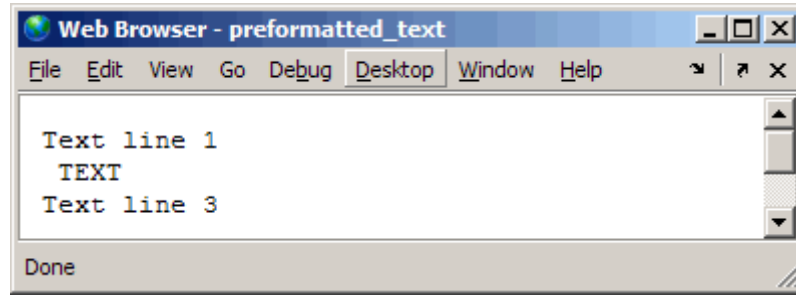
   b. Insert a tab before `TEXT` on line 4.

   c. On the last comment line type `Text line 3`.

   The resulting comments appear as follows.



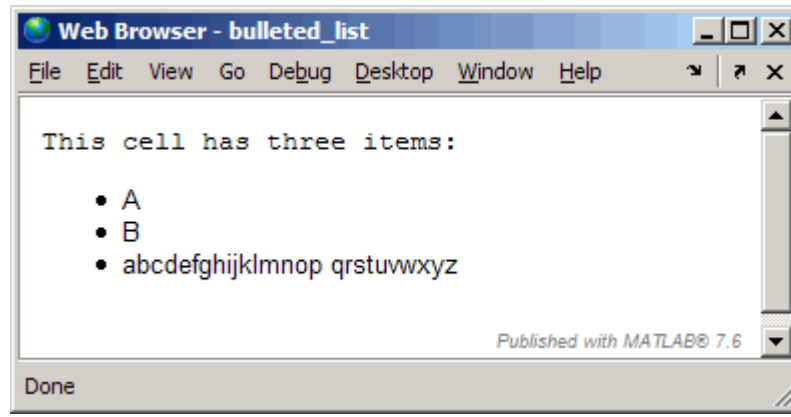
When you publish the M-file to HTML, the output appears as shown in the following figure.



If the results are not as you expect, see “About Formatted Blocks” on page 10-54.

Specifying Bulleted or Numbered Lists in M-Files for Publishing

The following steps describe how to specify text markup for a bulleted or numbered list so as to create a published document that appears as shown in the following image when you click the Publish button .

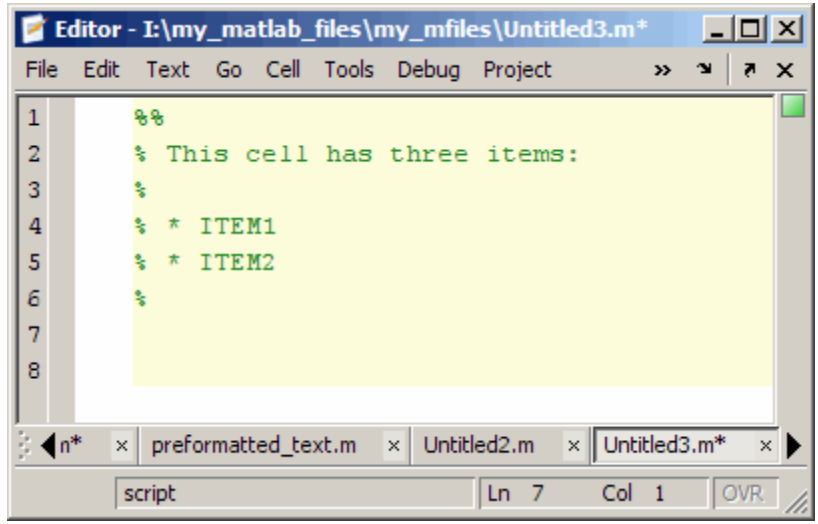


1. Position the cursor at the end of the line that precedes the location where you want to add a list. For example, if your M-file contains the following lines, position the cursor after the colon:

   ```
   %%
   % This cell has three items:
   ```

2. Select **Cell > Insert Text Markup > Bulleted List** or **Cell > Insert Text Markup > Numbered List**, depending on the type of list you want.

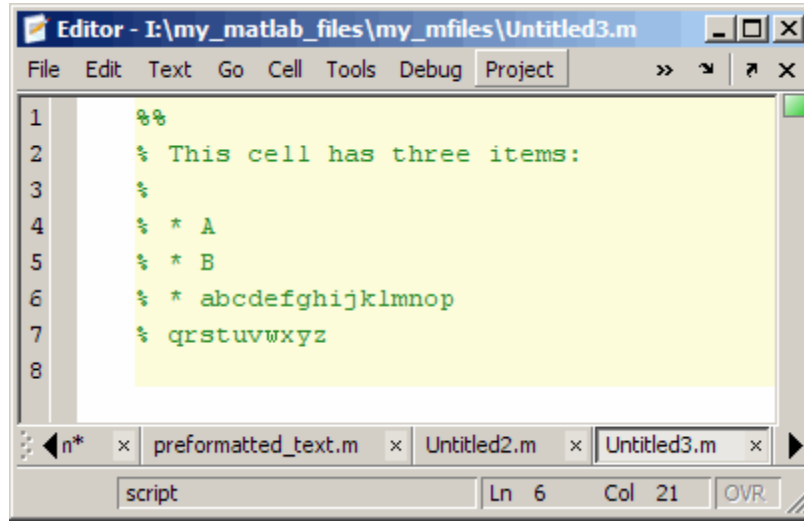
   MATLAB adds four lines of formatted comments to the M-file. The following figure shows the result when you insert a bulleted list.



If you insert a numbered list, the text markup is the same, except a number sign (#) indicates a numbered list item.

3 Replace the sample text, `ITEM1` and `ITEM2`, with your text. For example, replace `ITEM1` with `A` and replace `ITEM2` with `B`.

4 To create a multiline list item, break the line as desired, but do not insert the list item symbol (* or #) before the second line. For example, to insert the alphabet as a multiline list item, breaking the line at the letter `p`, type the alphabet as shown in the following figure.



Notice that the third list item is broken over two comment lines in the source, yet maintains the formatting of a list, as expected, when published (as shown at the beginning of this section).

If the results are not as you expect, see "About Formatted Blocks" on page 10-54.

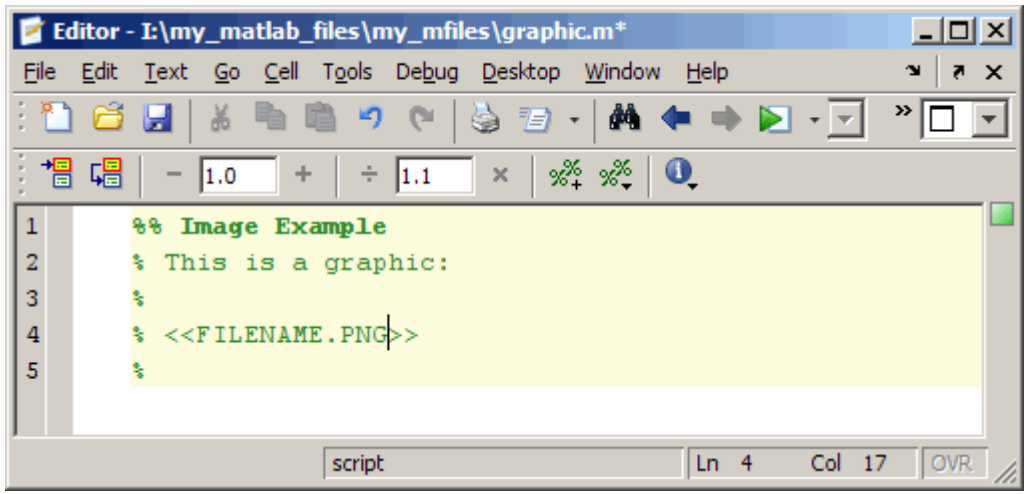
Specifying Graphics in M-Files for Publishing

You can insert text markup such that the published document includes an image that was not generated by the M-file code, as shown in the following example. (By default, images generated by the M-file code are included in the published document.)

1. Position the cursor where you want to add a graphic. For example, if your M-file contains the following lines, position the cursor after the colon:

```
%% Image Example
% This is a graphic:
```

2. Select **Cell > Insert Text Markup > Image**. MATLAB adds text markup, as shown in the following figure.



3 Replace `FILENAME.PNG`, with the file name of the graphic you want to insert, relative to the folder where MATLAB publishes the M-file.

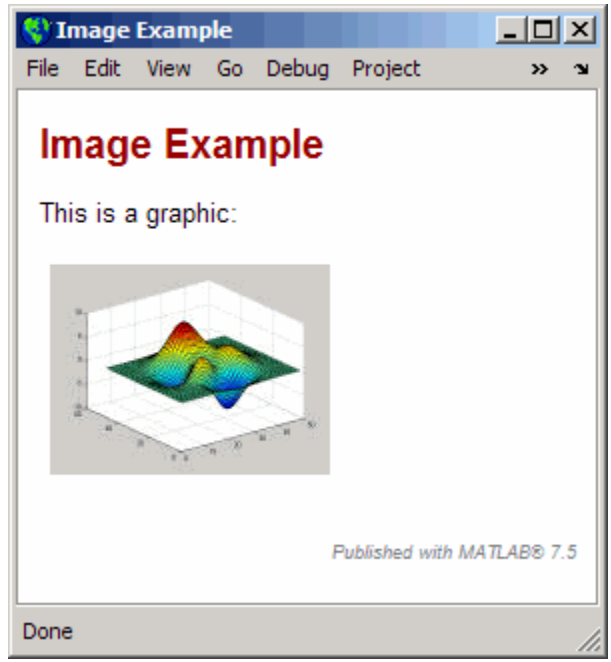
   For example, if you want to include the graphic, `surfpeaks.jpg`, and it is in the folder into which MATLAB publishes the M-file, then replace `FILENAME.PNG` with `surfpeaks.jpg`.

   By default, MATLAB publishes the M-file to an `/html` subfolder of the folder containing the M-file. You can change this folder, referred to as the output folder, by changing the publish configuration settings, as described in "Producing Published Output from M-Files" on page 10-68.

   If the graphic is not in the folder to which the M-file is published, then you must specify the location of the graphic file as a relative path from the location of the published output file, as illustrated in the following table.

Folder Containing the M-File	Folder Where MATLAB Publishes the M-File	Image File Location	How to Specify the Image in the M-File Comment
`I:/my_mfiles`	`I:/my_mfiles/html`	`I:/my_mfiles/html`	`% <<surfpeaks.jpg>>`
`I:/my_mfiles`	`I:/my_mfiles/doc`	`I:/my_mfiles/images`	`% <<../images/surfpeaks.jpg>>`

When you publish the M-file to HTML, the output appears as shown in the following figure.



If the results are not as you expect, see “About Formatted Blocks” on page 10-54.

Creating the surfpeaks.jpg Image

To create the `surfpeaks.jpg` image used in the preceding example, follow these steps:

1. Create an `html` subfolder in the folder where the M-file that references the graphic is located.

2. Enter the following in the Command Window:

   ```
   >> surf(peaks)
   ```

   A Figure window opens and displays the surfpeaks figure.

3 Save the figure as `surfpeaks.jpg` in the `html` subfolder that you created in step 1.

Note Unless you reduce the size of `surfpeaks.jpg`, it will appear larger than that shown in the previous example.

Using HTML Markup Tags in M-Files for Publishing

You can use the **Cell** menu to insert HTML code into your M-file. When you do so, the Editor inserts HTML code for a one-column, two-row table. You can use the inserted code as a guideline for inserting other HTML code.

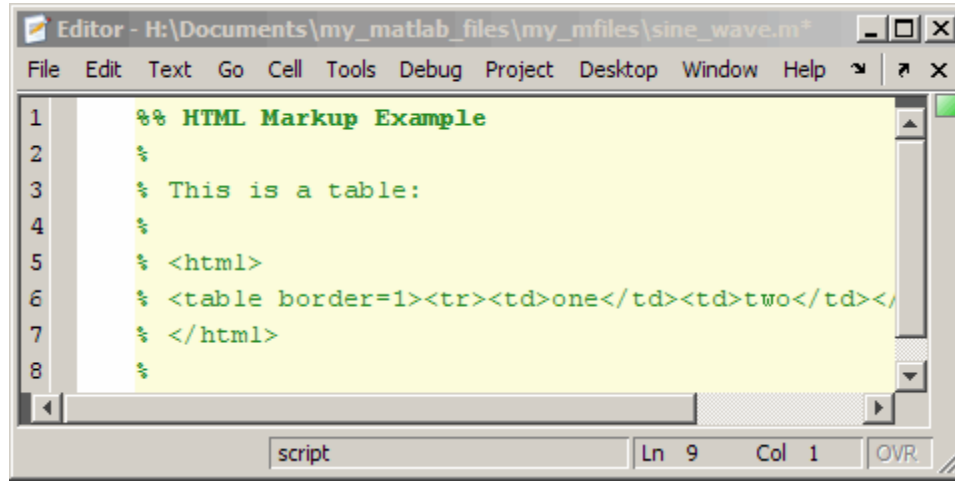
Note When you insert text markup for HTML code, the HTML code is published only when the specified output file format is HTML. For example, if you add HTML markup, but then specify LaTeX as the output file format, MATLAB software does not publish the text enclosed within the HTML markup. See "Producing Published Output from M-Files" on page 10-68 for information on specifying the output file format.

To insert the text markup for HTML code, follow these steps:

1 Position the cursor at the end of the comment that precedes the location where you want to insert HTML code. For example, if the M-file contains the following lines, position the cursor after the colon:

```
%% HTML Markup Example
% This is a table:
```

2 Select **Cell > Insert Text Markup > HTML Markup**. MATLAB adds HTML markup, as shown in the following figure.



3 Edit the inserted HTML code to specify the HTML code that you want to use.

If you publish the M-file to HTML and leave the inserted HTML code as is, MATLAB creates a single-row table with two columns, containing the values `one` and `two` as shown in the following figure.

If the results are not as you expect, see “About Formatted Blocks” on page 10-54.

Using LaTeX Markup for Publishing

You can use the **Cell** menu to insert LaTeX code. You can use the inserted code as a guideline for inserting other LaTeX code.

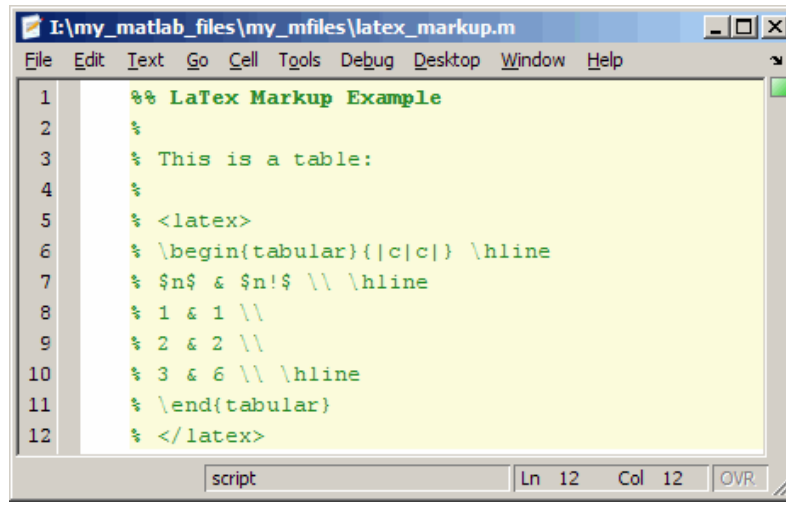
Note When you insert text markup for LaTeX code, that code is published only when the specified output file format is LaTeX. For example, if you add LaTeX markup, but then specify HTML as the output file format, MATLAB software does not publish the code enclosed within the LaTeX markup. See "Producing Published Output from M-Files" on page 10-68 for information on specifying the output file format.

To insert the text markup for LaTeX code, follow these steps:

1. Position the cursor at the end of the comment that precedes the location where you want to insert LaTeX code. For example, if the M-file contains the following lines, position the cursor after the colon:

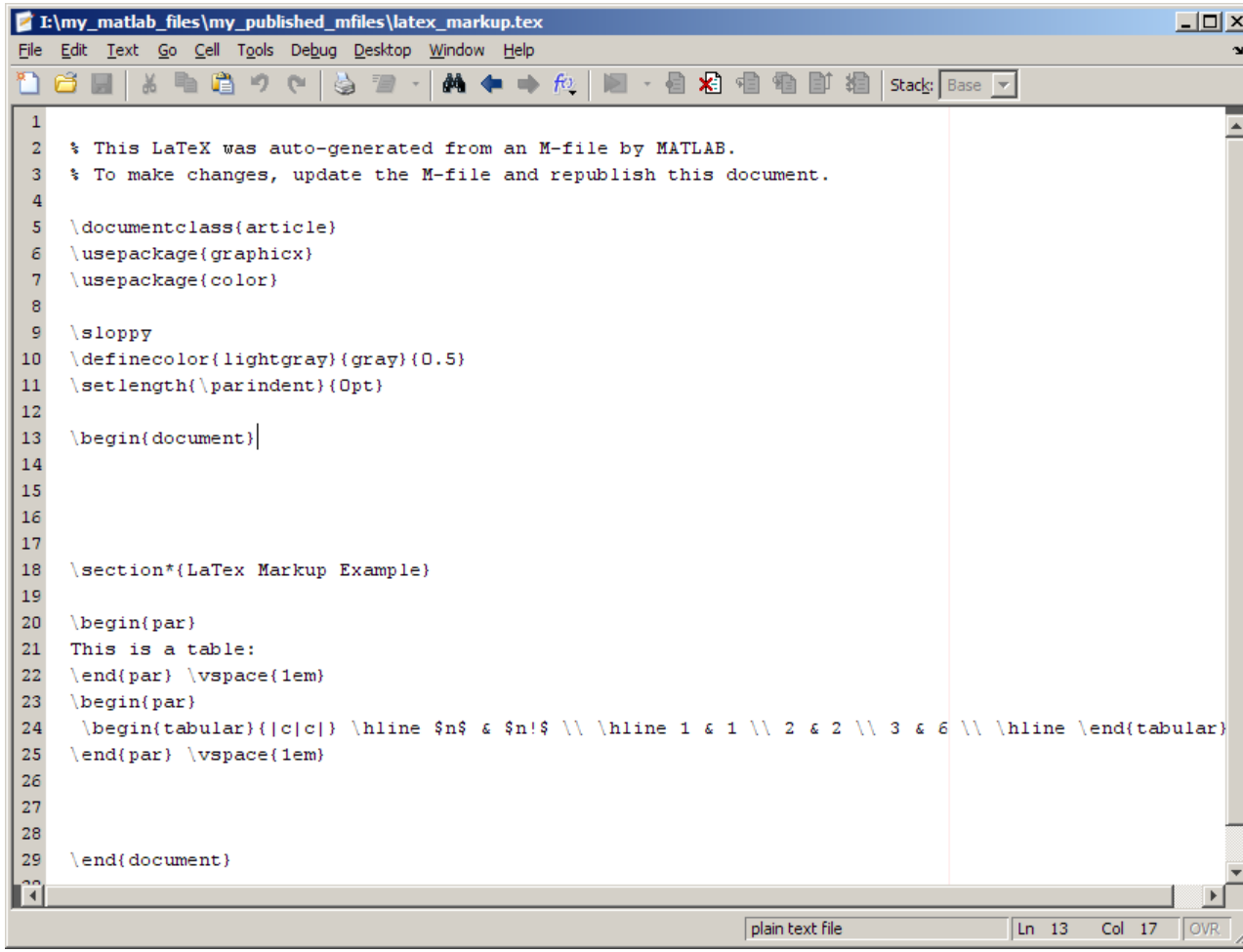
   ```
   %% LaTeX Markup Example
   % This is a table:
   ```

2. Select **Cell > Insert Text Markup > LaTeX Markup**. MATLAB adds LaTeX markup, as shown in the following figure.



3 Edit the inserted LaTeX code to specify the LaTeX code that you want to use.

If you publish the M-file to LaTeX, and leave the inserted markup text as is, the Editor opens a new file with the LaTeX code, as shown in the following figure. (See “Creating a Publish Configuration for an M-File” on page 10-70 for information on specifying LaTeX as the output format for a published document.)



If you compile the published LaTeX code, it appears as follows.

LaTex Markup Example

This is a table:

n	$n!$
1	1
2	2
3	6

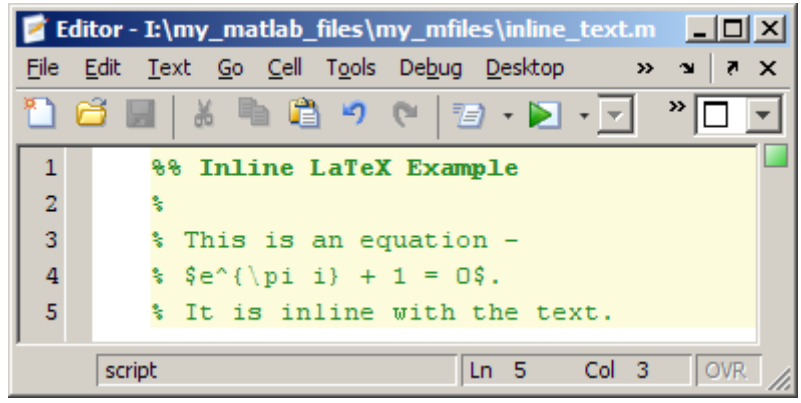
If the results are not as you expect, see “About Formatted Blocks” on page 10-54.

Including Inline LaTeX Math Symbols in M-Files for Publishing

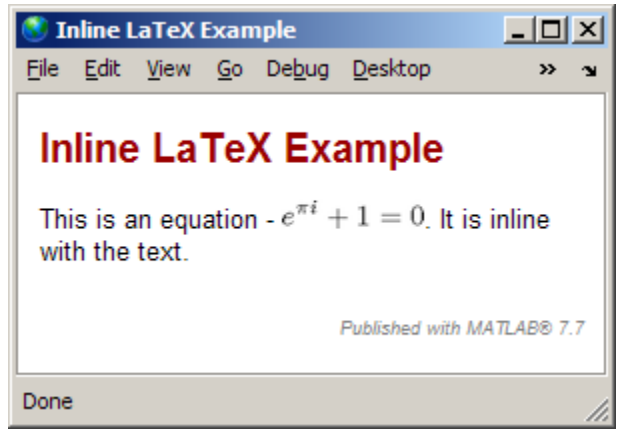
You can make LaTeX math symbols appear inline when you publish an M-file to any format, except Microsoft PowerPoint®. For Microsoft PowerPoint output, consider using an “Using LaTeX Markup for Publishing” on page 10-36 instead.

To make a LaTeX math symbol appear inline, enclose the string within dollar sign symbols ($) within a formatted comment block.

For example, suppose your M-file appears as follows:



When you publish the file to HTML, it appears as in the image that follows. Notice that the LaTeX math symbols are inline with the rest of the text:



For a list of symbols you can display, and the character sequence to create them, see the MATLAB Text `String` property.

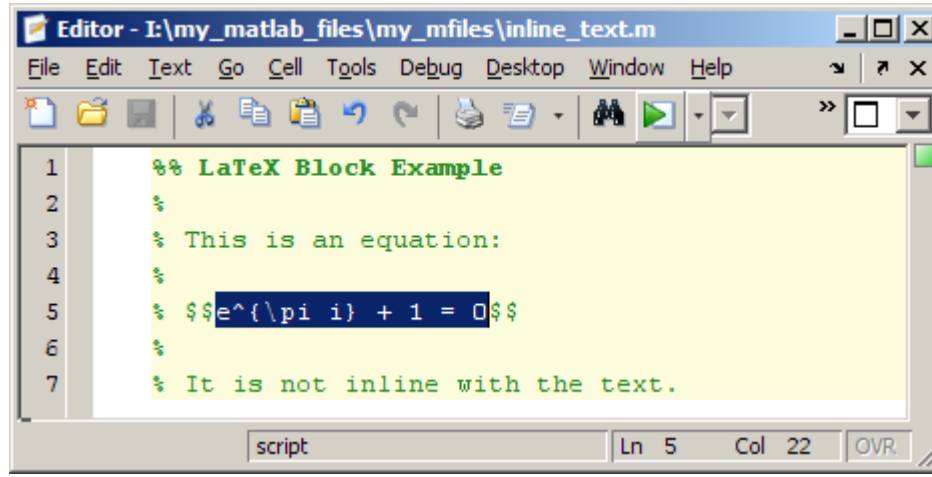
Including Blocks of LaTeX Math Symbols in M-Files for Publishing

You can use the **Cell** menu to insert LaTeX symbols in blocks offset from the main comment text. You can use the inserted code as a guideline for inserting other LaTeX code.

1. Position the cursor before the line where you want to add an equation or symbols. For example, if your M-file contains the following lines, position the cursor after the colon on the end of the third line:

```
%% LaTeX Block Example
%
% This is an equation:
% It is not inline with the text.
```

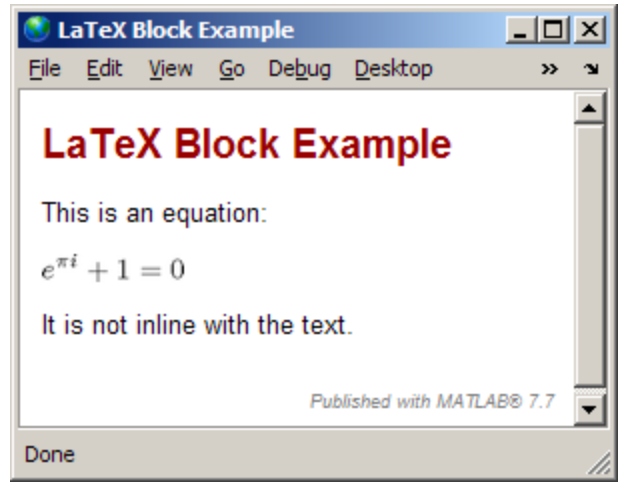
2. Select **Cell > Insert Text Markup > TeX Equation** to insert the sample equation markup.



3 Replace the inserted sample markup `e^{\pi i} + 1 = 0` with the LaTeX math symbols that you want.

For a list of symbols you can display, and the character sequence to create them, see the MATLAB Text `String` property.

If you publish the file to HTML, and leave the inserted sample text markup as is, the published document appears as shown in the following figure.



If the results are not as you expect, see "About Formatted Blocks" on page 10-54.

Forcing a Snapshot of Output in M-Files for Publishing

You can use the **Cell** menu to insert code that forces a snapshot of output, such as a figure. This is useful, for example, if you have a `for` loop that generates numerous figures and you want to include them all in the published output, after the `for` loop `end` statement.

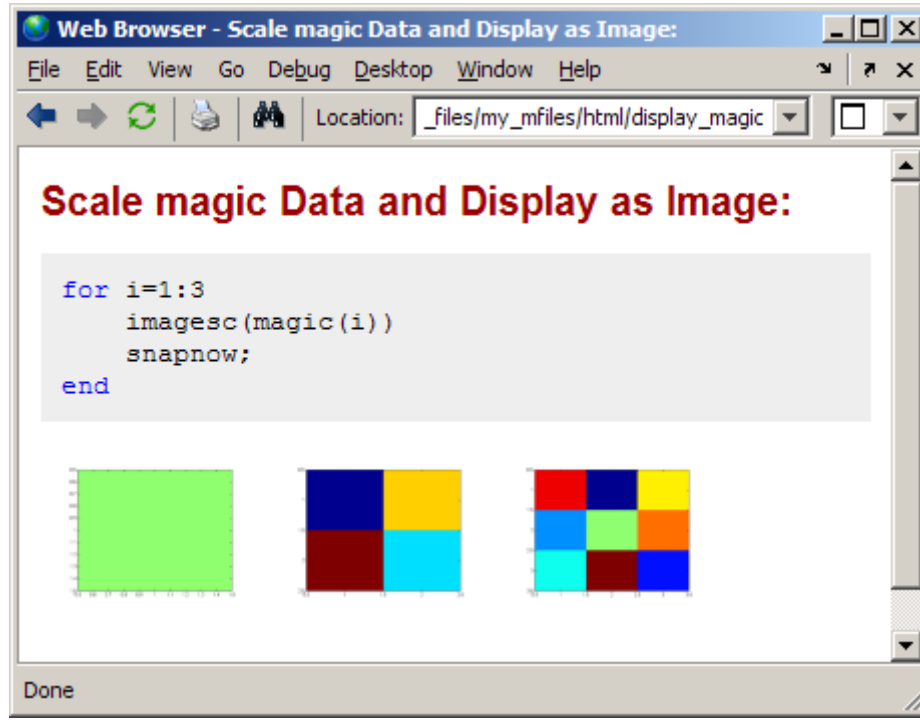
1. Position the cursor at the end of the line where you want to force a snapshot of the output. For example, if your M-file contains the following lines, position the cursor after the line containing the `imagesc` function:

```
%% Scale magic Data and Display as Image:

for i=1:3
    imagesc(magic(i))
end
```

2. Select **Cell > Insert Text Markup > Force Snapshot** to insert the `snapnow` function:

```
%% Scale magic Data and Display as Image:

for i=1:3
    imagesc(magic(i))
    snapnow;
end
```

3. If you publish the file to HTML, the published document resembles the following figure. The images in your published document will be larger than shown in the figure. To resize of images generated by M-file code, you use the **Max image width** and **Max image height Publish settings**, as described in "Producing Published Output from M-Files" on page 10-68.



Including Bold, Italic, and Monospaced Text Formats in M-Files for Publishing

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, as described in the following sections.

Marking Up Existing Comments with Font Formats

To mark up existing comments, follow these steps:

1. Within a comment, select text that you want to be bold, italic, or monospaced.

2. Select **Cell > Insert Text Markup**, and then select **Bold Text**, **Italic Text**, or **Monospaced Text**.

Inserting New Comments with Font Formats

To insert sample text that you will replace with your new comment text, follow these steps:

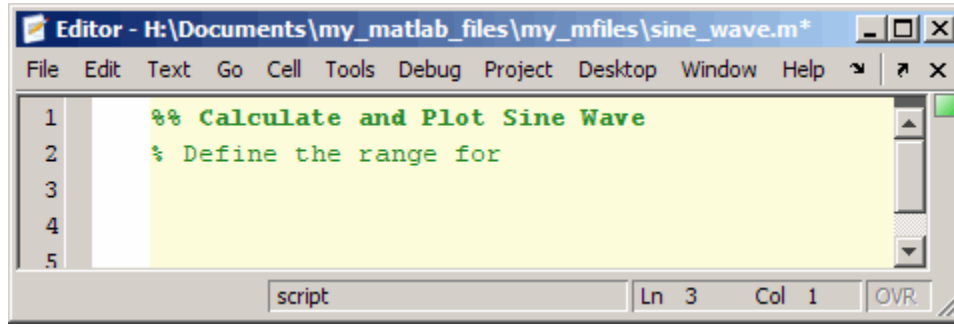
1. Select **Cell > Insert Text Markup**, and then select **Bold Text**, **Italic Text**, or **Monospaced Text**.
2. Replace the inserted text with the text that you want formatted.

When the Editor inserts sample text, the inserted text appears as follows:

```
% *BOLD TEXT*
% _ITALIC TEXT_
% |MONOSPACED TEXT|
```

Example of Font Formats

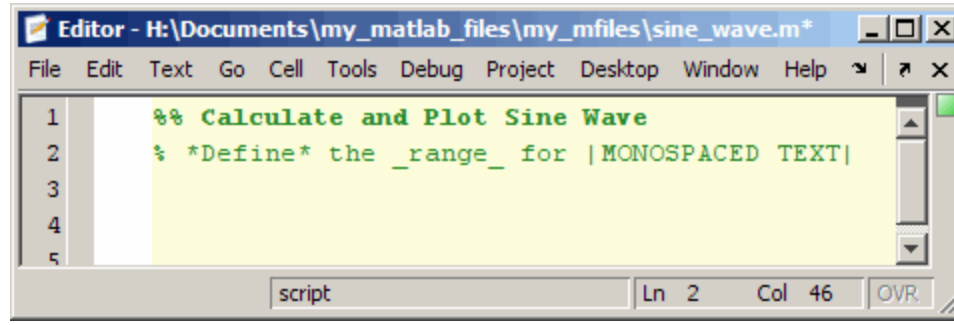
Suppose your M-file appears as follows.



Format the comments as follows:

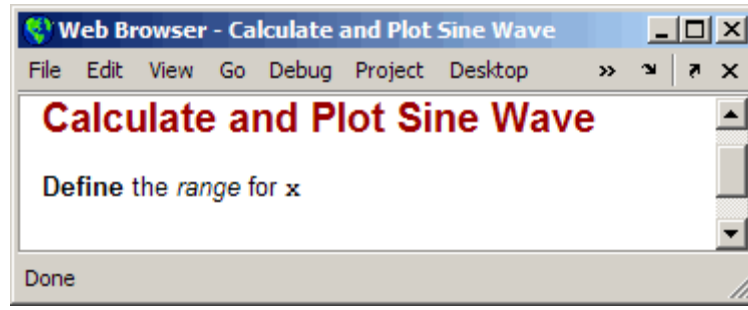
1. Select the word `Define`, and then select **Cell > Insert Text Markup > Bold Text**.
2. Select the word `range`, and then select **Cell > Insert Text Markup > Italic Text**.
3. Position the cursor after the word `for`, insert a space, and then select **Cell > Insert Text Markup > Monospaced Text**.

The M-file appears as follows.



4 Replace `|MONOSPACED TEXT|` with `|x|`.

If you publish the M-file to HTML, the output appears as shown in the following figure.

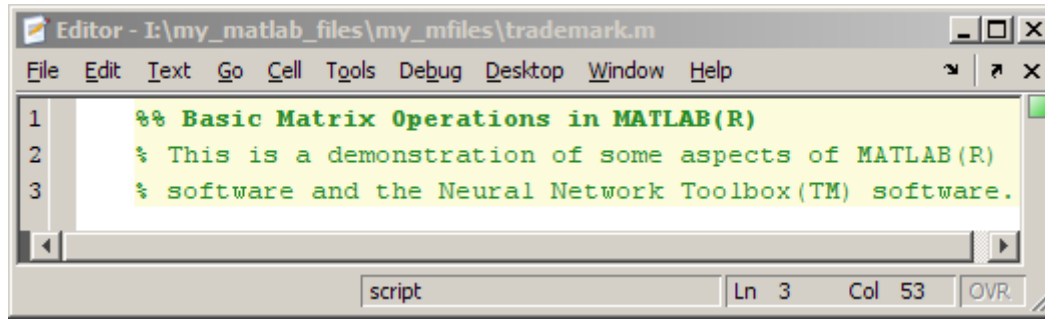


Including Trademarks in M-Files for Publishing

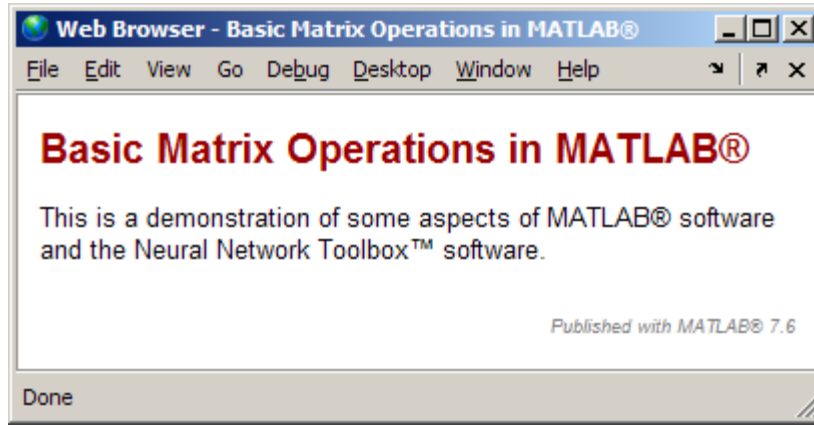
If the comments in your M-file include trademarked terms, you can include text to produce a trademark symbol (™) or registered trademark symbol (®) in the published output.

- To produce the trademark symbol, enter `(TM)` in an M-file comment.
- To produce the registered trademark symbol, enter `(R)` in an M-file comment.

For example, suppose you enter lines in an M-file as shown in the following image.



If you publish the M-file to HTML, it appears as follows in the MATLAB Web Browser.



Including Links in M-Files for Publishing

You can insert dynamic or static links within an M-file comment, and then publish the M-file to HTML, or XML. You can also publish static hyperlinked text to Microsoft Word.

When you specify a *dynamic* link, MATLAB resolves the URL when the user clicks the link in the published document. You do not specify the exact URL in the code. This is useful when, for example, you want to point the reader to

some MATLAB code or documentation. When you specify a *static* link, you specify the complete URL within the M-file code. This is useful when you want to point the user to a location on the Web. For both static and dynamic links, the publish document contains active links.

You can include or exclude the URL from a published static link. You might, for example, include the URL when you anticipate that readers of your published document might view it in printed format, and therefore need the URL. You might exclude the URL, when you are confident that readers will view your published document online and therefore be able to click the link.

This section includes the following topics:

Inserting Static Links and Publishing URLs

You can insert a URL that you know at the time the M-file is written, such as `www.mathworks.com`, by following these steps:

1. Within a comment, position the cursor where you want to insert the hyperlinked text. For example, suppose you want to specify a link to more information about a topic. You might have the following comment within the M-file:

   ```
   %%
   % For more information, see our Web site:
   ```

   Position your cursor after the colon (`:`).

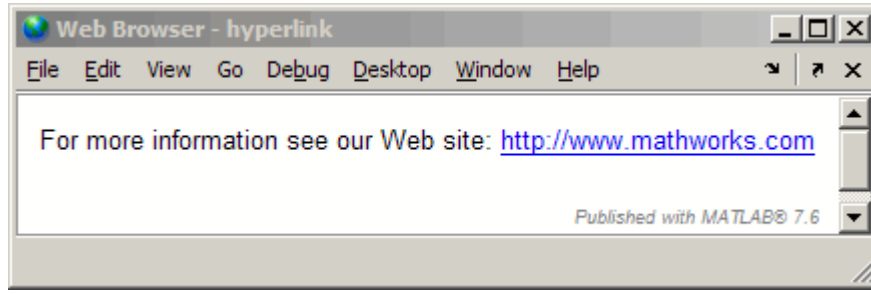
2. Select **Cell > Insert Text Markup > Hyperlinked Text**. The Editor inserts the following:

   ```
   <http://www.mathworks.com The MathWorks>
   ```

3 Replace `www.mathworks.com` with the URL you want to use.

4 Delete the string, `The MathWorks`.

When you publish the M-file to HTML, the results resemble the following figure (except the URL in this image is still `http://www.mathworks.com`).



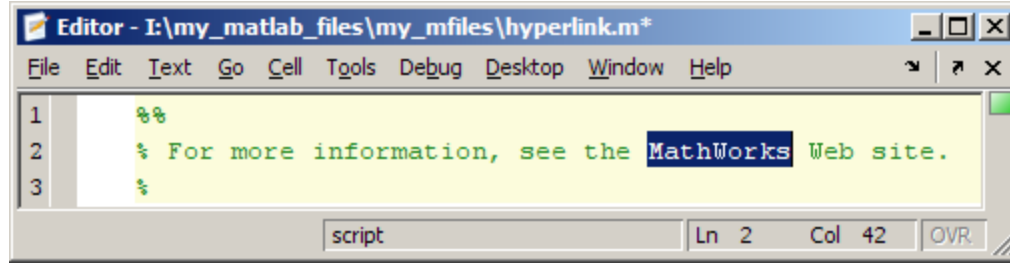
Inserting Static Links Without Publishing URLs

To insert hyperlinked text without a printed URL, follow these steps:

1 Within a comment, position the cursor where you want to insert the hyperlinked text. For example, suppose you want to specify a link to the MathWorks Web site. You might have the following lines within your M-file:

```
%%
% For more information, see the MathWorks Web site.
```

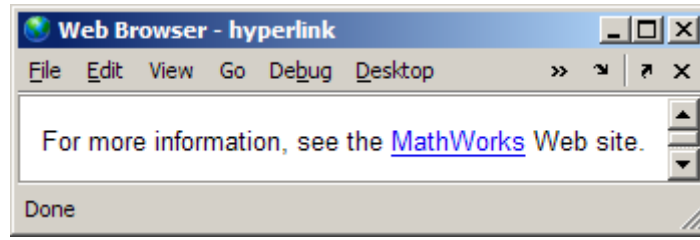
Select the text you want to replace with a link. For example, select “MathWorks," as shown in the following figure.



2 Select **Cell > Insert Text Markup > Hyperlinked Text**. The Editor replaces the selected text with the following:

```
<http://www.mathworks.com MathWorks>
```

If you publish the M-file to HTML, the results are as shown in the following figure.



3 Replace `www.mathworks.com` with the URL that you want to use.

4 Replace `MathWorks` with the text that you want to appear as the hyperlinked text.

Inserting Dynamic Links

You can insert a link which MATLAB evaluates at the time a reader clicks that link in the published document. You implement these links as `matlab:` links. Unlike other uses of `matlab:`, you do not need to use it within another function, such as `disp`.

You cannot insert dynamic links in Microsoft Word files.

Note A reader of your published document must have MATLAB running for dynamic links to work.

For more information, see `matlabcolon`. The following sections provide two examples:

- "Inserting a Dynamic Link to a File" on page 10-50

- "Inserting a Dynamic Link to a MATLAB Function Reference Page" on page 10-51

Inserting a Dynamic Link to a File. You can specify a link to a file that you know is in your readers' `matlabroot`. You do not need to know where each reader installed MATLAB. To insert a dynamic link to a file, follow these steps:

1. Within a comment, position the cursor where you want to insert the link. For example, suppose you want to specify a link to the MATLAB help topic for the `publish` function. Also suppose you have the following comment within the M-file:

   ```
   %%
   % See the code
   % for the publish function.
   ```

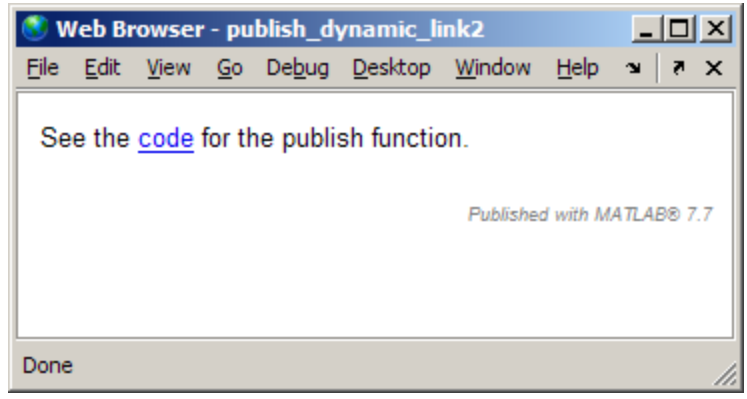
2. Replace the word `code` with the following markup:

   ```
   <matlab:web(fullfile(matlabroot,'toolbox','matlab','codetools','publish.m')) code>
   ```

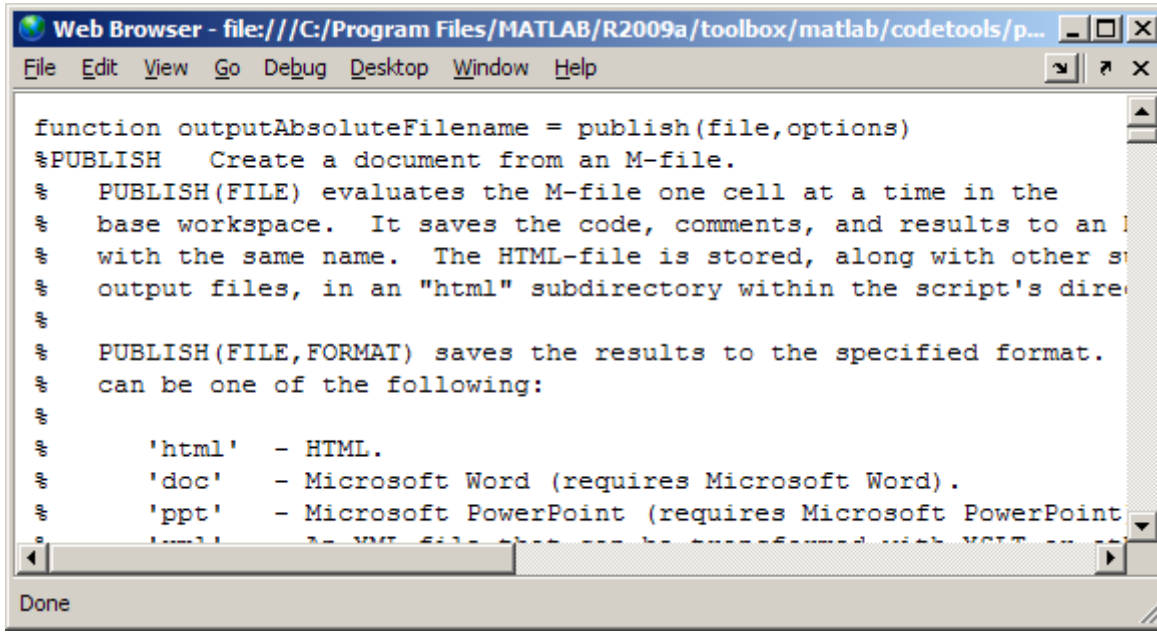
   The resulting M-file code now appears as follows:

   ```
   %%
   % See the <matlab:web(fullfile(matlabroot,'toolbox','matlab','codetools','publish.m')) code>
   % for the publish function.
   ```

When you publish the file to HTML, the results resemble the following figure.



When the reader of the published document clicks the `code` link, the MATLAB Web browser opens and displays the code for the publish function. On the reader's system, MATLAB issues the command (although the command does not appear in the reader's Command window). Notice in the Web browser title bar that the `matlabroot` specification in the M-File resolves to the reader's installation folder for MATLAB.



Inserting a Dynamic Link to a MATLAB Function Reference Page. You can specify a link to a MATLAB function reference page. For example, suppose you know the readers of your published document have MATLAB installed and running. To provide a link to the `publish` reference page, follow these steps:

1. Within a comment, position the cursor where you want to insert the hyperlinked text. For example, suppose you want to specify a link to the MATLAB help topic for the `publish` function. Furthermore, suppose you have the following comment within the M-file:

   ```
   %%
   ```

```
% See the help for the publish function.
```

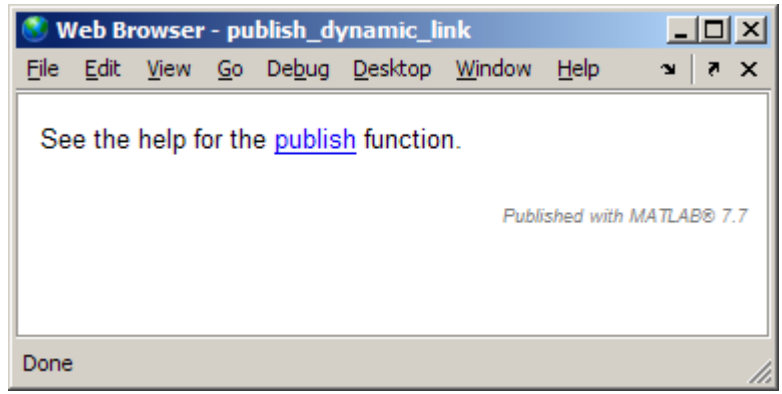
2 Replace the word `publish` with the following markup:

```
<matlab:doc('publish') publish>
```

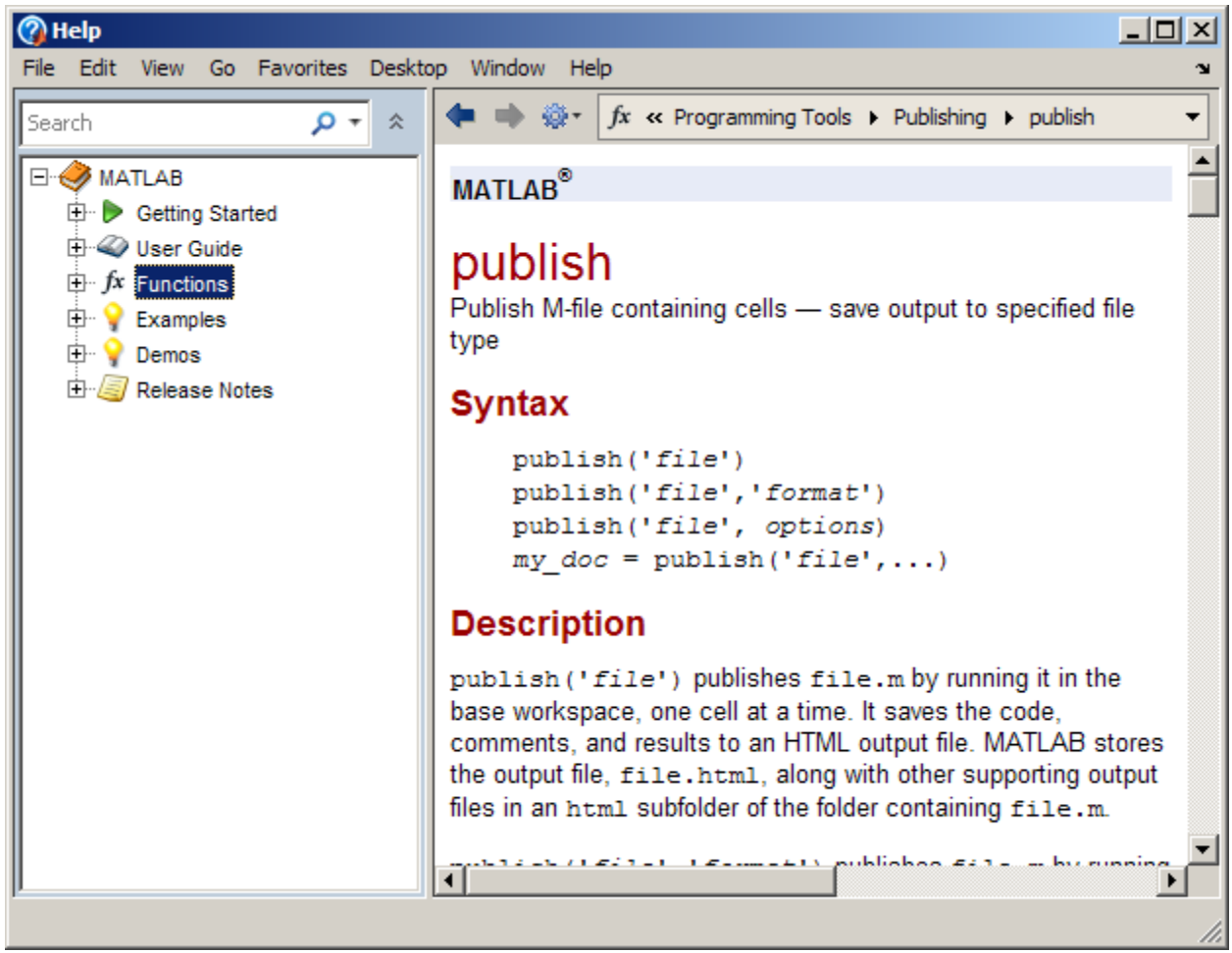
The resulting M-file code now appears as follows:

```
%%
% See the help for the <matlab:doc('publish') publish> function.
```

When you publish the file to HTML, the results resemble the following figure.



When the reader of the published document clicks the `publish` link, the MATLAB help browser opens and displays the reference page for the publish function. On the reader's system, MATLAB issues the command (although the command does not appear in their Command window).



Effect of Using Hyperlinked Text from the MATLAB Command Window

You cannot use statements that display hyperlinked text in the MATLAB Command Window to create hyperlinked text in a published M-file. If you try, the published document shows the code rather than the hyperlink.

For example, suppose you enter the following code in the Command Window:

```
disp('<a href="http://www.mathworks.com">Link to MathWorks</a>')
```

When you press **Return**, the Command Window displays a link to the MathWorks Web site:

Link to MathWorks

However, if you include the preceding `disp` statement in an M-file that you publish, the HTML tag and the included text appear in the published document, rather than a link:

```
disp('<a href="http://www.mathworks.com">Link to MathWorks</a>')
```

Instead, use one of the methods described in these sections:

- “Inserting Static Links and Publishing URLs” on page 10-47
- “Inserting Static Links Without Publishing URLs” on page 10-48
- “Inserting Dynamic Links” on page 10-49

About Formatted Blocks

Multiple contiguous lines of comments immediately following a cell break are referred to as “descriptive” or “introductory text”. Within these lines of comments, empty lines create formatted blocks. Formatted blocks control how comments appear within the final published document.

Specify each of the following items as a formatted block to achieve the intended results in the published document:

- Preformatted text
- Bulleted and numbered lists
- Graphics
- HTML markup
- LaTeX markup
- TeX equations

The following sections provide general information on formatted blocks:

Understanding How Formatted Blocks are Demarcated

A formatted block starts on the first comment line *after* one of the following:

- A cell break
- A blank comment line (a percent sign with no other characters on the line)

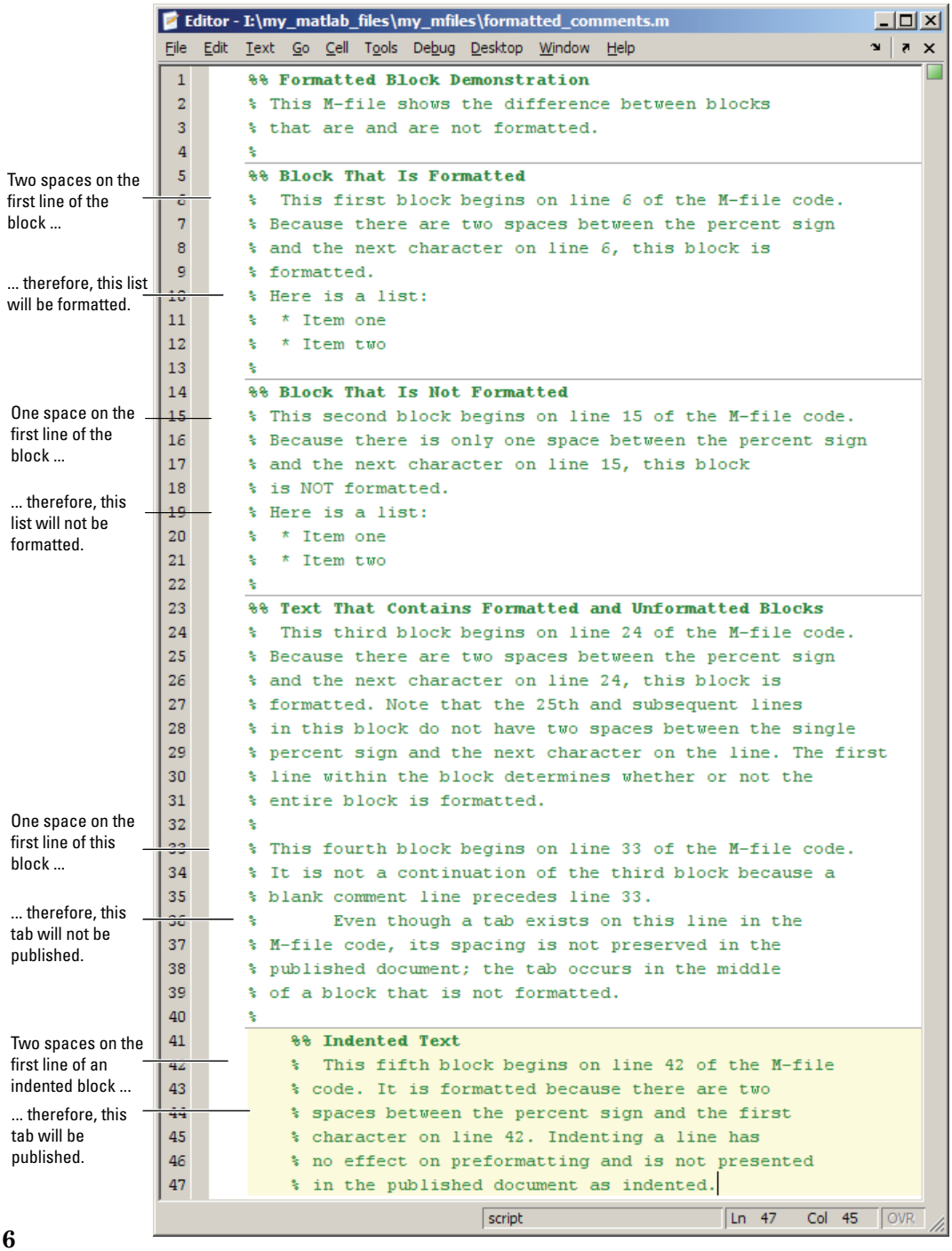
A formatted block ends on the last comment line *before* one of the following:

- A cell break
- A blank comment line
- Any line of uncommented M-file code

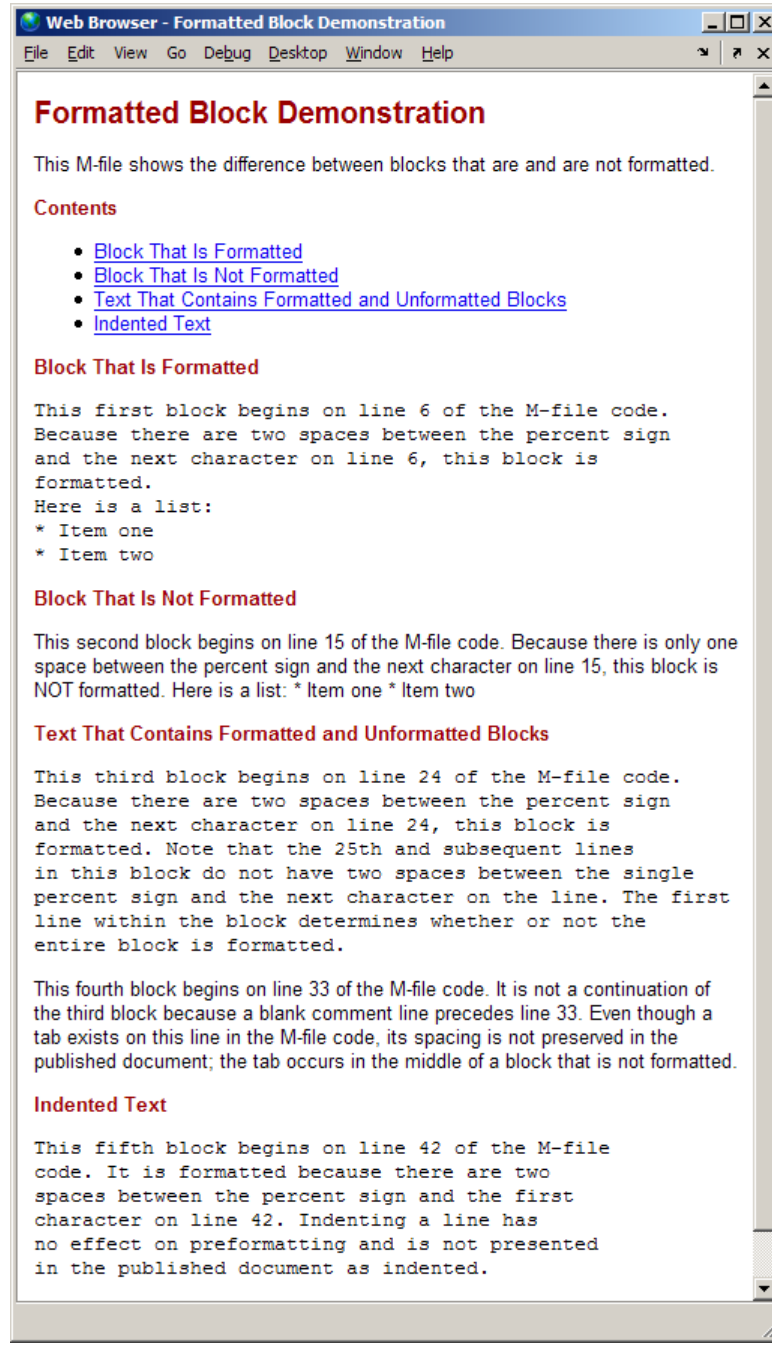
Understanding How Formatted Blocks Work

A comment is part of a formatted block if, on the first line of the block there are two spaces between the single percent sign (%) and the next character. Any number of white spaces can precede the percent sign. That is, the percent sign can be indented. The following two images demonstrate the difference between blocks that are and are not formatted:

- The first image show the M-file source code.
- The second image shows the M-file published to an HTML document.




```
1  %% Formatted Block Demonstration
2  % This M-file shows the difference between blocks
3  % that are and are not formatted.
4  %
5  %% Block That Is Formatted
6  %  This first block begins on line 6 of the M-file code.
7  % Because there are two spaces between the percent sign
8  % and the next character on line 6, this block is
9  % formatted.
10 % Here is a list:
11 %  * Item one
12 %  * Item two
13 %
14 %% Block That Is Not Formatted
15 % This second block begins on line 15 of the M-file code.
16 % Because there is only one space between the percent sign
17 % and the next character on line 15, this block
18 % is NOT formatted.
19 % Here is a list:
20 %  * Item one
21 %  * Item two
22 %
23 %% Text That Contains Formatted and Unformatted Blocks
24 %  This third block begins on line 24 of the M-file code.
25 % Because there are two spaces between the percent sign
26 % and the next character on line 24, this block is
27 % formatted. Note that the 25th and subsequent lines
28 % in this block do not have two spaces between the single
29 % percent sign and the next character on the line. The first
30 % line within the block determines whether or not the
31 % entire block is formatted.
32 %
33 % This fourth block begins on line 33 of the M-file code.
34 % It is not a continuation of the third block because a
35 % blank comment line precedes line 33.
36 %       Even though a tab exists on this line in the
37 % M-file code, its spacing is not preserved in the
38 % published document; the tab occurs in the middle
39 % of a block that is not formatted.
40 %
41     %% Indented Text
42     %  This fifth block begins on line 42 of the M-file
43     % code. It is formatted because there are two
44     % spaces between the percent sign and the first
45     % character on line 42. Indenting a line has
46     % no effect on preformatting and is not presented
47     % in the published document as indented.
```


Web Browser - Formatted Block Demonstration
File Edit View Go Debug Desktop Window Help
Formatted Block Demonstration
This M-file shows the difference between blocks that are and are not formatted.
Contents
Block That Is Formatted
Block That Is Not Formatted
Text That Contains Formatted and Unformatted Blocks
Indented Text
Block That Is Formatted
This first block begins on line 6 of the M-file code.
Because there are two spaces between the percent sign
and the next character on line 6, this block is
formatted.
Here is a list:
* Item one
* Item two
Block That Is Not Formatted
This second block begins on line 15 of the M-file code. Because there is only one space between the percent sign and the next character on line 15, this block is NOT formatted. Here is a list: * Item one * Item two
Text That Contains Formatted and Unformatted Blocks
This third block begins on line 24 of the M-file code.
Because there are two spaces between the percent sign
and the next character on line 24, this block is
formatted. Note that the 25th and subsequent lines
in this block do not have two spaces between the single
percent sign and the next character on the line. The first
line within the block determines whether or not the
entire block is formatted.
This fourth block begins on line 33 of the M-file code. It is not a continuation of the third block because a blank comment line precedes line 33. Even though a tab exists on this line in the M-file code, its spacing is not preserved in the published document; the tab occurs in the middle of a block that is not formatted.
Indented Text
This fifth block begins on line 42 of the M-file
code. It is formatted because there are two
spaces between the percent sign and the first
character on line 42. Indenting a line has
no effect on preformatting and is not presented
in the published document as indented.

If you want, you can experiment with the code presented in the previous images. Open `formatted_block_demo.m` by running the following command:

```
edit(fullfile(matlabroot,'help','techdoc',...
'matlab_env','examples','formatted_block_demo.m'))
```

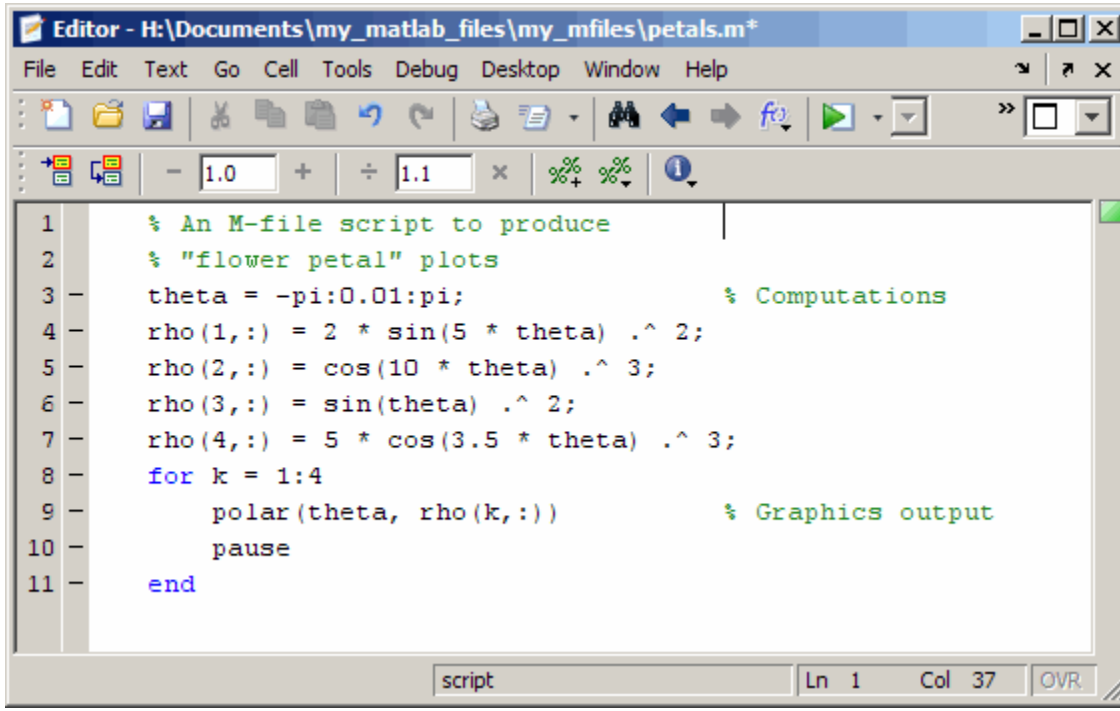
To work with `formatted_block_demo.m` on your system, save the file to a folder for which you have write permission. In the example, the file is saved to `I:\my_matlab_files\my_mfiles\formatted_comments.m`.

Cleaning Up Text Markup Before Publishing M-Files

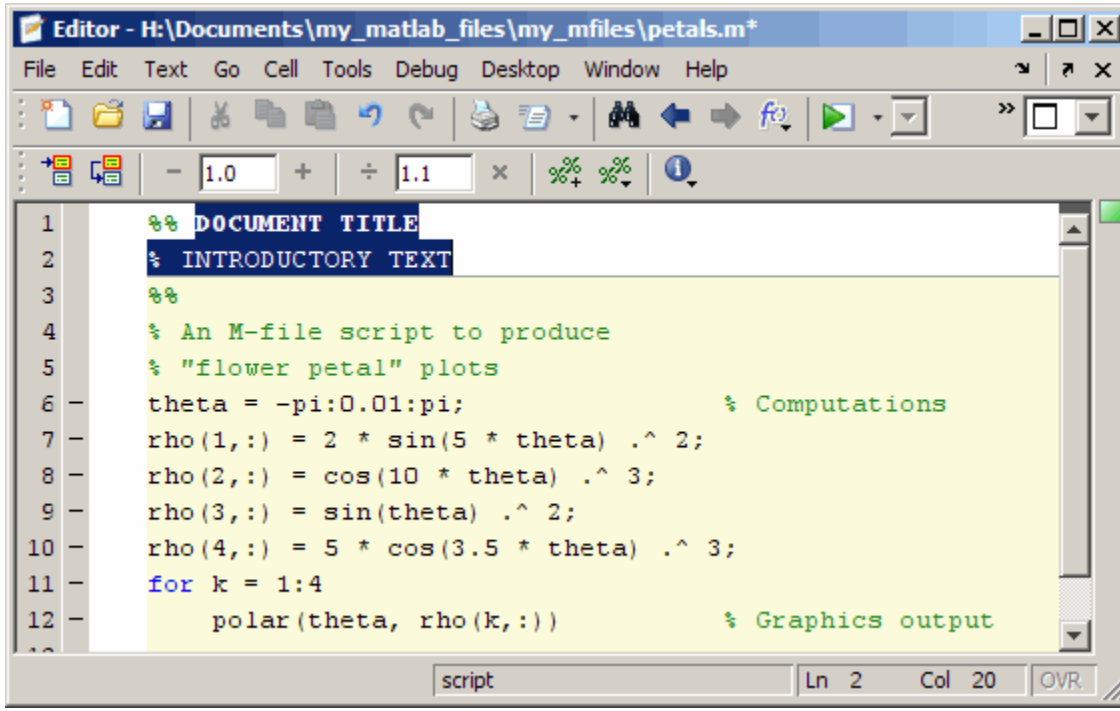
When you insert text markup into an existing M-file using the **Cell** menu options, you might find that more comment lines than you need are inserted. You can adjust the inserted comments as needed for your purposes. If you delete blank comment lines that the **Cell** menu options insert there may be unintended consequences, however. See "Specifying Preformatted Text in M-Files for Publishing" on page 10-26 for details.

The following example shows how you might use **Cell** menu options with an existing M-file.

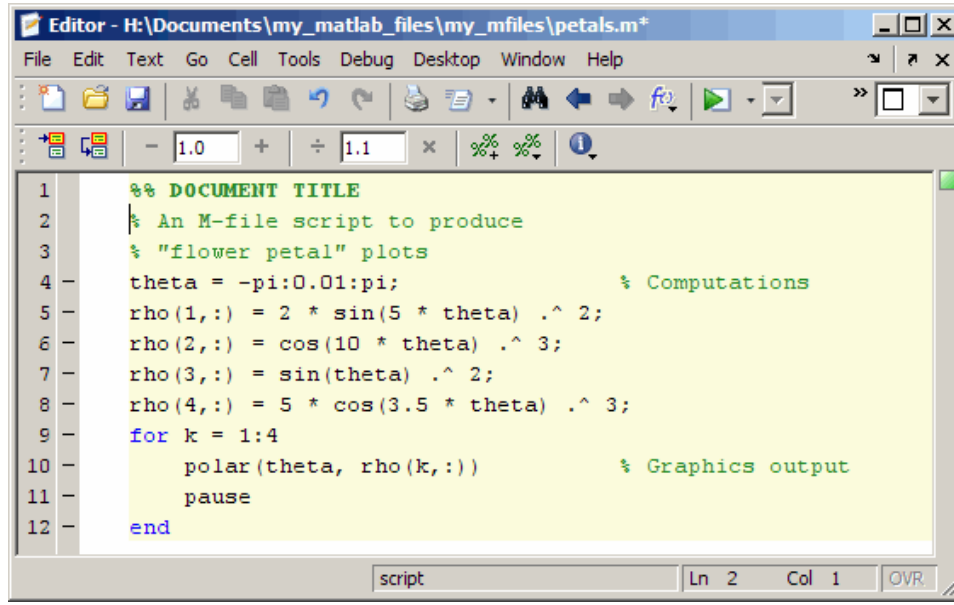
Suppose an M-file currently appears in the Editor as shown in the following image.



If you position the cursor anywhere within the file and select **Cell > Insert Text Markup > Document Title and Introduction**, the M-file looks like the following.



The file already contains a comment with introductory text, so you can delete the `% INTRODUCTORY TEXT` line and the double percent sign (`%%`) line, so the code appears as follows.



Summary of Markup for Formatting M-Files for Publishing

The following two tables provides a summary of the text markup that you can type into an M-file to achieve the same results as using the **Cell > Insert Text Markup** menu options. These tables are particularly useful if you are not using the MATLAB Editor, or if you do not want to use menus to apply formatting. For a description of the **Cell** menu options, see "Formatting M-File Comments for Publishing" on page 10-18.

Summary of Markup Not Requiring a Formatted Block

This table summarizes markup that does not require that it be included within a formatted block.

Result in Published Document	Example of Corresponding M-File Markup
Document title and introduction	`%% DOCUMENT TITLE` `% INTRODUCTORY TEXT`
Section title and description	`%% SECTION TITLE` `% DESCRIPTIVE TEXT`
Section title without cell break	`%%% SECTION TITLE` `% DESCRIPTIVE TEXT`
Cell break without title or description	`%%`
Bold text	`% *BOLD TEXT*`
Italic text	`% _ITALIC TEXT_`
Monospaced text	`% |MONOSPACED TEXT|`
Hyperlinked text	`% <http://www.mathworks.com MathWorks>`
Trademark symbol	`% TEXT(TM)`
Registered trademark symbol	`% TEXT(R)`
Code to force a snapshot of the current output	`snapnow;`

Summary of Markup Requiring a Formatted Block

This table summarizes markup that requires that it be included within a formatted block. See “About Formatted Blocks” on page 10-54 for details.

Result in Published Document	Example of Corresponding M-File Markup
Image	`% <<FILENAME.PNG>> %`
Bulleted list	`% * ITEM1` `% * ITEM2`
Numbered list	`% # ITEM1` `% # ITEM2`

HTML markup	`% <html>`
	`% <table border=1><tr>`
	`% <td>one</td>`
	`% <td>two</td></tr></table>`
	`% </html>`

LaTeX markup	`% <latex>`
	`% \begin{tabular}{|r|r}`
	`% \hline $n$&$n!$\\`
	`% \\hline 1&1\\ 2&2\\ 3&6\\`
	`% \\hline`
	`% \end{tabular}`
	`% </latex>`

TeX equation	`% $$e^{\pi i} + 1 = 0$$`

Formatting M-File Code for Publishing

In this section...

Overview of Formatting M-File Code for Publishing

This section describes ways to control how output that the MATLAB software generates when it evaluates executable M-file code for a published document. For example, you can direct MATLAB to include the last, or all plots generated by a `for` loop. You can interweave comments, code, and output throughout your published document to draw your readers' attention to certain areas of interest.

Formatting Code Output in a Published M-File

The tool you use to specify how output is presented in the document is the same tool you use to specify document titles and section headers; namely the double percent sign (%%) which is referred to as a cell break. When you insert a cell break into a file, it directs MATLAB to publish the code and output contained in the cells created by the break. Because MATLAB considers the entire M-file to be a cell, when you insert a cell break, MATLAB considers the file to contain two cells; one above the cell break and one below. The examples in the remaining topics demonstrate how you can use this behavior to control the output produced by M-file code.

Example of Formatting Code Output in a Published M-File

This section provides an example to demonstrate how an M-file appears when published. It demonstrates how the published example file appears before and after cell breaks are added to achieve the published results.

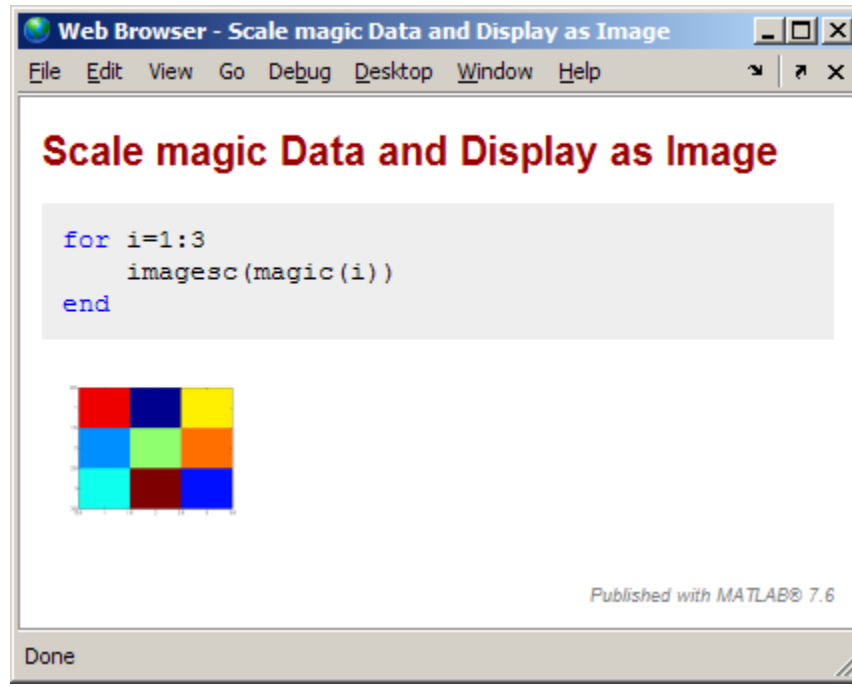
Sample M-File Before Inserting Cell Breaks in Code

Suppose your M-file contains the following code:

```
%% Scale magic Data and Display as Image

for i=1:3
    imagesc(magic(i))
end
```

The following image illustrates how the code presented appears when you publish it to HTML. The plot in the figure is smaller than it appears if you publish the M-code using factory default settings. For information on setting publishing properties for images, see "Producing Published Output from M-Files" on page 10-68.

Notice that the published document displays the plot after the end of the `for` loop and that only the last plot generated by the code is included.



Sample M-File After Inserting Cell Breaks in Code

By placing cell breaks within a loop, you can display the output generated by M-file code when iterating a loop.

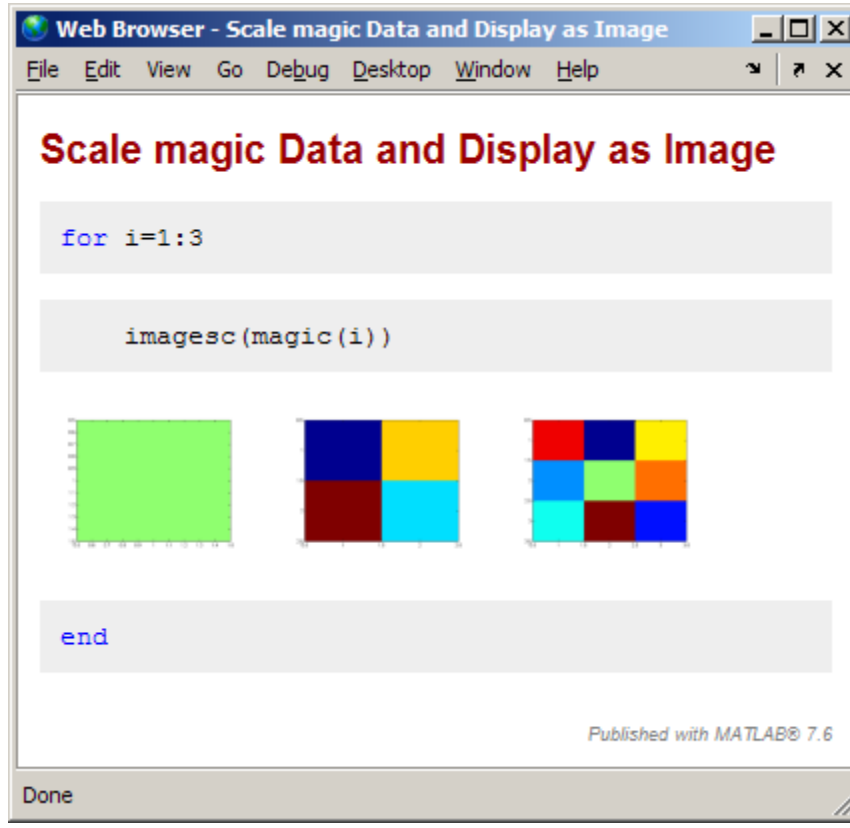
To include the plot generated by each iteration of the loop in the published document, insert a cell break after the opening for statement. Position the cursor at the end of the first line of the `for` loop, and then select **Cell > Insert Cell Break**.

The code now appears like this:

```
%% Scale magic Data and Display as Image

for i=1:3
    %%
    imagesc(magic(i))
end
```

Now when you publish the code to HTML, it appears as follows. The plots in the figure are smaller than they appear if you publish the M-code using factory default settings. For information on setting publishing properties for images, see “Producing Published Output from M-Files” on page 10-68.

Notice that the published document displays the plot within the `for` loop code. You can also use text markup for similar results with figures. See “Formatting M-File Code for Publishing” on page 10-64 for details.


Web Browser - Scale magic Data and Display as Image
File Edit View Go Debug Desktop Window Help
Scale magic Data and Display as Image
for i=1:3
imagesc(magic(i))
end
Published with MATLAB® 7.6
Done

Producing Published Output from M-Files

In this section...

About Producing Published Output

Once you have formatted an M-file for publishing, as described in "Formatting M-File Comments for Publishing" on page 10-18 and "Formatting M-File Code for Publishing" on page 10-64 you are ready to publish it. The easiest method for publishing an M-file is to use factory default publishing properties. This method is appropriate if your M-file requires no input arguments and you want to publish to HTML. However, if your M-file requires input arguments, or you want to specify preferences for publishing, such as the output folder, output format, image format, and so on, you need to specify custom property settings.

Publishing M-Files Using No Input Arguments and Factory Default Settings

To publish a script M-file, or a function M-file that requires no input arguments:

1. Open the file in the Editor.
2. Click the Publish button  on the Editor toolbar.

By default, the Editor publishes the file using factory default settings. Factory default settings specify that the output file format is HTML, that the M-code is evaluated and included in the published output file, and so on.

If the file is neither in a folder on the search path, nor in the current folder, a dialog box opens and presents you with options that allow you to publish the file. You can either change the current folder to the folder containing the file, or you can add the folder containing the file to the MATLAB search path.

If the file has unsaved changes, publishing it from the Editor automatically saves the changes before publishing.

Using Publish Configurations to Publish M-Files with Input Arguments or Custom Settings

Using a publish configuration, you can specify custom settings, including input arguments for a function M-file in the Editor. You can associate multiple publish configurations with an M-file for different publish settings, input arguments, or both. MATLAB saves the configuration 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 publish `collatplot_new.m` because the input value is not defined. A publish configuration enables you to publish `collatzplot_new(`*`specific value`*`).`

You can also use publish configurations to provide preparatory or setup information prior to publishing an M-file, whether it takes input arguments or not.

Note M-file publish configurations use the base MATLAB workspace. Therefore, a value that you assign to a variable in an M-file publish configuration overwrites the value for that variable (assuming it currently exists) in the base workspace.

Function Alternative to Publishing

From the Command Window, execute the `publish` function to run the M-file and publish the results. See the `publish` function reference page for options you can set.

Creating a Publish Configuration for an M-File

Specifying M-File Input Using a Publish Configuration

Follow these steps to create a publish configuration for an M-file in the Editor. The example in this section shows how to create and use a publish configuration to specify input arguments to a function M-file.

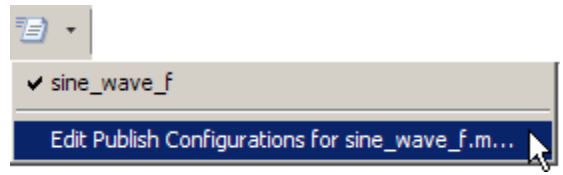
These steps specify Editor toolbar buttons, but you can also use equivalent items in the **File** menu.

1. Open the file that you want to publish in the Editor. This example uses the code that follows. This code is similar to the `sine_wave.m` file, after it has been formatted as described in "Formatting M-File Comments for Publishing" on page 10-18, but it is slightly altered to make it a function M-file. Save the code as `sine_wave_f.m`

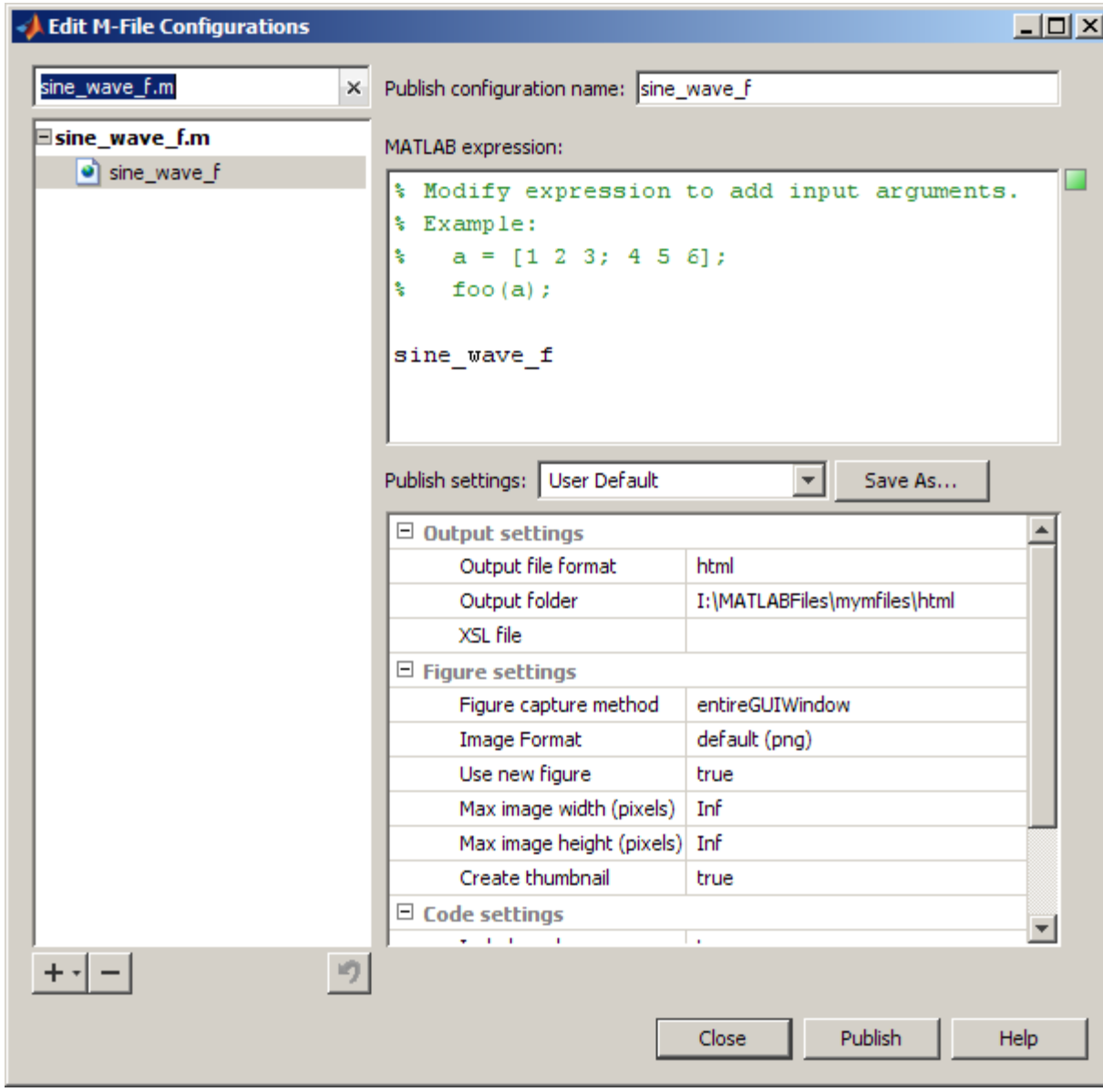
```
%% Plot Sine Wave
% Calculate and plot a sine wave.

%% Calculate and Plot Sine Wave
% Calculate and plot |y = sin(x)|.

function sine_wave_f(x)

y = sin(x);
plot(x,y)
```

```
%% Modify Plot Properties

title('Sine Wave', 'FontWeight','bold')
xlabel('x')
ylabel('sin(x)')
set(gca, 'Color', 'w')
set(gcf, 'MenuBar', 'none')
```

2 Click the down arrow next to the **Publish** button on the Editor toolbar, and click **Edit Publish Configuration for *`file name`***, where file name in this example is `sine_wave_f.m`.



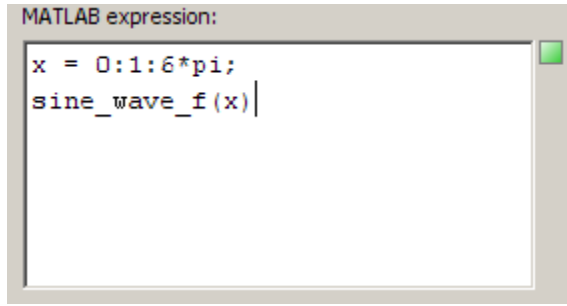
The Edit M-File Configurations dialog box opens, with the default publish configuration template for `sine_wave_f.m`, as shown in the following figure.



3 In the **Publish configuration name** field, type a name for the publish configuration, or accept the default name.

If you expect to create multiple configurations for an M-file, assign each a name that helps you identify the configuration. In this figure, the default name of the configuration is `sine_wave_f.`

4 In the **MATLAB expression** field, type the expression that you want the Editor to evaluate when it publishes the M-file. In this example, delete the commented statements and replace them as shown in the following figure.



You can modify the statements in the **MATLAB expression** area of the dialog box, and then click **Publish** to see the results of the changes. If you clear the **MATLAB expression** area, MATLAB publishes the M-file without evaluating any code. This is equivalent to setting the **Evaluate code** property in the **Publish settings** properties table to `false`.

5 In the **Publish settings** properties table, change the property values if you do not want to use the current settings.

You can modify the property settings, and then click **Publish** to see the results of the changes.

See "Specifying Values for the Publish Settings Property Table" on page 10-78 for details.

6 Do one of the following:

- To publish the file using the settings and MATLAB expression that you have specified, click **Publish**.

  For this example MATLAB creates the following files in `I:\my_matlab_files\my_mfiles\`html, which is a subfolder in the folder where `sine_wave_f.m` is located:

  - A published document file, `sine_wave_f.html`
  - A thumbnail file for the last image generated by the M-file code, `sine_wave_f.png`
  - Image files created by the executable M-file code, `sine_wave_f_##.png`

- To create another publish configuration for the same M-file, click the plus button  , and then select **Publish Configuration**.

  See "Creating Multiple Publish Configurations for an M-File" on page 10-97 for details.

- To close the Edit M-File Configurations dialog box, click **Close**. MATLAB saves the configuration and its association with the M-file.

After creating a configuration, you can view the MATLAB expression and use the configuration to publish the M-file without opening the Edit M-File Configurations dialog box. See "Running an Existing Publish Configuration" on page 10-96 for details.

Specifying Publish Configuration Settings

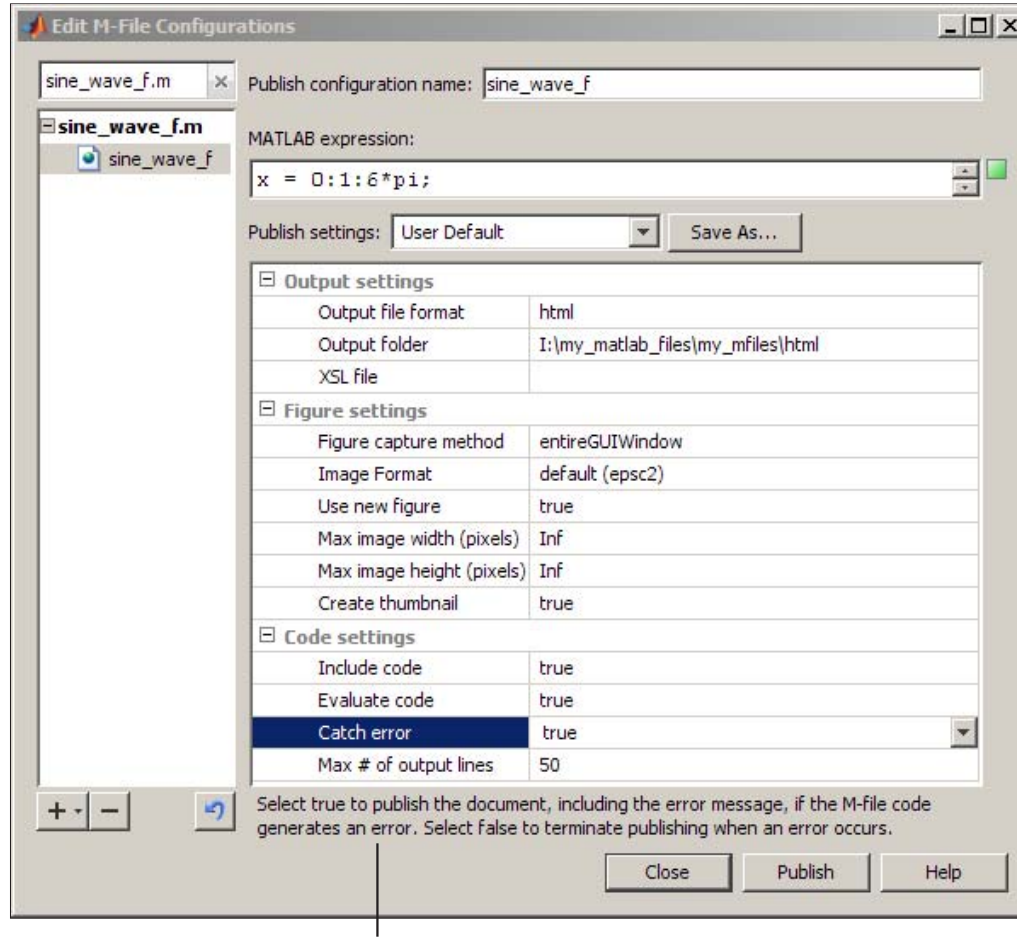
This section describes how to specify new publish settings for a configuration. Publish settings enable you to specify the folder to which a published file is saved, how images generated by the M-code are captured and included in the published document, and so on.

1. If the Edit M-File Configurations dialog box is not already open, click the down arrow on the **Publish** button  and then click the configuration that you want to change.

   This example uses the `sine_wave_f` publish configuration as described in "Creating a Publish Configuration for an M-File" on page 10-70.

2. View the properties table below the **Publish settings** field to see the current publish property values.

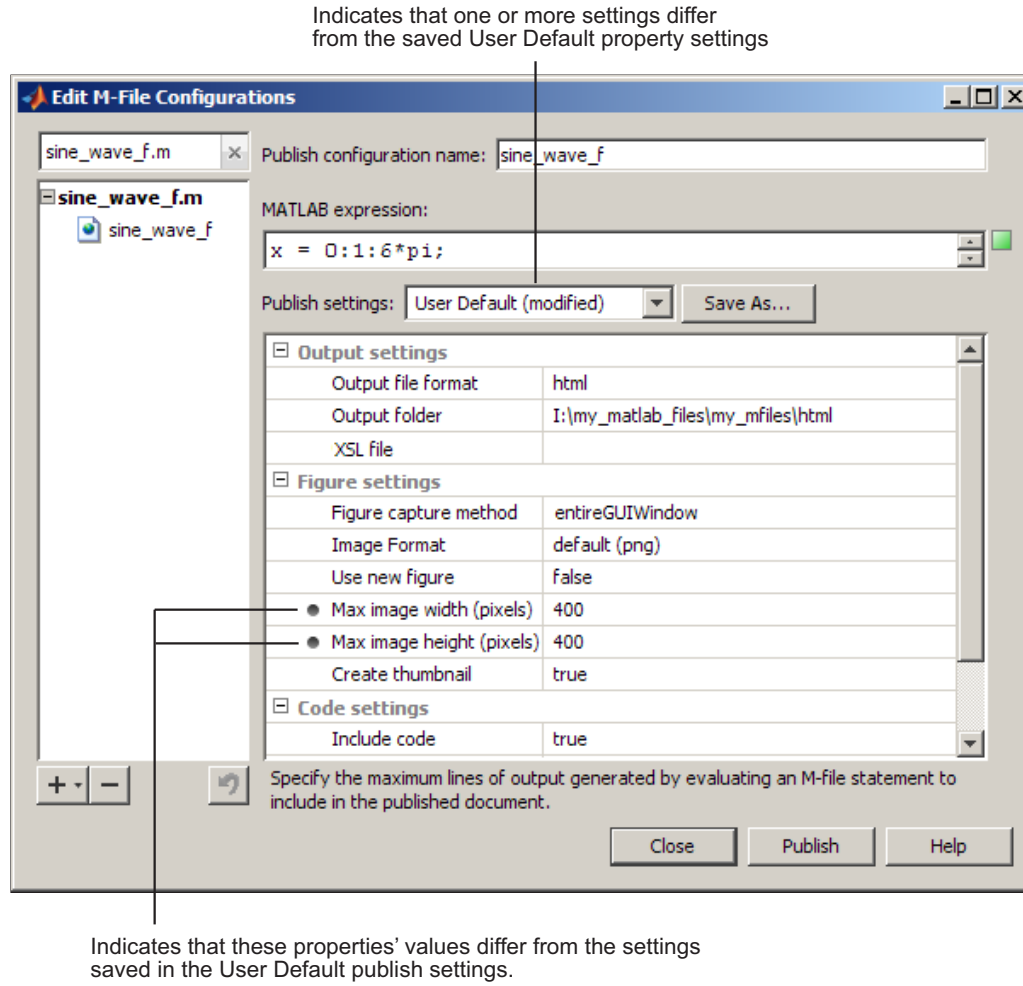
3. For information about a property, click the property name. A brief description of that property displays below the publish settings property table. For example, if you click **Catch error**, the dialog box appears as shown in the following image.



4 Optionally, you can change publish setting values by clicking in the column to the right of the property name and then entering or selecting a property value. This example changes **Max image width** and **Max image height** to `400`.

The Editor marks each property that you change with a dot (●) and adds the string, `(modified)`, next to `User Default` in the **Publish settings** field.

See “Specifying Values for the Publish Settings Property Table” on page 10-78 for information about the various properties you can set.



5 Click **Publish** to preview the publication of the M-file that is open in the Editor using the new settings.

6 When you are satisfied with the results, click **Save As**.

The Save Publish Options dialog box opens and displays the names of all the currently defined publish settings. By default the following publish settings are installed with MATLAB:

- `Factory Default`

  The MATLAB installation includes this named set of publishing properties for you to get started with publishing documents. It enables you to quickly publish an M-file to HTML and view the results. You can use it to test the effect of changing settings on the published output. If you determine that the test settings produce undesirable results, you can restore the `Factory Default` publish settings by selecting it from the **Publish settings** drop down list.

- `User Default`

  The MathWorks installs this named set of publishing properties in anticipation that you will have a set of publish properties that are common to most or all of your publishing configurations. Initially, `User Default` settings are identical to those in the `Factory Default`. See "Creating a Template for Typical Publish Settings" on page 10-93 for an example of changing the User Default settings to best suit your publishing needs.

7 In the **Settings Name** field, enter a meaningful name for the settings. For example, `reduce_image`. Then click **Save**.

You can now use the `reduce_image` publish settings with other publish configurations.

You can also overwrite the publishing properties saved in an existing publish settings name by selecting it from the **Publish settings** drop-down list, and then clicking **Overwrite**. However, you cannot overwrite the `Factory Default` publish settings.

Note When you overwrite a publish settings name, configurations that currently specify that publish settings name do not have their publish properties updated to use the new settings. Instead, their properties that now have different values from the updated publish settings are marked with a dot.

---

8 In the Edit M-File Configurations dialog box, do one of the following:

- Click **Publish** to publish the M-file that is open in the Editor using the new settings.
- Click **Close** to close the dialog box.

Specifying Values for the Publish Settings Property Table

The sections that follow describe each of the publish settings properties that you can adjust to fit your needs when you create or update a publish configuration. To access a publish configuration open the M-file for which you want create or update a publish configuration, and then select **File > Publish Configuration for *`file name`* > Edit Publish Configurations for *`file name`***.

You can set or adjust values for the following properties:

Output file format. Select one of the choices from the drop-down list to publish the document to one of the following file formats:

- `html` — Publishes to an HTML document.
- `xml` — Publishes to an XML document.
- `latex` — Publishes to a LaTeX document.
- `doc` — Publishes to a Microsoft Word document, if your system is a PC.
- `ppt` — Publishes to a Microsoft PowerPoint document, if your system is a PC.
- `pdf` — Publishes to a PDF document.

MATLAB names the published file with the same name as the publish configuration that produced it and stores it, along with images that MATLAB generates from M-file code, in the folder specified with the **Output folder** property.

Output folder. Type the full path of the folder to which you want MATLAB to publish the output document and its associated image files. For example, if your M-file is in `I:\my_matlab_files\my_mfiles`, you might specify `I:\my_matlab_files\my_word_files` if you are creating a publish configuration for documents that you publish to Word.

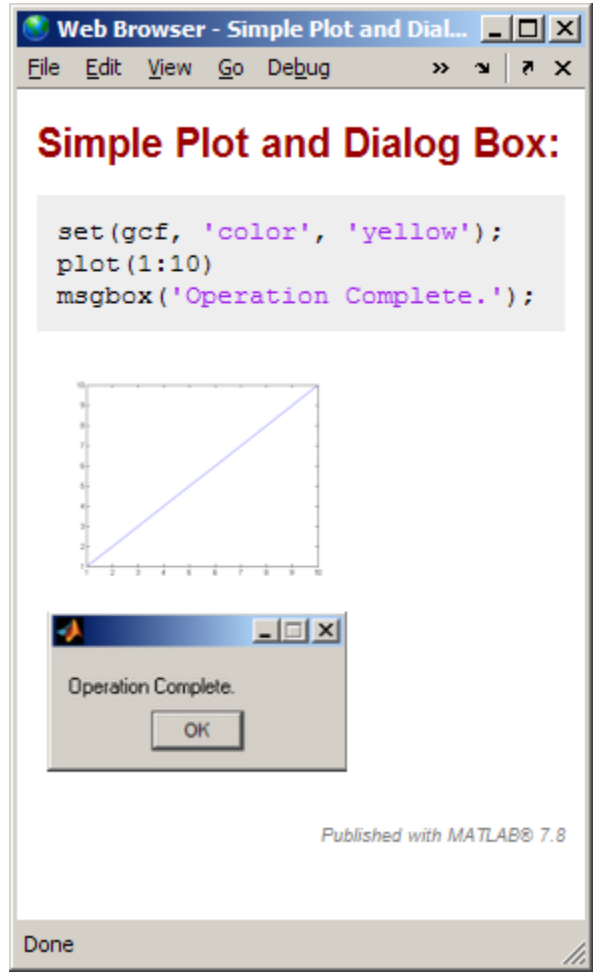
XSL file. Type the full path of the Extensible Stylesheet Language (XSL) file, that you want to use when you specify the **Output file format** as **HTML**, **XML**, or **LaTeX**. If you leave this field blank, MATLAB uses a default stylesheet which is installed with the MATLAB software.

Figure capture method. Specify a figure capture method to indicate how you want figures and dialog boxes that the M-file code creates to appear in published documents.

The following list provides some suggestions on how to set this property, depending on the type of document you are publishing. (The document shown in the images that follow was published with the **Max image width** property set to 150 pixels.)

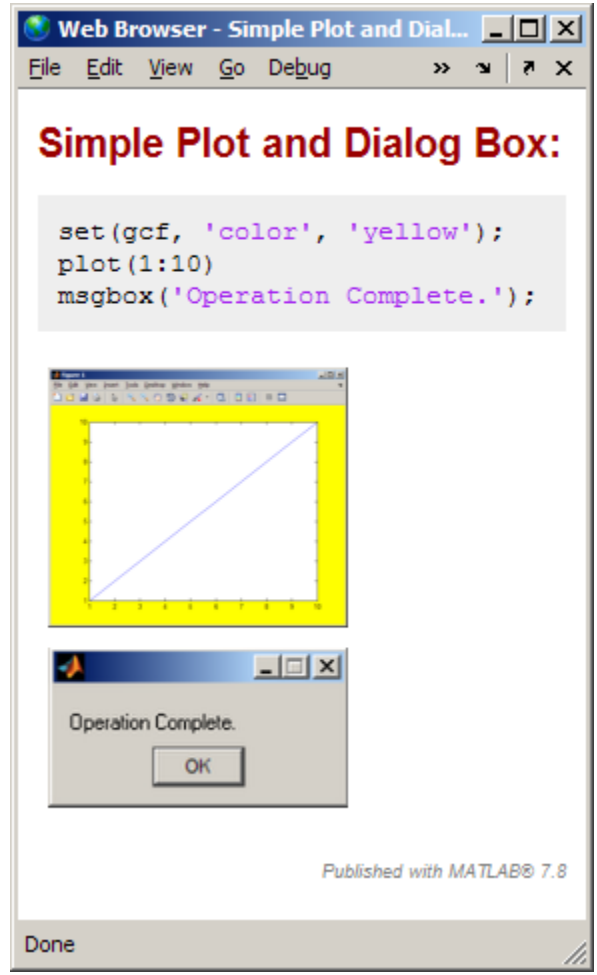
- To create a published document that presents the Figure data and complete dialogs boxes that the M-file creates, set the **Figure capture method** to **entireGUIWindow**.

Notice that this method sets the figure background to white, presents just the plot for the figure, but includes the window decorations (title bar, toolbar, menu bar, and window border) for the dialog box in the published document.

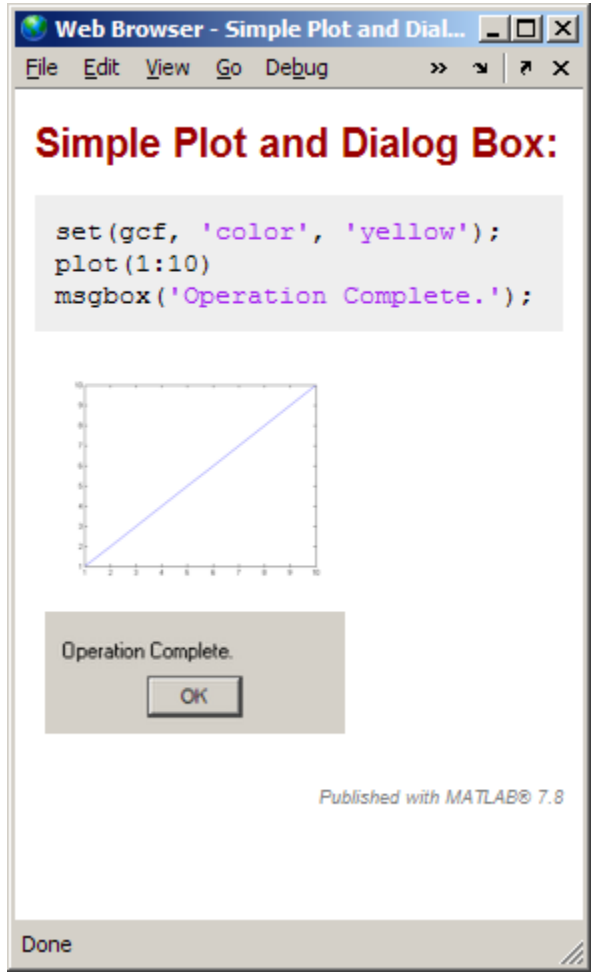


- To create a published document that presents a tutorial on using MATLAB, set the **Figure capture method** to **entireFigureWindow**.

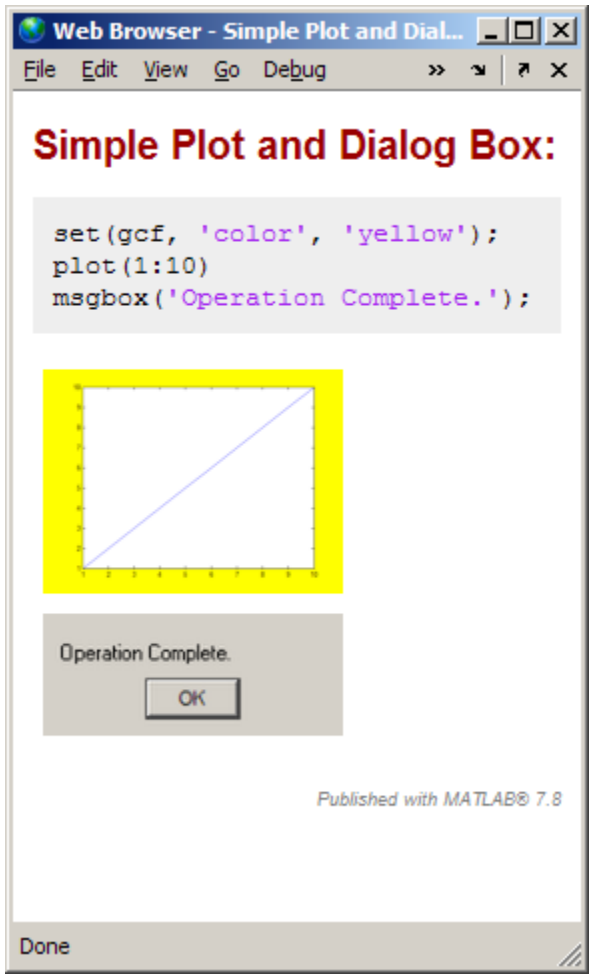
Notice that this method preserves the background color for figures and includes the window decorations for both figures and dialog boxes in the published document.



- To create a published document that presents figures with the plot background set to white and dialog boxes without window decorations, set the **Figure capture method** to **print**.



- To create a published document that presents figures and dialog boxes excluding window decorations, but including the background color for figures, set the **Figure capture method** to **getframe**.



The following table summarizes the effects of the various Figure capture methods.

Use this Figure Capture Method	To Get Figure Captures with these Appearance Details	
	Window Decorations	**Plot Backgrounds**
entireGUIWindow	Included for dialog boxes; Excluded for figures	Set to white for figures; matches the screen for dialog boxes
print	Excluded for dialog boxes and figures	Set to white
getframe	Excluded for dialog boxes and figures	Match the screen plot background
entireFigureWindow	Included for dialog boxes and figures	Match the screen plot background

Note Typically, MATLAB figures have the `HandleVisibility` property set to `on`. Dialog boxes are figures with the `HandleVisibility` property set to `off` or `callback`. If your results are different from those listed in the preceding list and table, the `HandleVisibility` of your figures or dialog boxes might be atypical. For more information, see "HandleVisibility Property" in the MATLAB Graphics documentation.

Image format. Select the file type for images produced when publishing M-files. The image file types available in the drop-down list depend on the **Figure capture method** you specify.

Note When the **Output file type** is `pdf`, the supported **Image Format** properties are `bmp` and `jpeg`.

Use new figure. Set to **true** if you want MATLAB to create a new Figure window with a white background and at the default size before publishing if the M-file code generates a figure. After publishing finishes, MATLAB closes the Figure window.

To use a figure with different properties for publishing, set this property to **false**. Then open a Figure window, change the size and background color, for example, and then publish. Figures in your published document use the characteristics of the figure you opened before publishing.

Note This preference applies to executable M-file code that generates a figure. It does not apply to figures included in a published document using the **Cell > Insert Text Markup > Image** menu option.

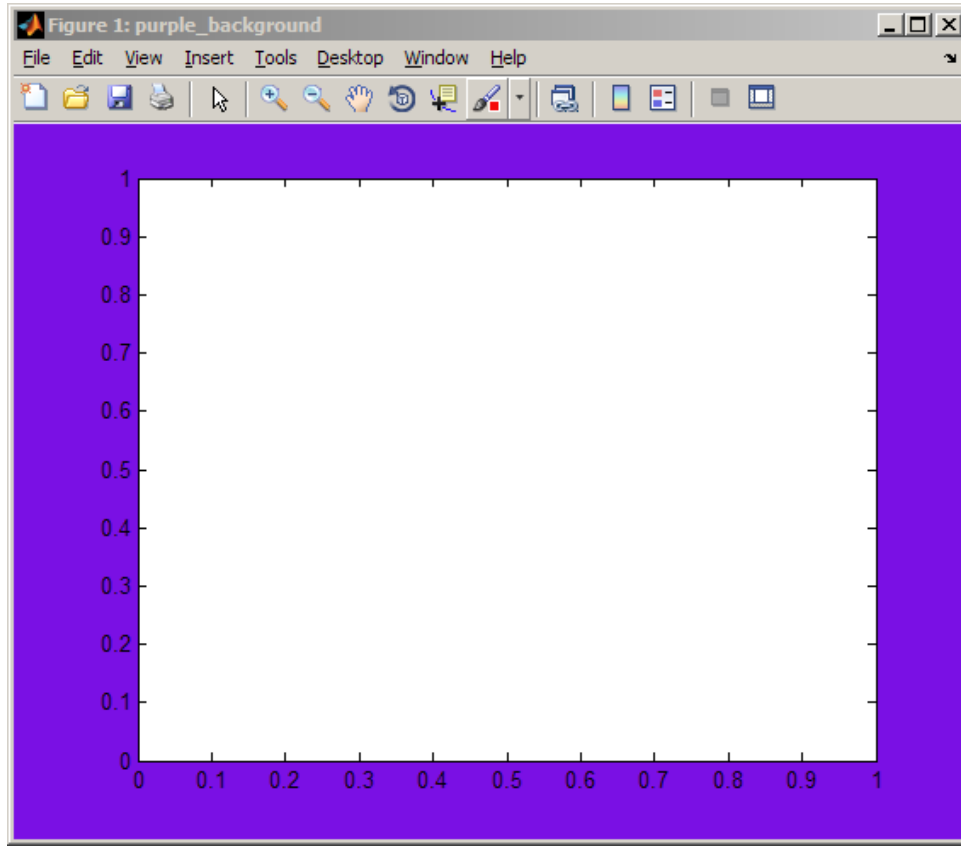
The following example demonstrates how to specify new Figure window properties for published images by setting the **Use new figure** publish settings property to **false**:

1. Create `sine_wave_f.m`, as described in "Creating a Publish Configuration for an M-File" on page 10-70.

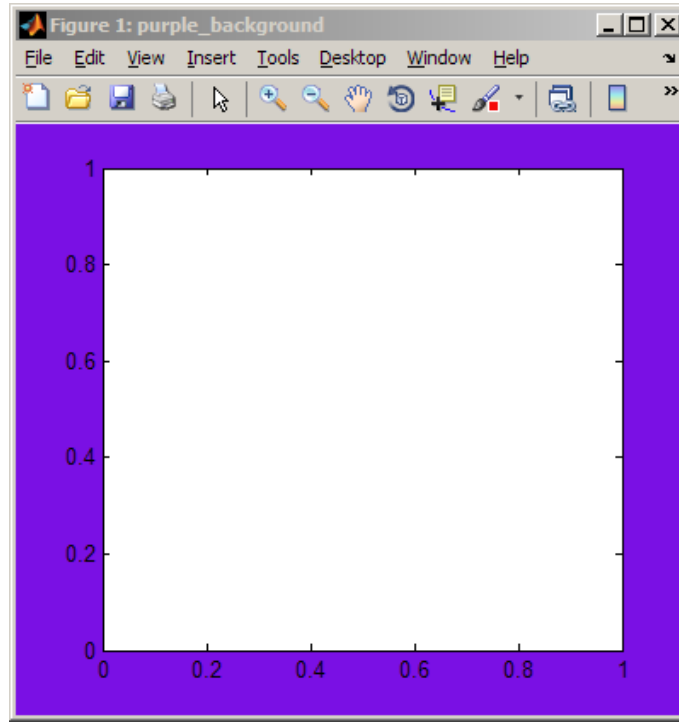
2. Create a Figure window by saving the following code in an M-file and then running it:

```
function createfigure
%CREATEFIGURE

% Create figure
figure1 = figure('Name','purple_background',...
'Color',[0.4784 0.06275 0.8941]);
colormap('hsv');

% Create subplot
subplot(1,1,1,'Parent',figure1);
box('on');

% Create xlabel
xlabel({''});

% Create title
title({''});
```

   The following figure appears.



3 Reduce the size of the figure by dragging and dropping the edges. For example:



4 Do not close the window.

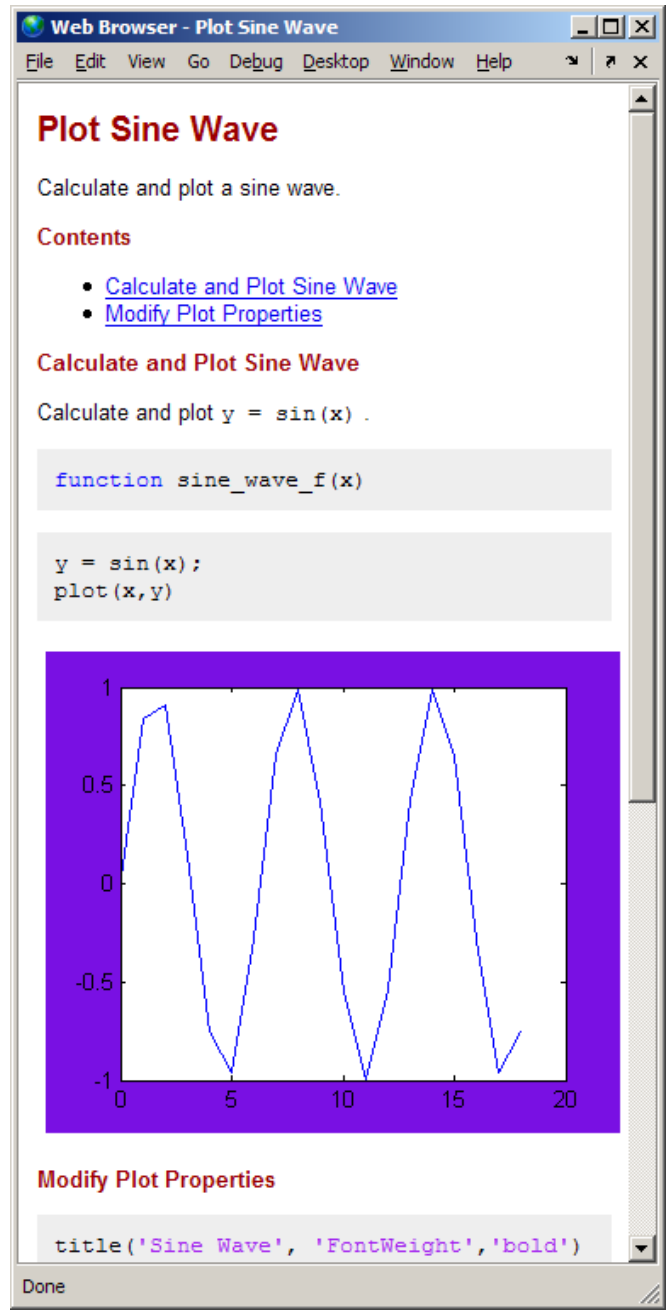
5 Make `sine_wave_f.m` the active file in the Editor, and then select **File > Publish Configurations for sine_wave_f.m > Edit Publish Configurations for sine_wave_f.m.**

6 In the **Publish settings** drop-down list, select `Factory Default`.

7 If you have previously set **Publish settings** for `sine_wave_f.m`, the Change Publish Settings dialog box opens. Click **Change to Factory Default**.

8 In the **Publish settings** properties table, set **Use new figure** to **false**.

9 Click **Publish**. MATLAB publishes `sine_wave_f.m` as shown in the following figure.


Web Browser - Plot Sine Wave
File Edit View Go Debug Desktop Window Help
Plot Sine Wave
Calculate and plot a sine wave.
Contents
Calculate and Plot Sine Wave
Modify Plot Properties
Calculate and Plot Sine Wave
Calculate and plot y = sin(x) .
function sine_wave_f(x)
y = sin(x);
plot(x,y)
1
0.5
0
-0.5
-1
0
5
10
15
20
Modify Plot Properties
title('Sine Wave', 'FontWeight','bold')
Done

Max image width. Overwrite the current value to restrict the width of images in the published output. Note the following about this property:

- It applies only to images that the M-code generates. It does not apply to images you include using the method described in "Specifying Graphics in M-Files for Publishing" on page 10-31.
- It applies when you select an **Image Format** property setting that is a bitmap, such as `.png` or `.jpg`.
- It does not apply when the **Image Format** property setting is a vector format, such as `.eps`.
- The image's aspect ratio is maintained. If you restrict both height and width using **Max image width** and **Max image height** properties to resize the image, then MATLAB maintains the aspect ratio by using the maximum you specified for one dimension and something less than the maximum for the other dimension.
- MATLAB ignores this property when the **Output file format** is `pdf`.

Max image height. Overwrite the current value to restrict the height of images in the published output. Note the following about this property:

- It applies only to images that the M-code generates. It does not apply to images you include using the method described in "Specifying Graphics in M-Files for Publishing" on page 10-31.
- It applies when you select an **Image Format** property setting that is a bitmap, such as `.png` or `.jpg`.
- It does not apply when the **Image Format** property setting is a vector format, such as `.eps`.
- The image's aspect ratio is maintained. If you restrict both width and height using **Max image width** and **Max image height** properties to resize the image, then MATLAB maintains the aspect ratio by using the maximum you specified for one dimension and something less than the maximum for the other dimension.
- MATLAB ignores this property when the **Output file format** is `pdf`.

Create thumbnail. Set to **true** to direct MATLAB to create a thumbnail image if the **Image Format** preference is a bitmap, such as `.png` or `.jpg`. For example, you can use this thumbnail to represent your M-file in HTML pages. If you create your own demos and include them in the Help browser **Demos** pane via a `demos.xml` file, MATLAB automatically creates a list of your demos that includes the thumbnail for each.

Set to **false** to direct MATLAB to not create a thumbnail image.

Include code. Set to **true** to have MATLAB include the M-file code in the published document. Set to **false** to the have MATLAB exclude the code from all output file formats except HTML. When the output file format is HTML, MATLAB inserts the M-file code in the output file as an HTML comment. Therefore, when viewed in a Web browser, for example, the M-file code is not displayed.

Use the MATLAB `grabcode` function if you want to extract the M-file code from the published HTML file.

For example, suppose you publish `I:/my_matlabfiles/my_mfiles/sine_wave_f.m` to HTML using a publish configuration with the **Include code** property set to **false**. If you share the published document with colleagues, they can view the published document in a Web browser. If your colleagues want to see the M-file code that generated the published document, they can issue the following command from the folder containing `sine_wave_f.html`:

```
grabcode('sine_wave_f.html')
```

MATLAB opens the M-file code that created `sine_wave_f.html` in the Editor.

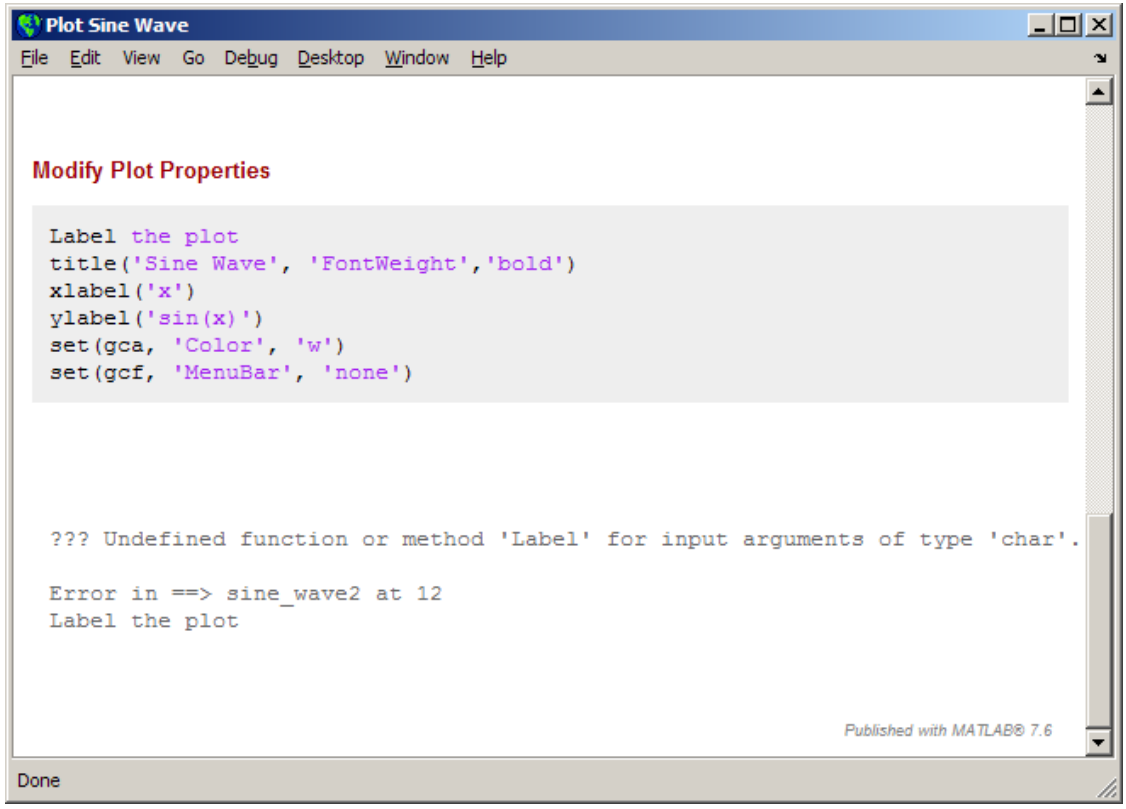
See “Creating a Publish Configuration for an M-File” on page 10-70 for the `sine_wave_f.m` code.

Evaluate code. Set to **true** to direct MATLAB to evaluate the M-file code while publishing the results and include the results in the output document.

Set to **false**, to direct MATLAB to not evaluate the code nor include code results when publishing an M-file.

Because MATLAB does not evaluate the code, there might be invalid code in the M-file. Therefore, you might not want to set this property to **false** without first running the M-file.

For example, suppose you include comment text, `Label the plot`, in an M-file, but forget to preface it with the comment character. If you publish the document to HTML, and set **Evaluate code** to **true**, the published document includes the error, such as shown in the following figure.



Use this property with the **Max # of output lines** property to specify the maximum number of lines you want to include in the output. This property is helpful when you have code that produces a lot of output and you only want to include a sample of it in the published document.

Catch error. Set to **true** to direct MATLAB to publish and include the error message text in the published document if an error occurs when it evaluates the code.

Set to **false** to direct MATLAB to terminate the publish operation if an error occurs when it evaluates the code.

This property has no effect if you set the **Evaluate code** property to false.

Max # of output lines. Type a value to specify the maximum number of output lines that you want to include after each cell break in the published document.

For example, suppose your M-file code includes a loop, such as the following:

```
for n = 1:100
  disp(x)
end;
```

If you publish the document, then by default, all 100 lines of the output generated by the preceding code is included in the published document. If you want to include a smaller representative sample of the output, set **Max # of output lines** to a small value, such as 10.

Creating a Template for Typical Publish Settings

Use the **User Default** publish settings installed with MATLAB to create a template for all or most of your publish configurations.

Initially, the `User Default` publish setting has the same property values as the `Factory Default` publish settings. Update and save your most commonly used property settings to avoid having to reset the same settings each time you create a new publish configuration.

For example, suppose that you frequently publish your M-files using the factory installed `User Default` settings, with a few exceptions. You want to change the factory installed `User Default` settings to:

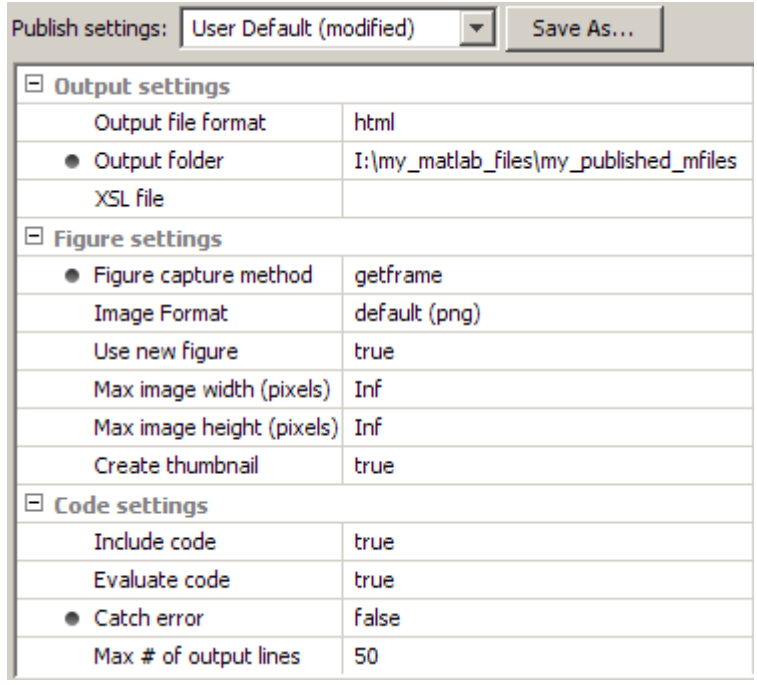
- Save the published document files to `I:\my_matlab_files\my_published_mfiles`

- Use the `getframe` figure capture method
- Terminate publishing if an error occurs while the M-file code is being evaluated

Update the `User Default` publish settings, as follows:

1. If the Edit M-File Configurations dialog box is not already open, click the down arrow on the **Publish** button, and then click the configuration for which you want to set the properties as described in the preceding list.
2. From the **Publish settings** drop-down list, select `User Default`.

   If the Change Publish Settings dialog box opens, click **Change to User Default**.
3. Adjust the values in the publish settings properties table, so that the **Publish settings** appear as shown in the following figure.



4. Click **Save As**.

   The Save Publish Settings dialog box opens.

5. In the **Publish settings** drop-down list, select `User Default`, and then click **Overwrite**.

   The `User Default` publish settings are now saved with the specified property values.

Now, suppose you want to create a publish configuration using all the same settings, except you want to publish your M-file to a Microsoft Word document. Follow these steps:

1. In the Editor, open the M-file that you want to publish to a Word document.

2. Click the down arrow next to the Publish button  on the Editor toolbar and click **Edit Publish Configuration for *`file name`***, where the file name is the name of the file that you want to publish to a Word document.

   The Edit M-File Configurations dialog box opens.

3. If you want, adjust the MATLAB expression.

4. Notice that the **Publish settings** are set to **User Default** and the publish settings properties table contains the values you set in the preceding list of steps.

5. Change the **Output file format** from **html** to **doc**.

6. Click **Save As**.

   The Save Publish Settings dialog box opens.

7. In the **Settings name** box, type a name for the new group of publish settings. For example, `WordDefault`.

8. Click **Save**.

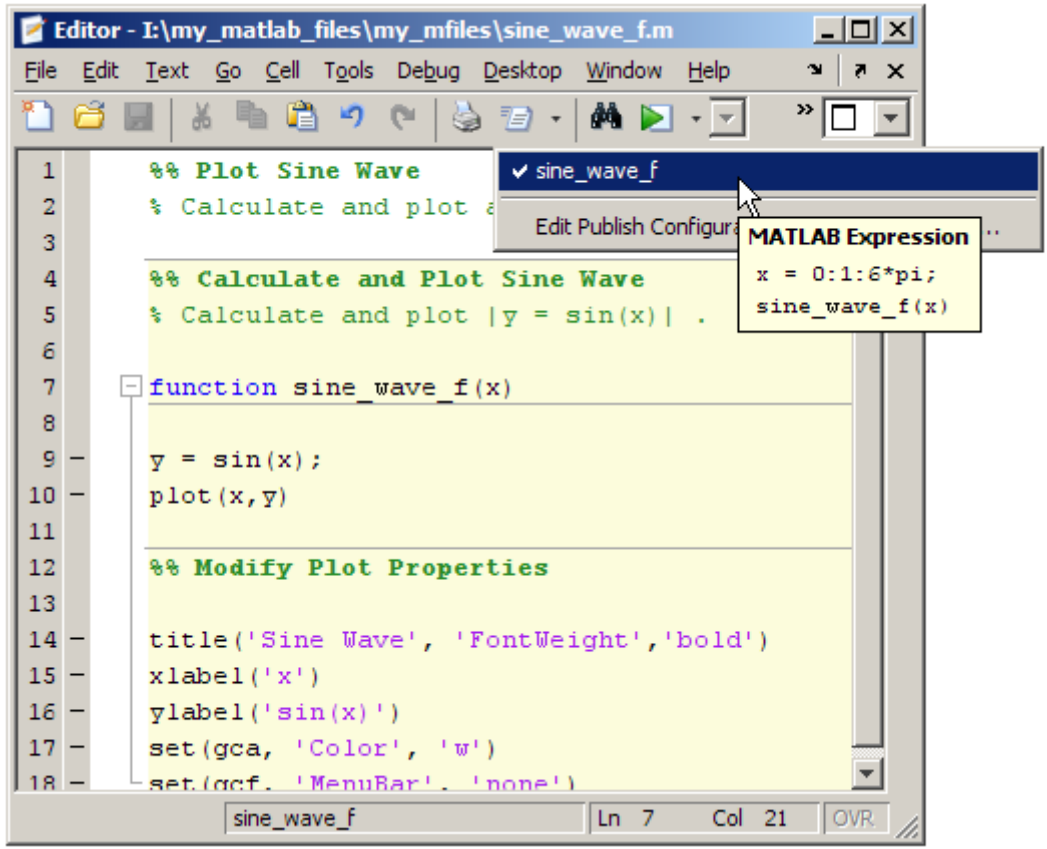
Now you can use any one of the following publish settings as the basis for new publish settings, for the next publish configuration you create:

- `Factory Default`
- Your customized `User Default`
- `Word Default`
- Any other publish settings that you create and save with a unique name

Running an Existing Publish Configuration

After creating a publish configuration, you can run the configuration without opening the Edit M-File Configurations dialog box, as follows:

1. In the Editor toolbar, click the down arrow on the Publish button **Publish** button  , and position the mouse pointer on a publish configuration name. MATLAB displays a Tooltip showing the publish configuration's MATLAB expression so you can see what will be evaluated when you publish the M-file using the named configuration.



2 To use the publish configuration, select a configuration name. MATLAB publishes the M-file using the MATLAB expression you specified in the publish configuration. For example, if you select `sine_wave_f`, MATLAB sets the value of the input argument, `x`, to `0:1:6*pi` and passes it to the M-file function before evaluating and publishing it. (To see how to set the MATLAB expression, see "Creating a Publish Configuration for an M-File" on page 10-70.)

Creating Multiple Publish Configurations for an M-File

You can create multiple publish configurations for a given M-file. You might do this to publish the M-file with different values for input arguments, with different publish setting property values, or both. Create a named

configuration for each purpose, all associated with the same M-file. Then, any time you publish the M-file, you can chose and run whichever particular publish configuration that you want. For example, for `sine_wave_f(x)` you might use different values for `x` and adjust publishing properties for these purposes:

- For reviewing with colleagues, publish the document to Word. Use publish settings to adjust the size of images generated by the code so they are not cropped in the document. Evaluate and include the code, as well as any errors generated by the code in the Word document.
- For inclusion in a blog, publish the document to HTML. Use publish settings to specify an argument value and set publishing properties to evaluate and include the code, but exclude errors generated by the code from the output published to HTML.
- For presentation at a meeting, use the same settings as used for publishing to the blog, but publish to Microsoft PowerPoint.
- For a polished presentation, publish to PDF.

The following sections provide instructions for creating multiple configurations for `sine_wave_f.m`.

- "Example of Publishing sine_wave_f.m to Microsoft Word" on page 10-98
- "Steps for Publishing sine_wave_f.m to HTML" on page 10-101
- "Steps for Publishing sine_wave_f.m to Microsoft® PowerPoint" on page 10-105
- "Steps for Publishing sine_wave_f.m to PDF" on page 10-107

Example of Publishing sine_wave_f.m to Microsoft Word

The following steps provide an example of settings you might use when you want to publish an M-file to Word. This example uses the `sine_wave_f.m` file, the code for which is presented in "Creating a Publish Configuration for an M-File" on page 10-70.

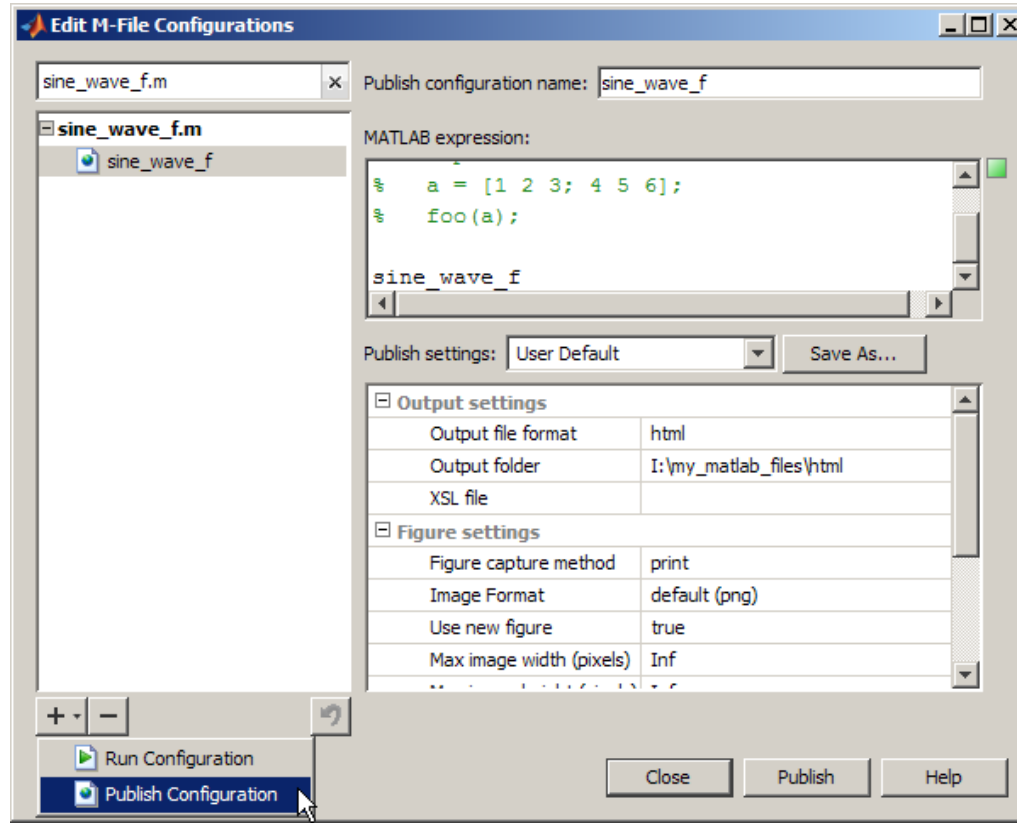
1. Copy `sine_wave_f.m` to your current folder. If you have write permission to your current folder, you can type the following in the Command Window to copy the file from the MATLAB root directory:

```
copyfile(fullfile(docroot,'techdoc','matlab_env','examples', ...
'sine_wave_f.m'),'.','f')
```

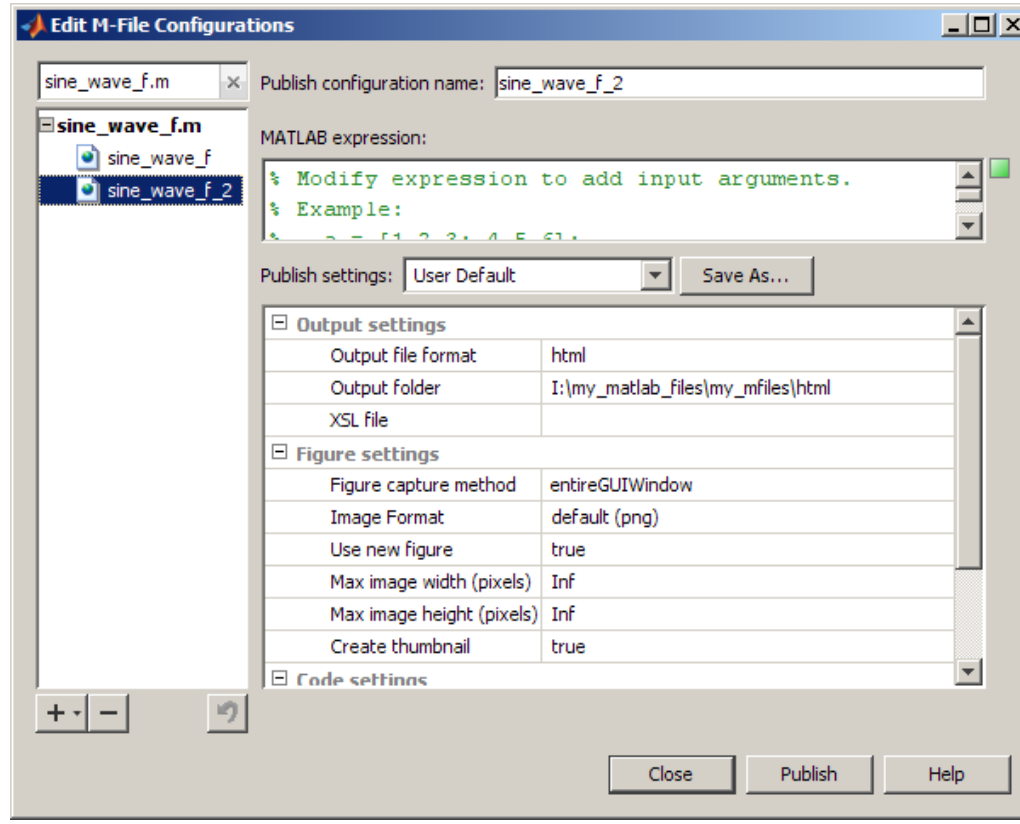
2. In the Editor, open `sine_wave_f.m`.

3. Select **File > Publish Configuration for sine_wave_f.m > Edit Publish Configurations for sine_wave.m**.

4. Select `sine_wave_f` in the list of M-files and configurations, click the down arrow next to the **Add** button +, and then select **Publish Configuration**.



MATLAB creates a new publish configuration, `sine_wave_f_n` where the value of *n* depends on the number of publish configurations you have previously created for `sine_wave_f`.



5 Rename `sine_wave_f_n` to `sine_wave_word`, and replace the default template expression with the following code:

```
x = 0:1:rand*pi;
sine_wave_f(x)
```

6 Change the values for **Publish settings**, as follows so that the M-file is published to a Word document, including the code, its output and any

errors the code may generate. The maximum values for the image height and width are set so that the images are not cropped in the Word document:

a For **Output file format**, select `doc` from the drop-down list.

b For **Image format**, select `jpeg` from the drop-down list.

c For **Max image width**, type 400.

d For **Max image height**, type 400.

7 Click **Publish** to test how the settings affect the Word document.

You can continue to test and change publish settings until you achieve the results that you want.

Tip In addition to testing that your M-file code evaluates as expected and publishes to Word as expected, you might run the spelling and grammar checker in Word to be sure that the comments in your M-file do not contain typographical or grammatical errors.

---

8 Optionally, if you plan to reuse these publish settings later, click **Save As**. In the Save Publish Settings dialog box, in the **Settings name** field, type `word_settings`, and then click **Save**.

Steps for Publishing sine_wave_f.m to HTML

These steps provide an example of creating a configuration for `sine_wave_f.m`, that publishes the M-file to HTML. You might do this to publish output for inclusion in a blog, for example.

1 Copy `sine_wave_f.m` to your current folder. If you have write permission to your current folder, you can type the following in the Command Window to copy the file from the MATLAB root directory:

```
copyfile(fullfile(docroot,'techdoc','matlab_env','examples', ...
'sine_wave_f.m'),'.','f')
```

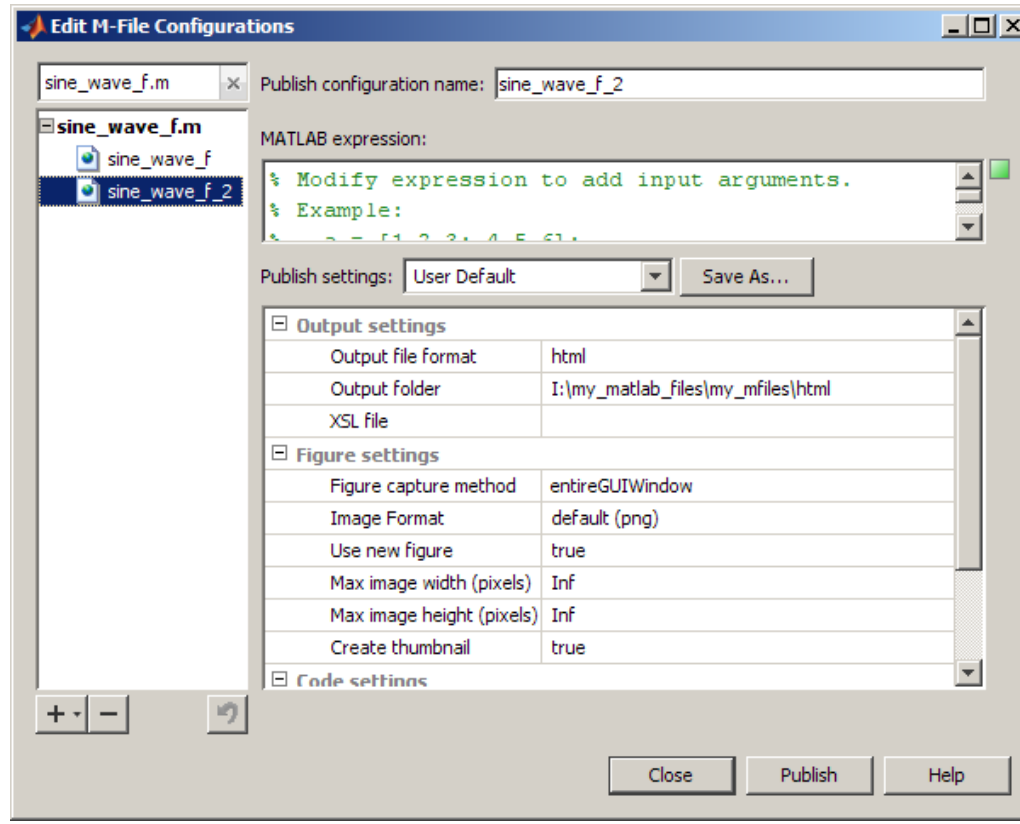
2 In the Editor, open `sine_wave_f.m`.

3 Select **File > Publish Configuration for sine_wave_f.m > Edit Publish Configurations for sine_wave_f.m**.

Select `sine_wave_f` in the list of M-files and configurations, click the down arrow next to the **Add** button +, and then select **Publish Configuration**.

4 Select `sine_wave_f` in the list of M-files and configurations, click the down arrow next to the **Add** button +, and then select **Publish Configuration**.

MATLAB creates a new publish configuration, `sine_wave_f_n` where the value of *n* depends on the number of publish configurations you have previously created for `sine_wave_f`.

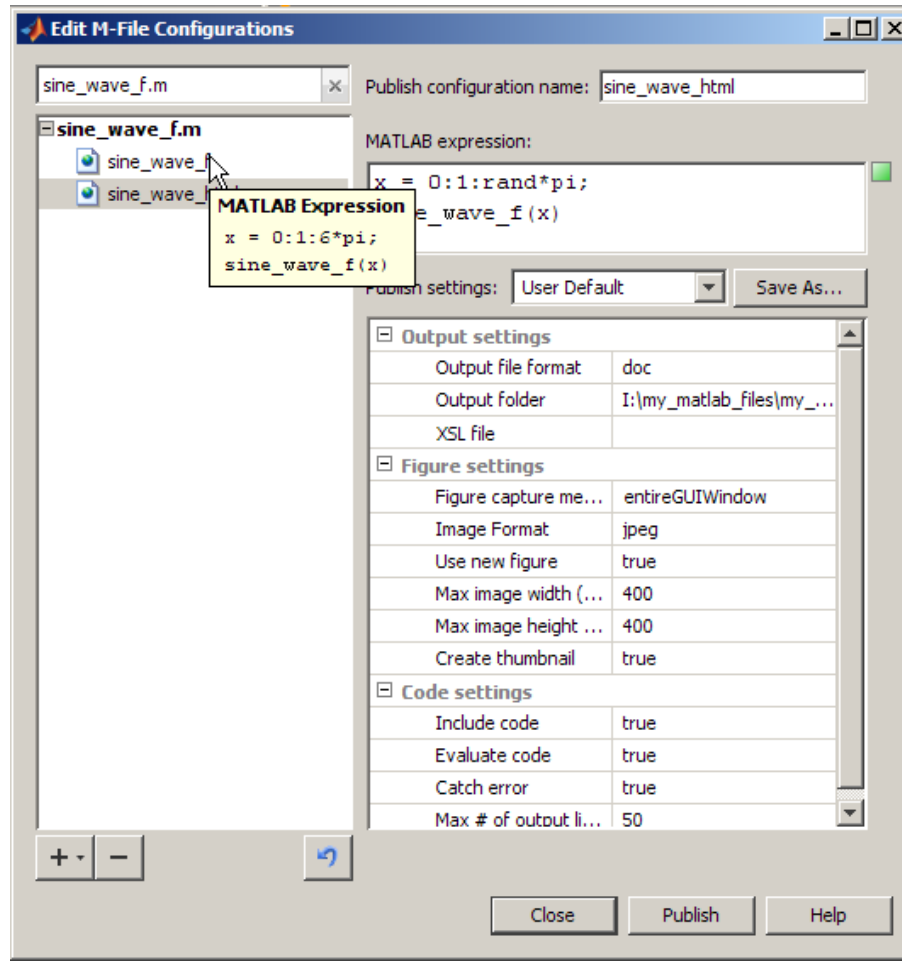


5 In the **Publish configuration name** field, replace `sine_wave_f_n` with `sine_wave_html`.

6 In the **MATLAB expression** field, replace the default expression with the following:

```
x = 0:1:rand*pi;
sine_wave_f(x)
```

Tip To get a quick view of the expression used in a different configuration, position the mouse pointer on the name of a different publish configuration without selecting it. In the following figure, `sine_wave_html` is selected, but the mouse pointer is positioned on `sine_wave_f`. You can see the MATLAB expression specified for the `sine_wave_f` configuration in the Tooltip.



7 Change the values for **Publish settings**, as follows so that the M-file is published to an HTML document, including the code, its output and any errors the code may generate. The maximum values for the image height and width are set so that the images are not cropped in the Word document:

   a For **Output file format**, select `html` from the drop-down list.

   b For **Max image width**, type 400.

   c For **Max image height**, type 400.

8. Click **Publish** to test how the settings affect the HTML document.

   You can continue to test and change publish settings until you achieve the results that you want.

9. Optionally, if you plan to reuse these publish settings later, click **Save As**. In the Save Publish Settings dialog box, in the **Settings name** field, type `html_settings`, and then click **Save**.

Steps for Publishing sine_wave_f.m to Microsoft PowerPoint

These steps provide an example of creating a configuration for `sine_wave_f.m`, that publishes the M-file to Microsoft PowerPoint. You might do this to publish output for presentation in a meeting, for example.

1. Copy `sine_wave_f.m` to your current folder. If you have write permission to your current folder, you can type the following in the Command Window to copy the file from the MATLAB root directory:

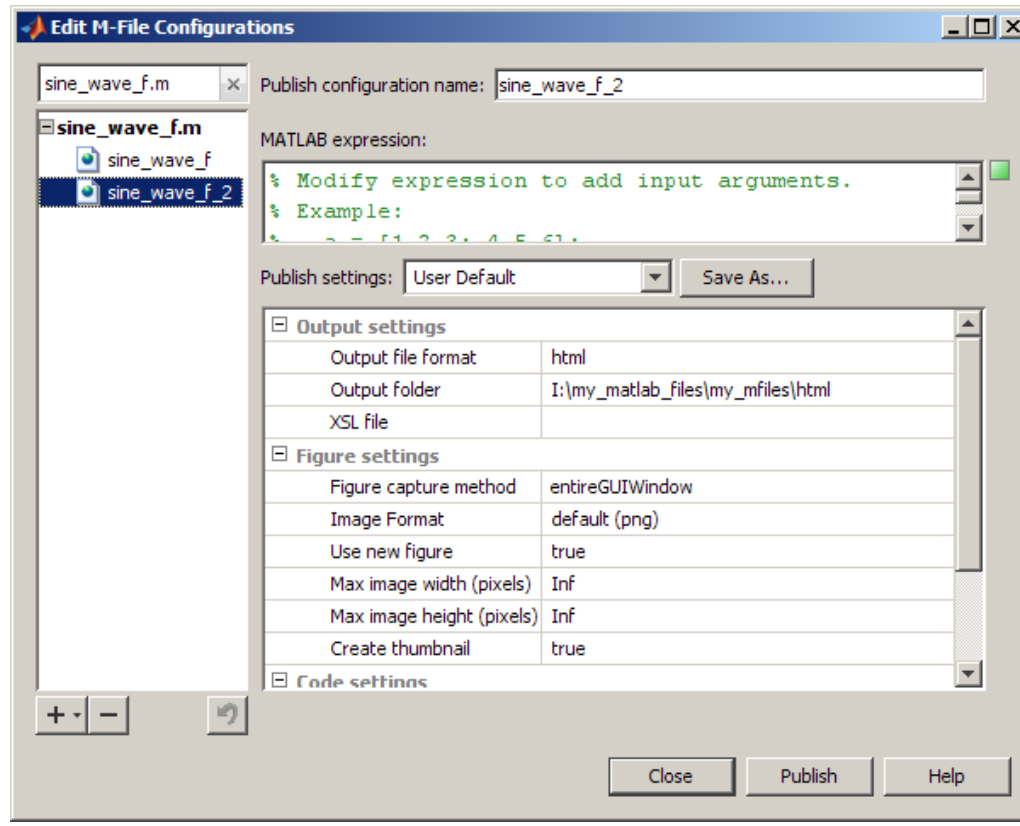
   ```
   copyfile(fullfile(docroot,'techdoc','matlab_env','examples', ...
   'sine_wave_f.m'),'.','f')
   ```

2. In the Editor, open `sine_wave_f.m`.

3. Select **File > Publish Configuration for sine_wave_f.m > Edit Publish Configurations for sine_wave_f.m**.

4. Select `sine_wave_f` in the list of M-files and configurations, click the down arrow next to the **Add** button  , and then select **Publish Configuration**.

   MATLAB creates a new publish configuration, `sine_wave_f_`*n* where the value of *n* depends on the number of publish configurations you have previously created for `sine_wave_f`.



5 In the **Publish configuration name** field, replace `sine_wave_f_`*n* with `sine_wave_ppt`.

6 In the **MATLAB expression** field, replace the default expression with the following:

```
x = 0:1:6*pi;
sine_wave_f(x)
```

7 Assume for the purposes of a PowerPoint presentation, you do not want to include the code.

Change the **Output file format** to **ppt** and **Include code** to **false**.

8 Click **Publish** to test how the PowerPoint output appears.

9 Optionally, if you plan to reuse these publish settings later, click **Save As**. In the Save Publish Settings dialog box, in the **Settings name** field, type `ppt_settings`, and then click **Save**.

Steps for Publishing sine_wave_f.m to PDF

This example uses the `sine_wave_f.m` file, the code for which is presented in "Creating a Publish Configuration for an M-File" on page 10-70.

1 Copy `sine_wave_f.m` to your current folder. If you have write permission to your current folder, type the following in the Command Window to copy the file from the MATLAB root directory:

```
copyfile(fullfile(docroot,'techdoc','matlab_env','examples', ...
'sine_wave_f.m'),'.','f')
```

2 Open `sine_wave_f.m` in the Editor.

```
edit sine_wave_f.m
```

3 Select **File > Publish Configuration for sine_wave_f.m > Edit Publish Configurations for sine_wave_f.m**.

4 Select `sine_wave_f` in the list of M-files and configurations, click the down arrow next to the **Add** button  , and then select **Publish Configuration**.

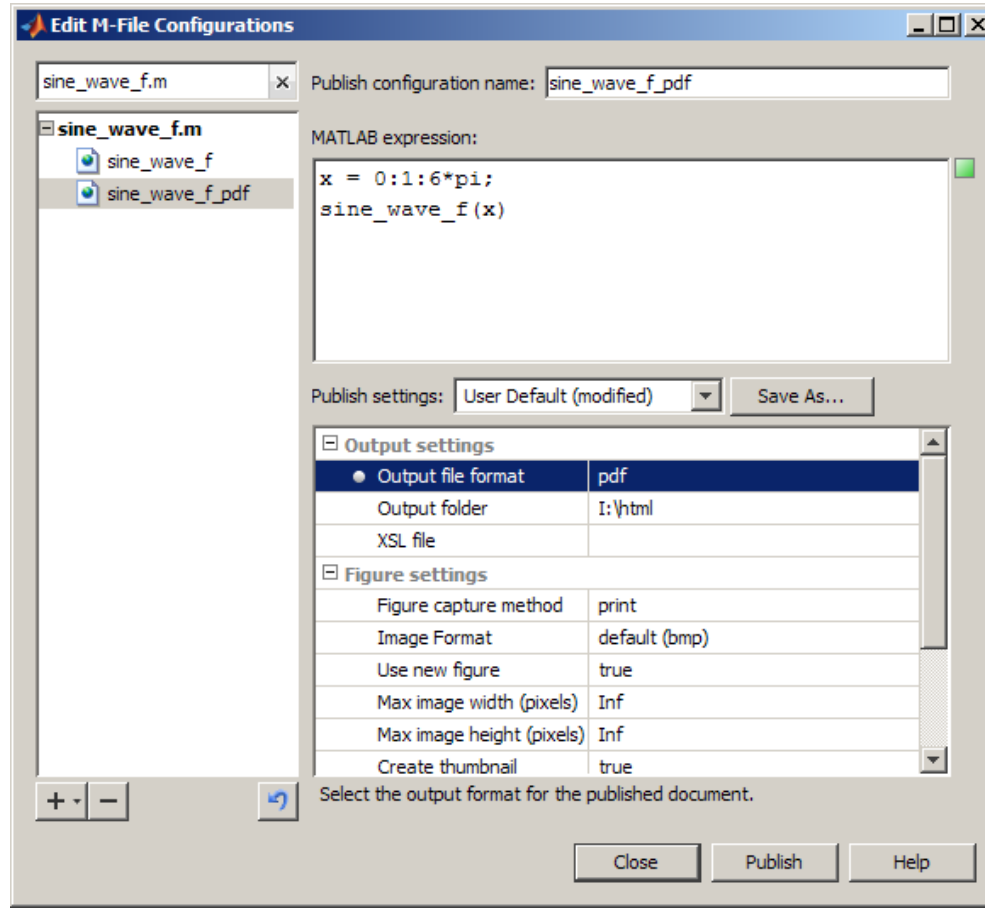
MATLAB creates a publish configuration, `sine_wave_f_`*n*, where the value of *n* depends on the number of publish configurations you have previously created for `sine_wave_f`.

5 In the **Publish configuration name** field, replace `sine_wave_f_`*n* with `sine_wave_pdf`.

6 In the **MATLAB expression** field, replace the default expression with the following:

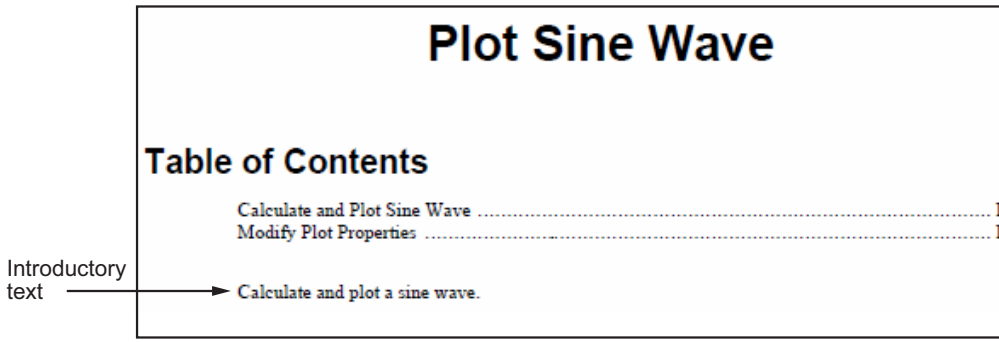
```
x = 0:1:6*pi;
sine_wave_f(x)
```

7 Change the **Output file format** to `pdf`.



8 Click **Publish** to test how the PDF output appears.

Notice that when you publish to PDF, unlike other publishing output formats, the introductory text appears after the table of contents:



9 Optionally, if you plan to reuse these publish settings later, click **Save As**. In the Save Publish Settings dialog box, in the **Settings name** field, type `pdf_settings`, and then click **Save**.

About the publish_configurations.m File

When you create one or more publish configurations using the Edit M-File Configurations dialog box, the Editor updates the `publish_configurations.m` file in your preferences folder. (This is the folder that MATLAB returns when you run the MATLAB `prefdir` function.)

Although you can port this file from the preferences folder on one system to another, there can only be one `publish_configurations.m` file on a system. Therefore, you should only do this if you have not already created configurations on the second system. In addition, because this file may contain references to file paths, you need to be sure the specified M-files and paths exist on the second system.

The MathWorks recommends that you not update `publish_configurations.m` in the MATLAB Editor or a text editor. Changes that you make using tools other than the Edit M-File Configurations dialog box may be overwritten later. Each time you save a configuration using the Edit M-File Configurations dialog box, MATLAB updates the `publish_configurations.m` file, as well as the `run_configurations.m` file. For more information, see "About the run_configurations.m File" on page 8-108.

Finding Publish Configurations

The method you use to find publish configurations is the same as the one you use to find run configurations. For details, see “Find Configurations” on page 8-108.

Removing Publish Configurations

If you no longer need a publish 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 publish configuration. The method you use to delete publish configurations is the same as the one you use to delete run configurations. For details, see “Remove Configurations” on page 8-110 for details.

Reassociating and Renaming Publish Configurations

Each publish 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. The method you use to reassociate and rename publish configurations is the same as the one you use to reassociate and rename run configurations. See “Reassociate and Rename Configurations” on page 8-111 for details.

11

Using Notebook to Publish to Microsoft Word

Notebook is useful for creating electronic or printed records of MATLAB sessions, class notes, textbooks or technical reports to Microsoft Word. As an alternative to Notebook, consider using cells to publish to Microsoft Word. For more information, see Chapter 10, "Publishing M-Files".

Note Notebook is available only on Windows systems that have Microsoft Word installed. For supported versions of Word, see "Configuring Notebook" on page 11-27.

About Using Notebook to Publish to Word

In this section...

Using Notebook to Create an M-book

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.

Creating or Opening an M-Book

This section includes information on performing the following tasks:

- "Creating an M-Book from the MATLAB Desktop" on page 11-2
- "Creating an M-Book While Running Notebook" on page 11-5
- "Opening an Existing M-Book" on page 11-6
- "Converting a Word Document to an M-Book" on page 11-7

Creating an M-Book from the MATLAB Desktop

To create a new M-book from within MATLAB desktop, type the following in the Command Window:

```
notebook
```

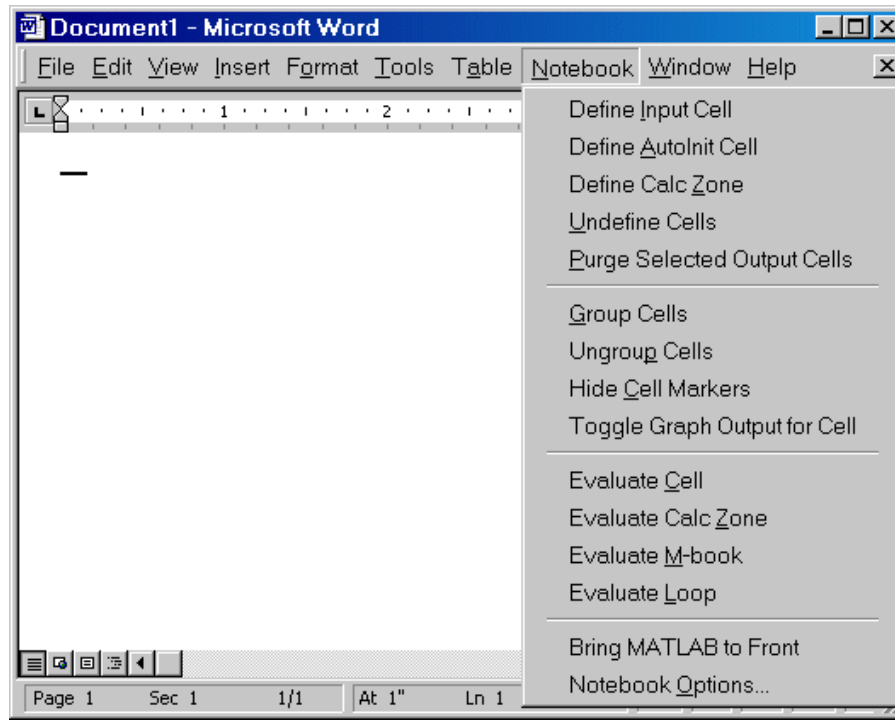
If you are running Notebook for the first time, you might need to configure it. See “Configuring Notebook” on page 11-27 for more information.

Notebook starts Microsoft Word on your system and creates a new M-book, called `Document1`.

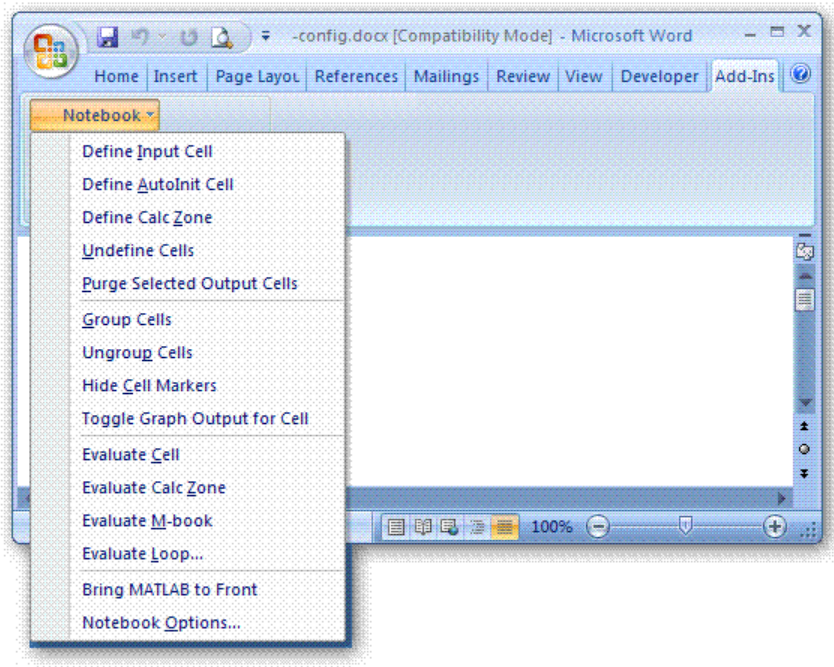
When Word is opening, if a dialog box appears asking you to enable or disable macros, choose to enable macros. Notebook defines Microsoft Word macros that enable MATLAB to interpret the different types of cells that hold MATLAB commands and their output. For more information on macro security, see “Configuring Notebook” on page 11-27.

Depending on the version of Word you are using, one of the following occurs:

- In Word 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.

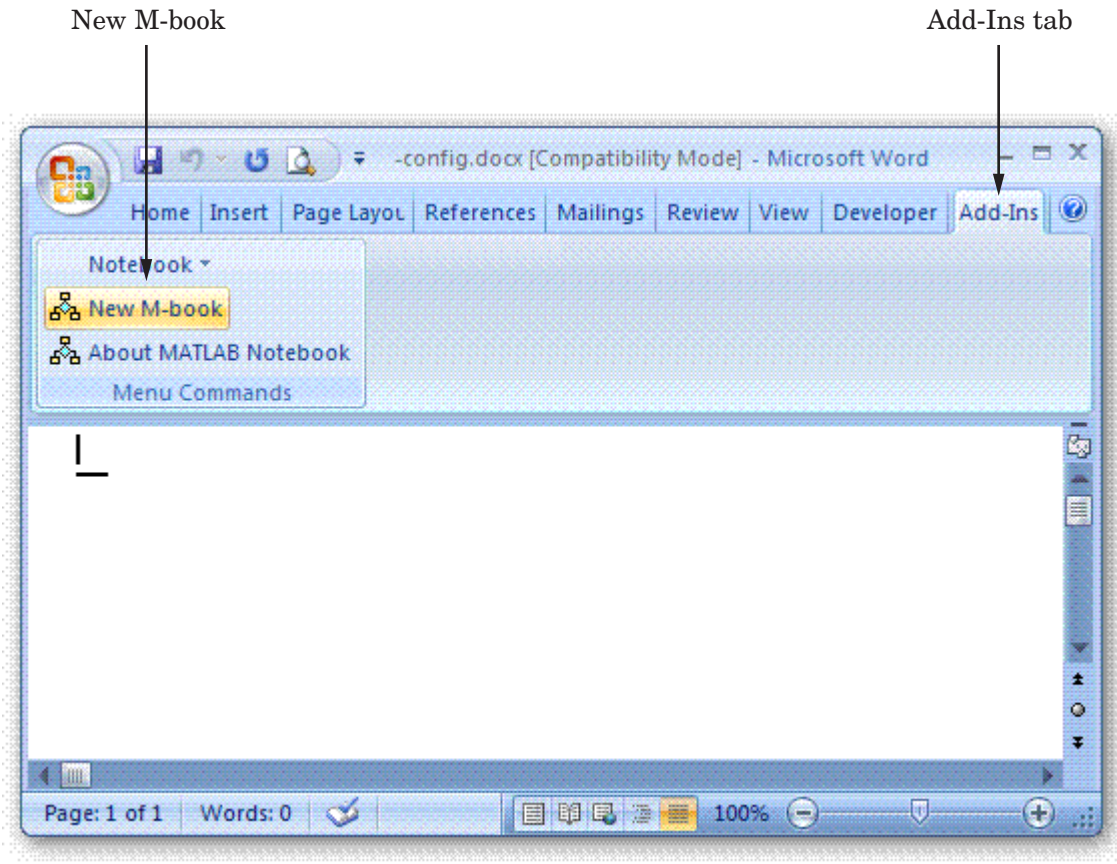


Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Creating an M-Book While Running Notebook

With Notebook running, you can create a new M-book as follows:

- In Word 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.

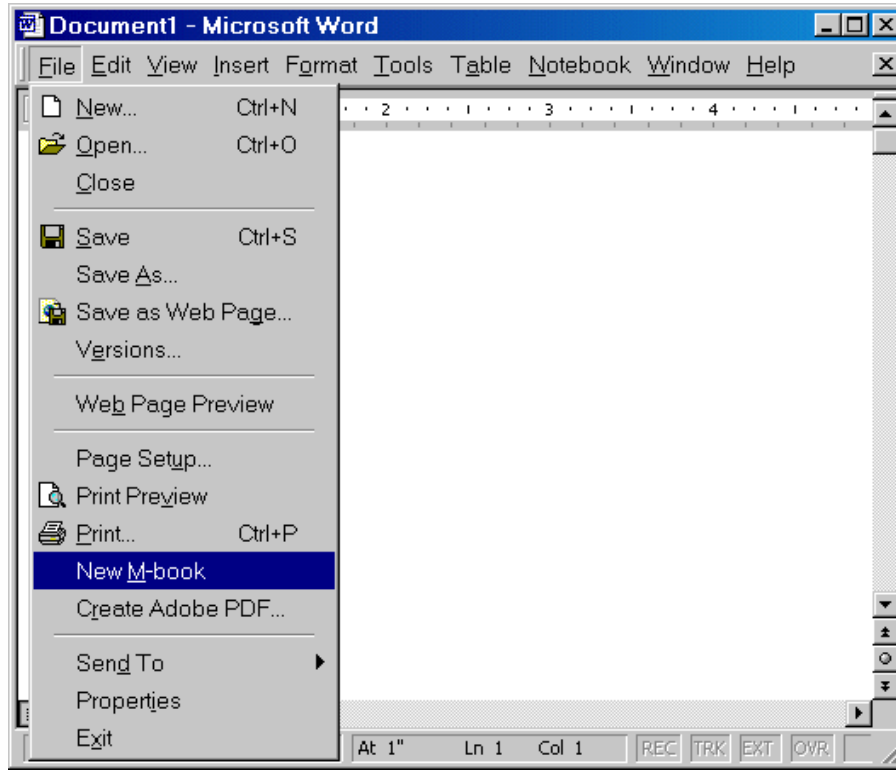
Opening an Existing M-Book

You can use the `notebook` command to open an existing M-book, as shown in the following code, where *`filename`* is the M-book you want to open.

```
notebook filename
```

Alternatively, you can double-click an M-book file in a Windows file management tool, such as Explorer.

When you double-click 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, as shown in the figure that follows.



Converting a Word Document to an M-Book

To convert a Word document to an M-book, follow the steps provided in one of the following sections, depending on which version of Word you are using:

- "Microsoft Word 2002, or 2003" on page 11-8
- "Microsoft Word 2007" on page 11-8

Microsoft Word 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**.

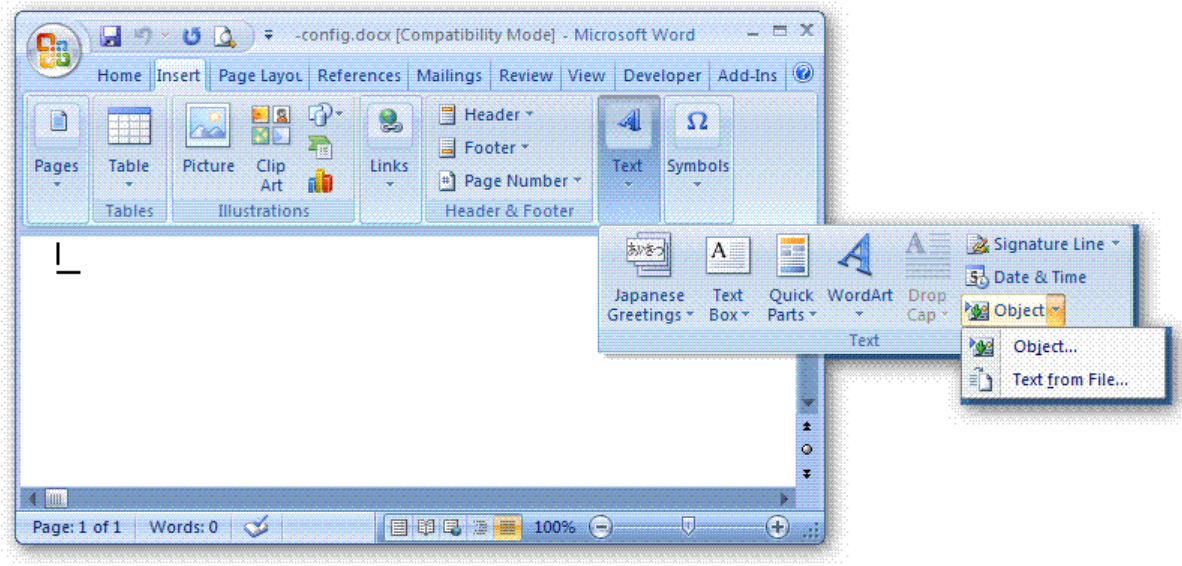
Microsoft Word 2007.

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` folder.

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 follow the steps described in these sections:

- "Defining MATLAB Commands as Input Cells for Notebook" on page 11-11
- "Evaluating MATLAB Commands with Notebook" on page 11-16

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 can also use calc zones to isolate related commands in a section of the M-book, and 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 the Editor 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 using 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...

For information about evaluating the input cells you define, see "Evaluating MATLAB Commands with Notebook" on page 11-16.

Defining Commands as Input Cells for Notebook

To define a MATLAB command in a Word document as an input cell, follow these steps:

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 11-22.

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 that 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 11-17 for information about evaluating a cell group. For information about ungrouping a cell group, see "Ungroup Cells" on page 11-36.

Creating a Cell Group for Notebook

To create a cell group, follow these steps:

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 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 one of the following 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 11-16 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 that you want to include in the calc zone, 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 11-19 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, follow these steps:

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

11-22. 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

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 11-22.

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), as follows:

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 11-12.

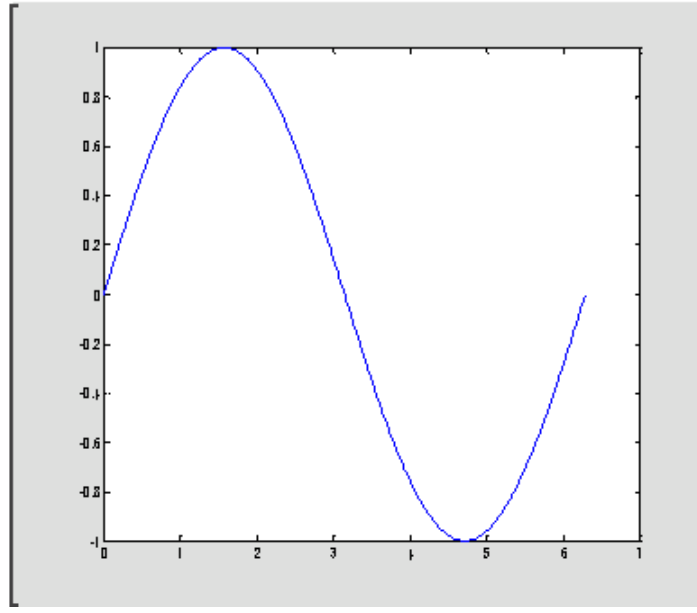
When MATLAB evaluates a cell group, the output for all commands in the group appears in a single output cell. By default, Notebook places the output cell immediately after the cell group the first time the cell group is evaluated. If you evaluate a cell group with an existing output cell, Notebook places the results in the output cell wherever the output cell 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 following 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, follow these steps:

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, follow these steps:

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 11-13.

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.

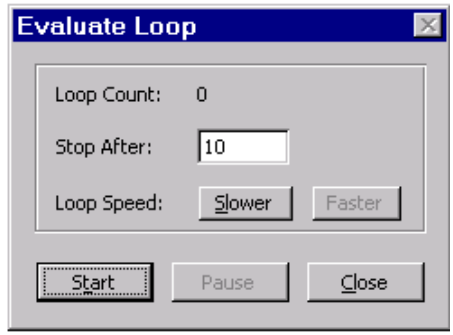
2 Select the **Stop evaluating on error** check box and click **OK**.

Using a Loop to Evaluate Input Cells Repeatedly with Notebook

To evaluate a sequence of MATLAB commands repeatedly, follow these steps:

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**.

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, follow these steps:

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, follow these steps:

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

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 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 that 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, as follows:

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, follow these steps:

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

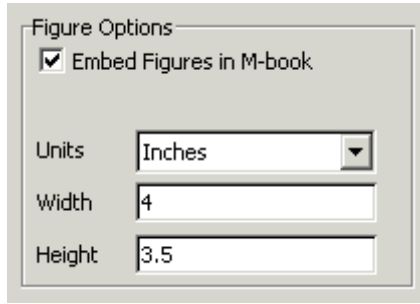
This section describes how to control several aspects of the graphic output produced by MATLAB commands in an M-book, including

Embedding Graphic Output in the M-Book

By default, graphic output is embedded in an M-book. To display graphic output in a separate figure window, follow these steps:

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`, Notebook directs all graphic output 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, follow these steps:

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.

Adjusting Graphic Output in Notebook

To set the default size of embedded graphics in an M-book, follow these steps:

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.

To crop graphic output, or add white space around it, follow the instructions for performing these tasks in Microsoft Word. See the Microsoft Word help for details.

Configuring Notebook

After you install Notebook, but before you begin using it, you must specify that Word can use the Notebook macros, and then configure Notebook. (Notebook is installed as part of the MATLAB installation process on Microsoft Windows platforms. For more information, see the MATLAB installation documentation for your platform.)

To specify that Word can use the Notebook macros:

- In Word 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:

1. Type `notebook` in the MATLAB Command Window.

   MATLAB opens a dialog box that indicates Notebook has not been configured and asks if you want to configure it now.

2. Click **Yes**.

   MATLAB configures Notebook and issues the following messages in the Command Window:

```
Welcome to the utility for setting up the MATLAB Notebook
for interfacing MATLAB to Microsoft Word

Setup complete
Warning: MATLAB is now an automation server
```

When MATLAB configures Notebook, it accesses the Microsoft Windows system registry to locate Microsoft Word and the Word templates folder, and to identify the version of Word. MATLAB then copies Notebook's `m-book.dot` template to the Word templates folder. MATLAB Notebook supports Word versions 2002, 2003, and 2007.

If you have previously configured Notebook, typing `notebook` in the MATLAB Command Window, starts Microsoft Word and creates a new M-book. The Command Window displays the message `Warning: MATLAB is now an automation server`.

If you suspect a problem with the current configuration, you can explicitly configure notebook by typing:

```
notebook ('-setup')
```

Notebook Feature Reference

In this section...

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 11-13.

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 11-19. 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 11-11. For information about evaluating input cells, see "Evaluating MATLAB Commands with Notebook" on page 11-16.

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 11-19.

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 11-16. For information about evaluating the entire M-book, see "Evaluating an Entire M-Book" on page 11-19.

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 11-20.

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 11-19.

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 11-12. For information about converting a cell group to individual input cells, see "Ungroup Cells" on page 11-36.

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 11-22 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 11-21.

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 11-24 and "Suppressing Graphic Output for Individual Input Cells in Notebook" on page 11-25 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 11-22. For information about deleting output cells, see "Purge Selected Output Cells" on page 11-35.

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 "Defining Cell Groups for Notebook" on page 11-12.

12

Source Control Interface

The source control interface provides access to your source control system from the MATLAB desktop. Source control systems, also known as version control, revision control, configuration management, and file management systems, are platform dependent—the topics for the Microsoft Windows platforms appear first, followed by the topics for the UNIX platforms.

Source Control Interface on Microsoft Windows

If you use source control systems to manage your files, you can interface with the systems to perform source control actions from within the MATLAB, Simulink, and Stateflow® products. Use menu items in the MATLAB, Simulink, or Stateflow products, 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 the MATLAB, Simulink, and Stateflow products.

This documentation uses the Microsoft® Visual SourceSafe® software as an example. Your source control system might use different terminology and not support the same options or might use them in a different way. Regardless, you should be able to perform similar actions with your source control system based on this documentation.

Perform most source control interface actions from the Current Folder browser. You can also perform many of these actions for a single file from the MATLAB Editor, a Simulink model window, or a Stateflow chart window—for more information, see “Performing Source Control Actions from the Editor, Simulink, or Stateflow File Menu on Microsoft Windows” on page 12-23. Another way to access many of the source control actions is with the `verctrl` function.

Setting Up the Source Control Interface on Microsoft Windows

In this section...

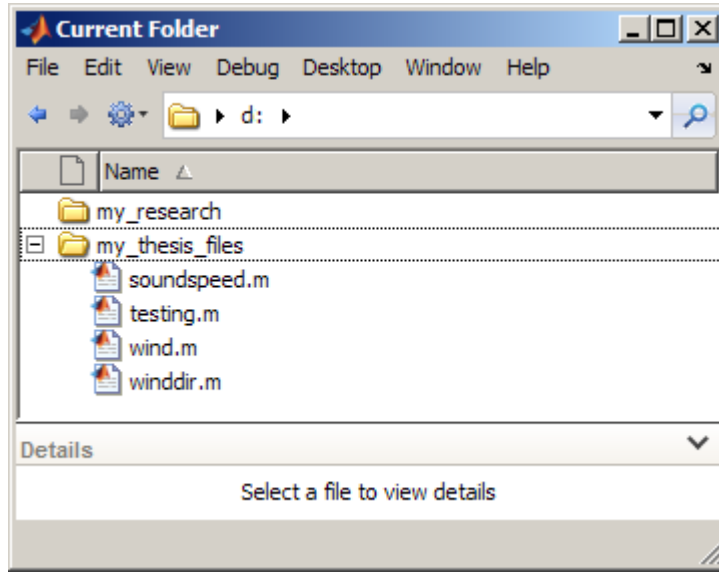
Create Projects in Source Control System

In your source control system, create the projects that your folders and files will be associated with.

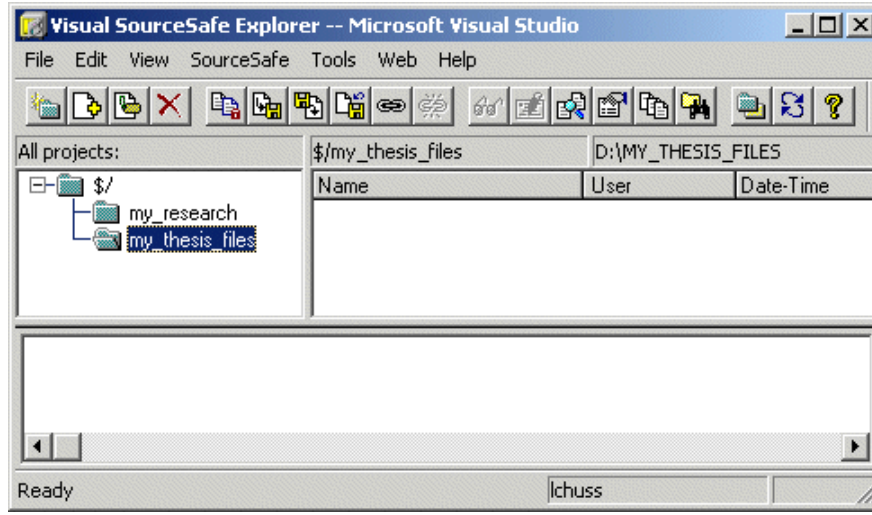
All files in a folder must belong to the same source control project. Be sure the working folder for the project in the source control system specifies the correct path to the folder on disk.

Example of Creating Source Control Project

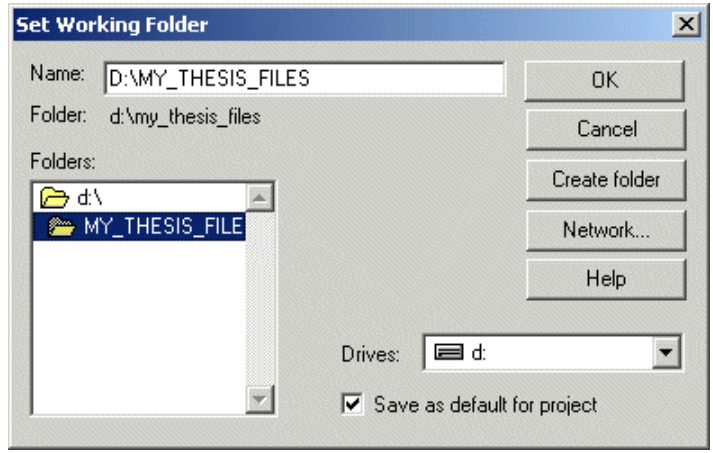
This example uses the project `my_thesis_files` in Microsoft Visual SourceSafe. This illustration of the Current Folder browser shows the path to the folder on disk, `C:\my_thesis_files`.



The following illustration shows the example project in the source control system.



To set the working folder 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.

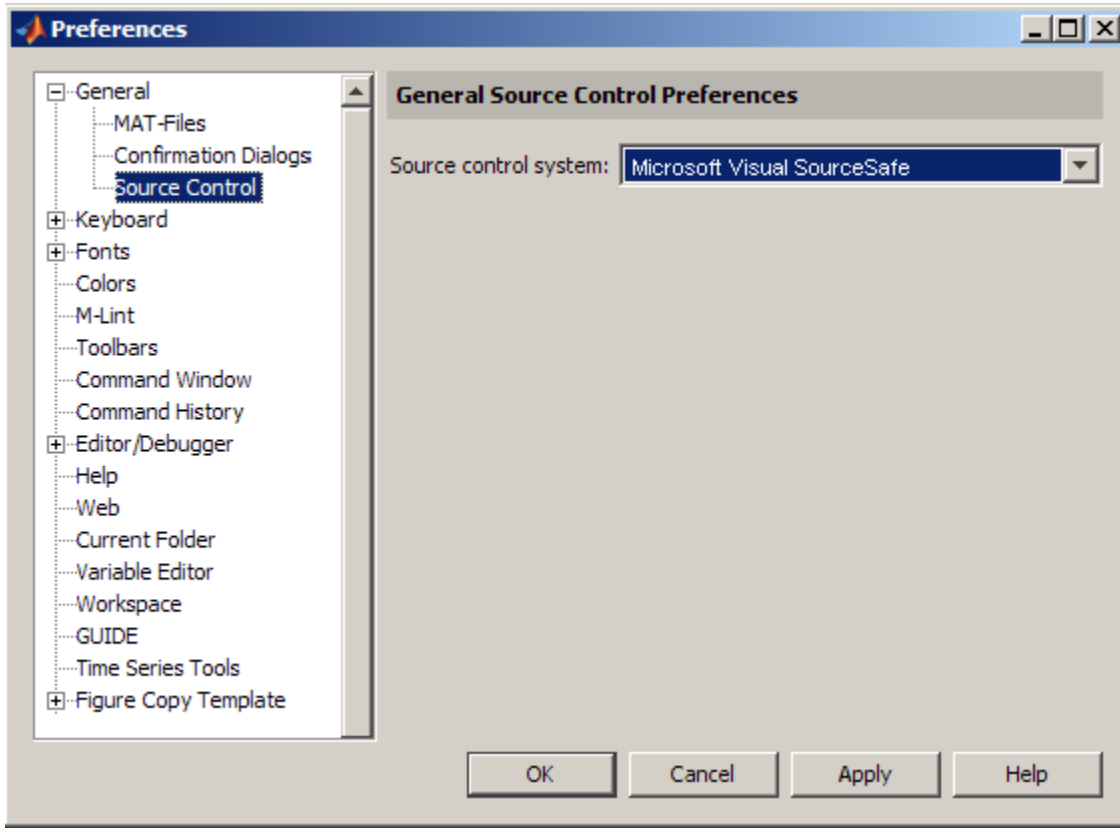


Specify Source Control System with MATLAB Software

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.

Source Control with 64-Bit Versions of MATLAB

If you run a 64-bit version of MATLAB and want MATLAB to interface with your source control system, your source control system must be 64-bit compliant. If you have a 32-bit source control system, or if you have a 64-bit source control system running in 32-bit compatibility mode,MATLAB cannot use it. In that event, MATLAB displays a warning about the problem in the Source Control preference pane.

Function Alternative for Specifying Source Control System

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.

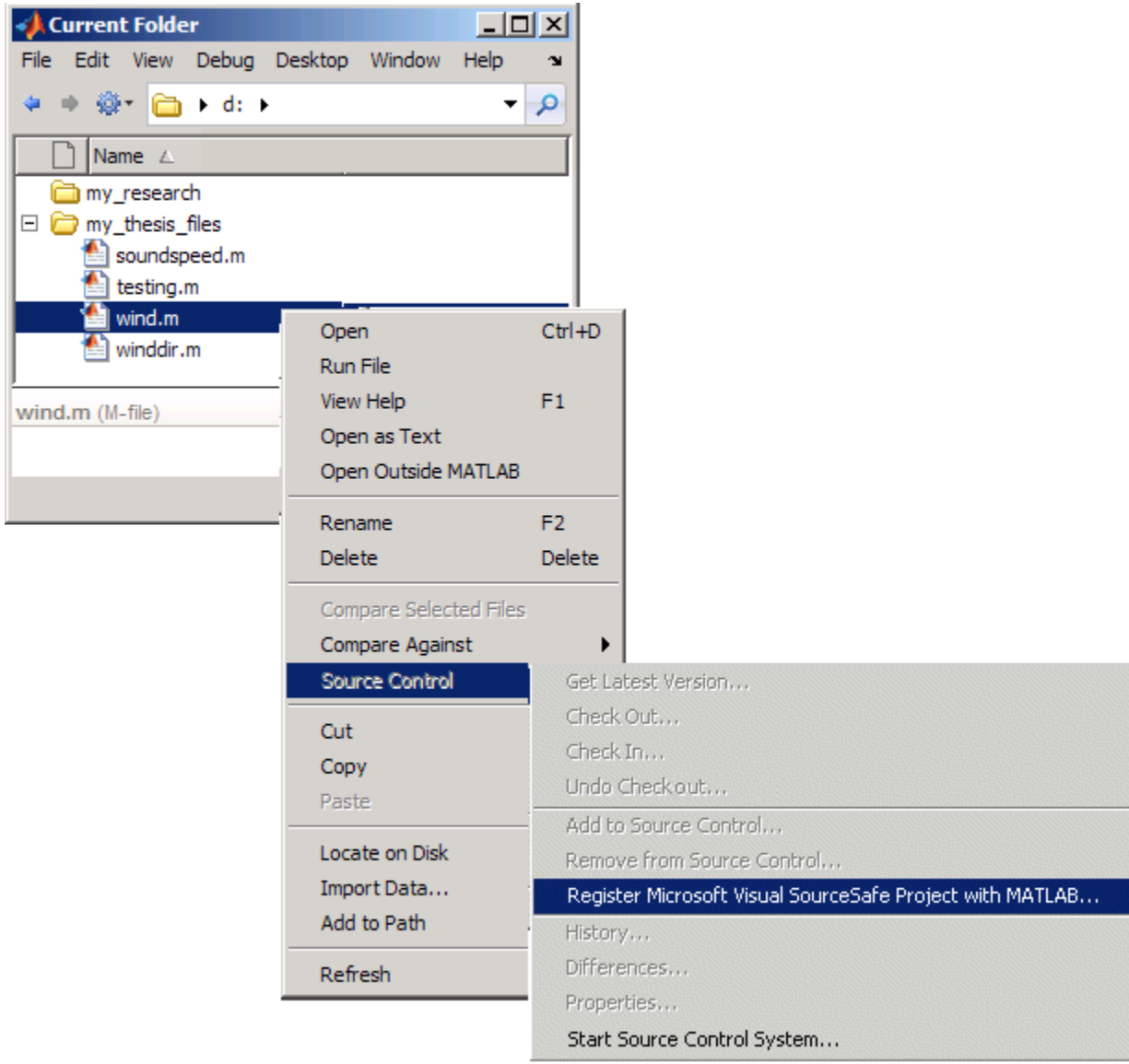
Register Source Control Project with MATLAB Software

Register a source control system project with a folder in MATLAB, that is, associate a source control system project with a folder and all files in that folder. Do this only one time for any file in the folder, which registers all files in that folder:

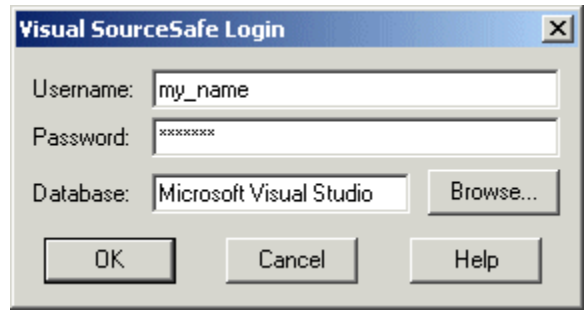
1. In the MATLAB Current Folder browser, select a file that is in the folder you want to associate with a project in your source control system. For example, select `D:\my_thesis_files\wind.m`. This will associate all files in the `my_thesis_files` folder.

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 with MATLAB Software" on page 12-5.

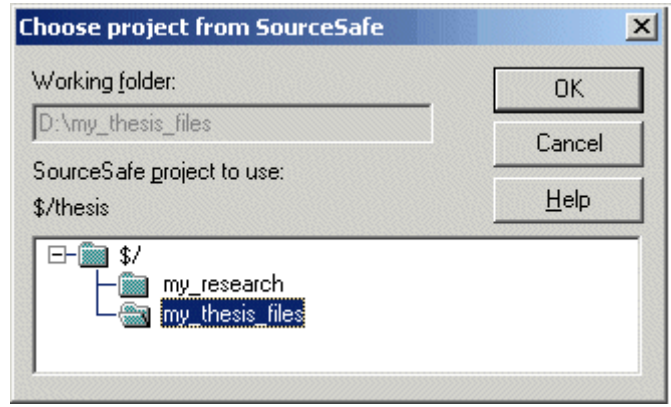
The following example shows Microsoft Visual SourceSafe.



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**.



4 In the resulting **Choose project from Name_of_Source_Control_System** dialog box, select the source control system project to associate with the folder and click **OK**. This example shows `my_thesis_files`.

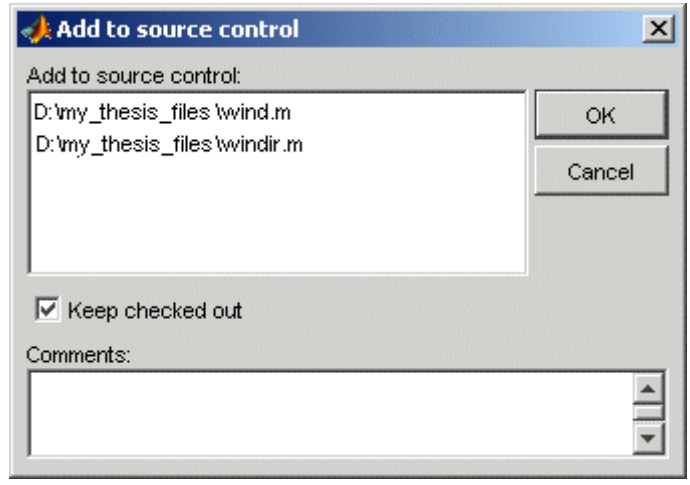


The selected file, its folder, and all files in the folder, are associated with the source control system project you selected. For the example, MATLAB associates all files in `D:\my_thesis_files` with the source control project `my_thesis_files`.

Add Files to Source Control

Add files to the source control system. Do this only once for each file:

1. In the Current Folder 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 the MATLAB Desktop on Microsoft Windows

In this section...

Before checking files into and out of your source control system from the MATLAB desktop, be sure to set up your system for use with MATLAB as described in "Setting Up the Source Control Interface on Microsoft Windows" on page 12-3.

Check Files Into Source Control

After creating or modifying files using MATLAB software or related products, check the files into the source control system by performing these steps:

1. In the Current Folder 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.

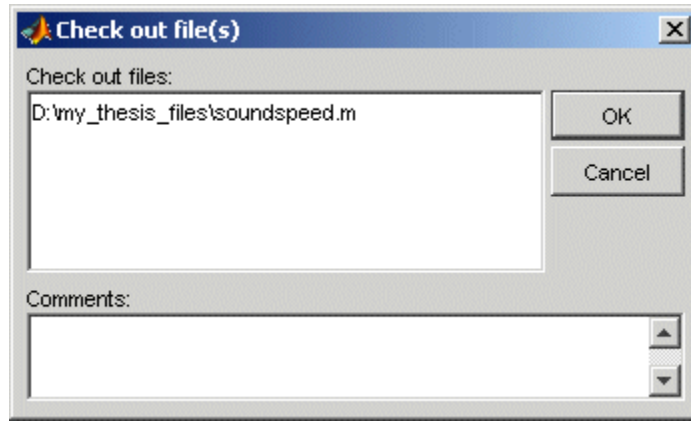
Function Alternative

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 Folder 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.

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 folder from your source control system.

Function Alternative

The function alternative is `verctrl` with the `checkout` argument.

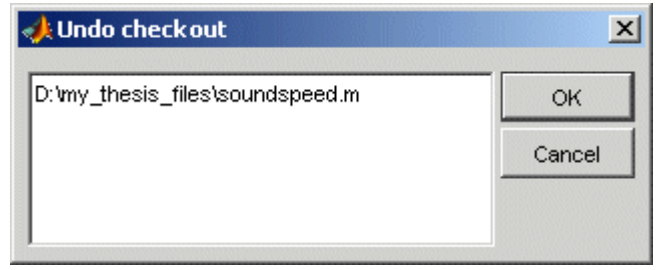
Undoing the Checkout

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 file name before you undo the checkout.

To undo a checkout, follow these steps:

1. In the MATLAB Current Folder 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 Microsoft Windows

In this section...

Getting the Latest Version of Files for Viewing or Compiling

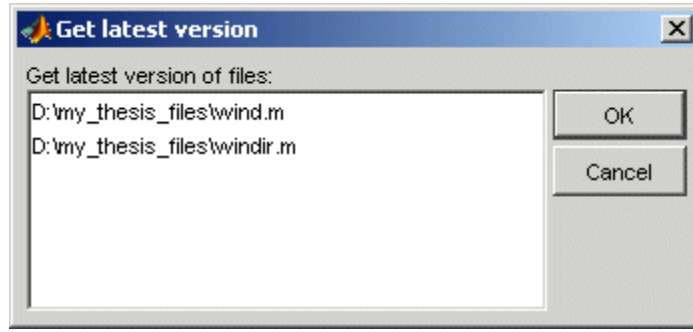
You can get the latest version of a file from the source control system for viewing or running. Getting a file differs from checking it out. When you get a file, it is write protected so you cannot edit it, but when you check out a file, you can edit it.

To get the latest version, follow these steps:

1 In the MATLABCurrent Folder browser, select the folders or files that you want to get. If you select files, you cannot select folders too.

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 folders 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.

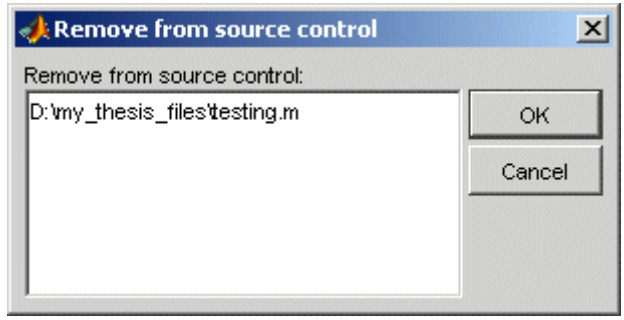
Removing Files from the Source Control System

To remove files from the source control system, follow these steps:

1 In the MATLAB Current Folder 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.

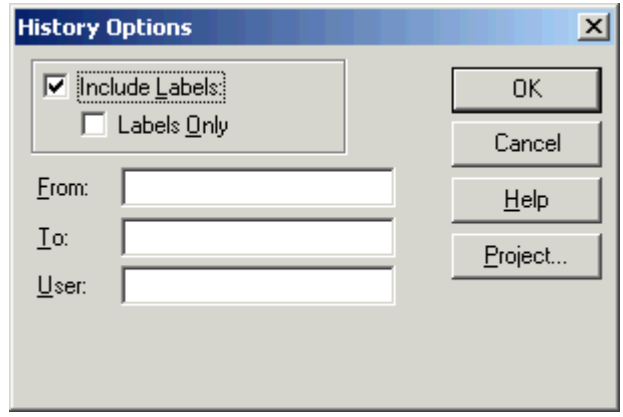
Showing File History

To show the history of a file in the source control system, follow these steps:

1 In the MATLAB Current Folder browser, select the file for which you want to view the history.

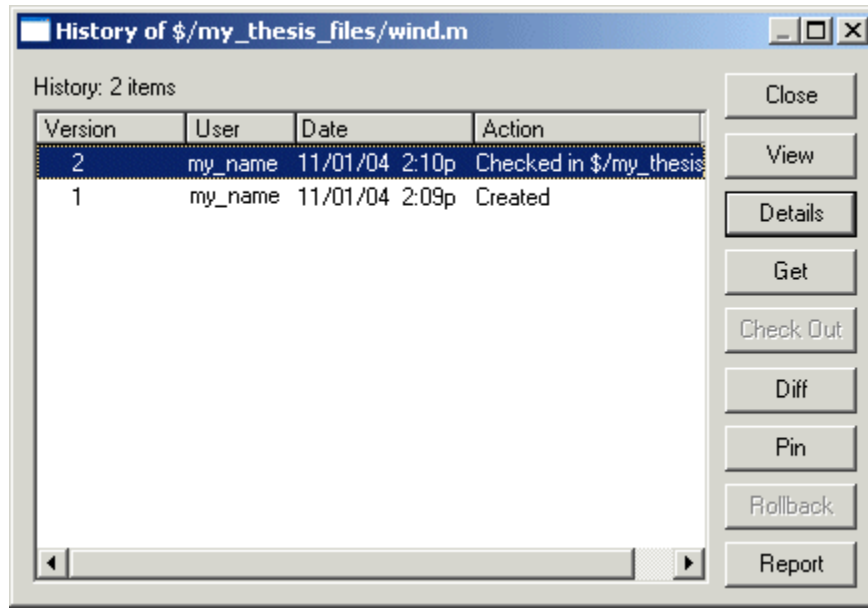
2 Right-click, and from the context menu, select **Source Control > History**.

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.

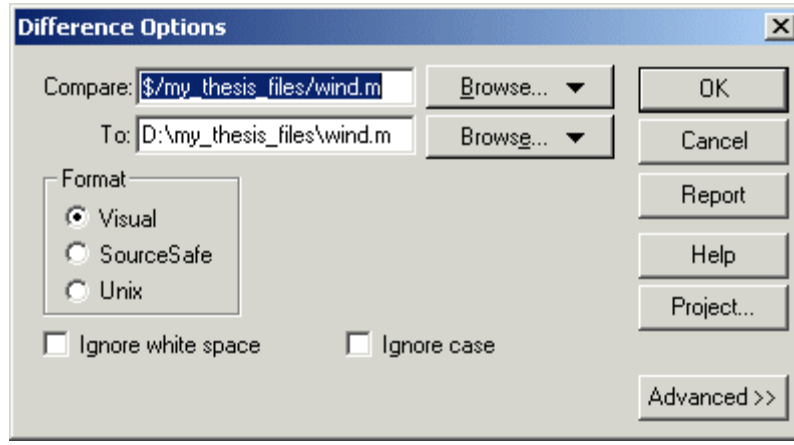
To view the differences, follow these steps:

1. In the MATLAB Current Folder browser, select the file for which you want to view differences. This is a file that has been checked out and edited.

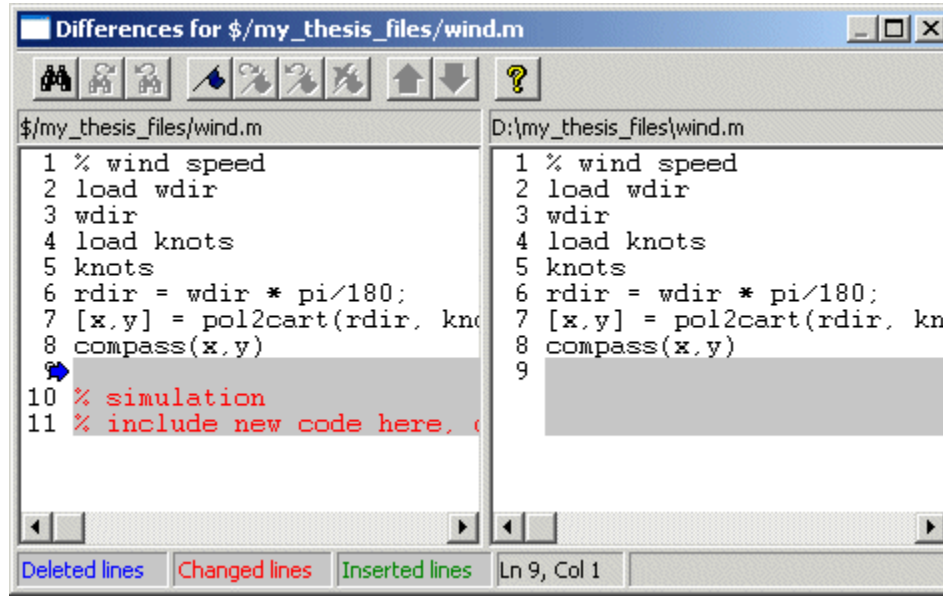
2 Right-click, and from the context menu, select **Source Control > Differences**.

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 SourceSafe.



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.

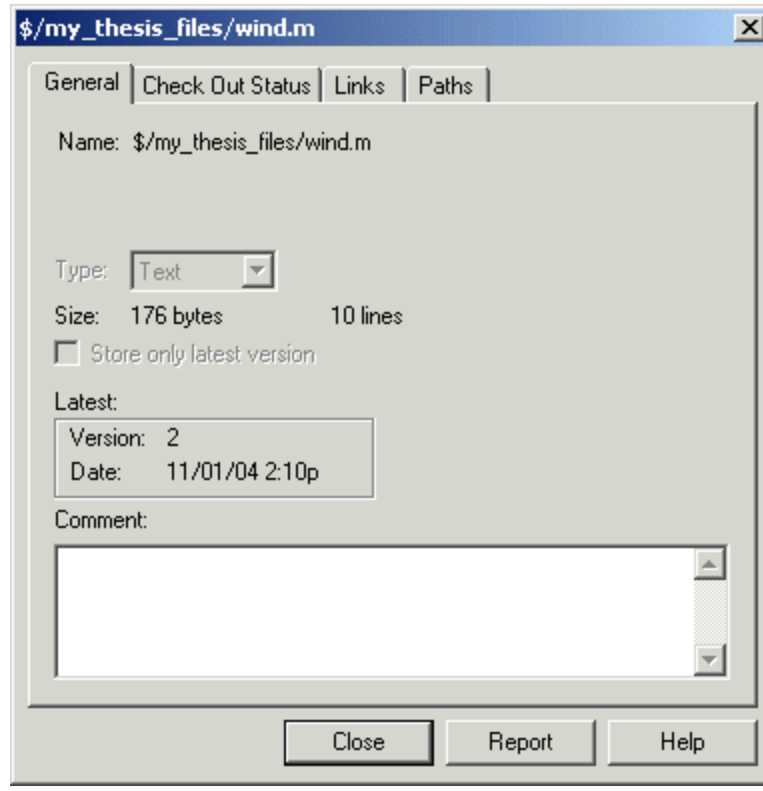
Viewing Source Control Properties of a File

To view the source control properties of a file, follow these steps:

1. In the MATLAB Current Folder 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.

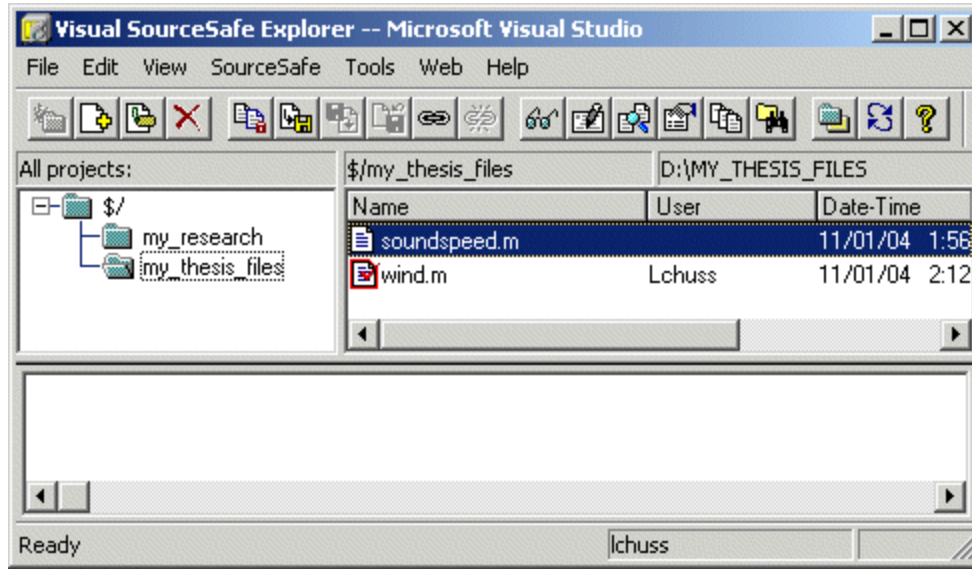
Starting the Source Control System

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 folder or file in the MATLAB Current Folder 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 folder 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, Simulink, or Stateflow File Menu on Microsoft Windows

You can create or open a file in the Editor, the Simulink or Stateflow products and perform most source control actions from their **File > Source Control** menus, rather than from the Current Folder browser. Following are some differences in the source control interface process when you use the Editor, 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 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 the Simulink or Stateflow products. Instead, register a project using the Current Folder browser, as described in "Register Source Control Project with MATLAB Software" on page 12-7.

Troubleshooting Source Control Problems on Microsoft Windows

In this section...

Source Control Error: Provider Not Present or Not Installed Properly

In some cases, MATLAB software 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 folders that include the `@` character in the folder or file name.

You might be able to work around this restriction by quoting nonstandard characters in file names, 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 the Simulink or Stateflow products, 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 Folder browser, as described in "Register Source Control Project with MATLAB Software" on page 12-7. You can then perform source control actions for all files in that project (folder).

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 Platforms

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 the MATLAB, Simulink, and Stateflow products.

The source control interface supports four popular source control systems, as well as a custom option:

- ClearCase® software from IBM® Rational®
- Concurrent Version System (CVS)
- ChangeMan® and PVCS® software from Serena®
- 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, a Simulink model window, or a Stateflow chart window. To perform source control actions on multiple files, use the Current Folder 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 Platforms

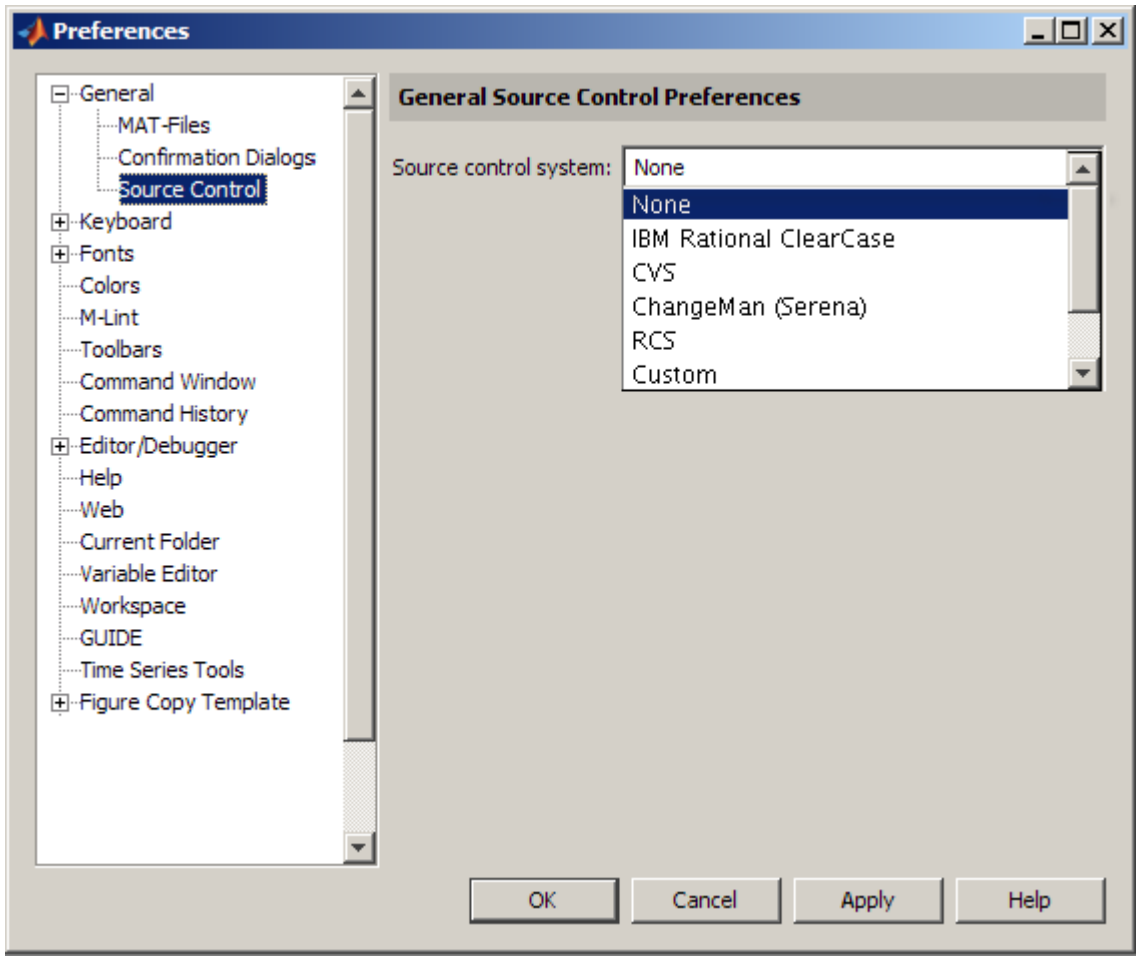
In this section...

MATLAB Desktop Alternative

To 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 with which you want to interface 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 Folder with ClearCase Software on UNIX Platforms

If you use ClearCase software on a UNIX platform, perform the following from ClearCase:

1. Set a view.
2. Check out the folder 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 software.

Checking Files Into the Source Control System on UNIX Platforms

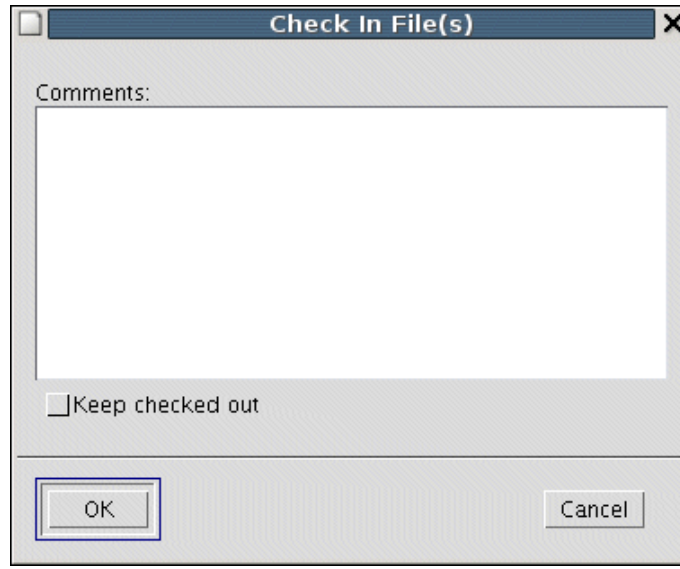
In this section...

Checking In One or More Files Using the Current Folder Browser

1. From the Current Folder 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, or the Simulink or Stateflow Products

1. From the Editor, or the Simulink or Stateflow products, 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`.

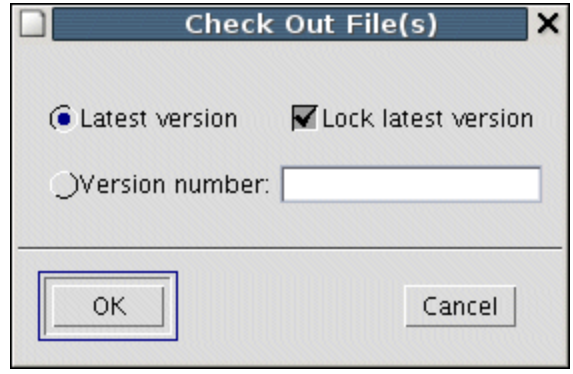
Checking Files Out of the Source Control System on UNIX

In this section...

Checking Out One or More Files Using the Current Folder Browser

1 In the Current Folder 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 using MATLAB software or another software product, and save the files. For example, edit an M-file in the Editor.

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, or the Simulink or Stateflow Products

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 Folder Browser" on page 12-33, 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 Platforms

In this section...

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 file name before you undo the checkout. Undo the checkout using the Current Folder browser for one or more files. For only one file, you can also use the Editor, or the Simulink or Stateflow products.

Undoing the Checkout for One or More Files Using the Current Folder Browser

1. In the MATLAB Current Folder 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, or the Simulink or Stateflow Products

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'})
```

13

Internationalization

How the MATLAB Process Uses Locale Settings

A *locale* is part of the user environment definition. It defines language, territory, and *codeset*, which is a coded character set. The MATLAB process uses the user-specified locale name on all platforms. MATLAB also reads the user-specified *UI language name*, and uses it to select localized resources in the specified language. By using this feature, a user can select localized resources in US-English. The user-specified UI language setting also controls language and country settings of the Sun™ Java Virtual Machine (JVM) software.

Consider the following when choosing your locale settings. To see what your current settings are, use the instructions in "Setting Locale on Windows Platforms" on page 13-4, "Setting Locale on Linux and Solaris Platforms" on page 13-5, or "Setting Locale on Macintosh Platforms" on page 13-6..

- **`Default Locale Setting`** — If the user-specified locale is not supported, MATLAB uses the default locale `en_US.US-ASCII`.
- **`UI Language Setting`** — The UI language setting should be set to either the same language as the user-specified locale or to `US-English`. Otherwise, non-7-bit ASCII characters might not display properly.
- **`Supported Encoding Scheme`** — MATLAB might not properly handle character codes greater than 2 bytes.
- **`Supported Character Set`** — MATLAB supports the character set specified by the user locale setting.
- **`M-File Compatibility`** — Non-7-Bit ASCII characters in M-files created on one platform might not be compatible on other platforms using different locale settings.
- **`Platform-Specific Localized Formats`** — MATLAB usually uses platform-neutral localized formats and rules.
- **`On Windows Platforms`** — User locale and system locale must be the same value on the Microsoft Windows platform. If these values are not the same, users might see garbled text or incorrect characters. For information on controlling these settings, see "Setting Locale on Windows Platforms" on page 13-4.

- **`On Macintosh Platforms`** — MATLAB automatically chooses a codeset for each combination of language and territory on the Apple Macintosh OS X platform. If you customize the locale setting on OS X, MATLAB ignores the customized portion. In Version 10.5 of the OS X operating system, MATLAB ignores the `LANG` environment variable.
- **`Running nodesktop Option On Macintosh OS X Version 10.5 Platforms`** — When you run MATLAB software with the `-nodesktop` startup option on the Macintosh OS X Version 10.5 platform, the MATLAB locale setting is not the Macintosh locale setting for the Terminal application. For example, for users selecting the `Japanese_Japan` region on the **Formats** tab, the MATLAB locale setting is `ja_JP.sjis`. The Macintosh locale setting is `ja_JP.UTF-8`.

Setting the Locale

In this section...

Setting Locale on Windows Platforms

MATLAB software uses the *system locale* and *user locale* on Windows platforms:

Setting User Locale on Windows Vista Platforms

1. Select **Start -> Control Panel -> Regional and Language Options**.
2. Open **Formats** tab.
3. Select an item from the drop-down list.

Setting System Locale on Windows Vista Platforms

1. Select **Start -> Control Panel -> Regional and Language Options**.
2. Open **Administrative** tab.
3. Click **Change system locale...** button.
4. Select an item from the drop-down list.
5. Reboot the system.

Note When you change the system locale, you must reboot your system; otherwise, you might see unexpected locale-setting behaviors.

Setting User Locale on Windows XP Platforms

1. Select **Start** -> **Control Panel** -> **Regional and Language Options**.
2. Open **Regional Options** tab.
3. Select an item from the drop-down list.

Setting System Locale on Windows XP Platforms

1. Select **Start** -> **Control Panel** -> **Regional and Language Options**.
2. Open **Advanced** tab.
3. Select an item from the drop-down list.
4. Reboot the system.

Note When you change the system locale, you must reboot your system; otherwise, you might see unexpected locale-setting behaviors.

Setting Locale on Linux and Solaris Platforms

Linux[16] and Sun Solaris™ platforms manage locale settings with six *locale categories*. These are the same categories used by C standard library functions.

The following locale categories are available:

- `LC_CTYPE` controls character data manipulations.
- `LC_COLLATE` controls character collation/sorting operations.

16. Linux is a registered trademark of Linus Torvalds.

- LC_TIME controls date/time data formatting or parsing.
- LC_NUMERIC controls numeric data formatting or parsing.
- LC_MONETARY controls monetary data formatting or parsing.
- LC_MESSAGES controls the user UI language.

Setting User Locale and User UI Language

Use the LANG environment variable to specify a single locale for all locale categories. The locale specified with this variable might be partially or entirely over-written by other environment variables.

Use the environment variables LC_CTYPE, LC_COLLATE, LC_TIME, LC_NUMERIC, and LC_MONETARY to specify a locale for a particular category.

Use the LC_ALL environment variable to over-write all locales specified with other environment variables. If a single locale has to be set to all locale categories, use LANG instead of LC_ALL.

Setting Locale on Macintosh Platforms

The Macintosh OS X platform manages the user locale setting and the user UI language setting.

Setting User Locale

1. Select **System Preferences ->International**
2. Open **Formats** tab
3. Select an item from the **Region** pop-up menu

Setting UI Language

1. Select **System Preferences ->International**
2. Open **Language** tab
3. Drag an item to the top of the **Languages** list

Calculating Dates in Programs

To ensure the correct calculation of functions using date values, replace `datenum` function calls with the use of the `dir` function `datenum` field.

For example, look at the modification date of your MATLAB `license.txt` file:

```
cd(matlabroot)
f=dir('license.txt')
```

MATLAB displays information similar to:

```
f =
       name: 'license.txt'
       date: '10-May-2007 17:48:22'
      bytes: 5124
      isdir: 0
    datenum: 7.3317e+005
```

If your code uses a command similar to:

```
n=datenum(f.date);
```

you must replace it with:

```
n=f.datenum;
```

Numeric Format Uses C Locale

MATLAB software reads the user locale for all categories except for the `LC_NUMERIC` category. This category controls numeric data formatting and parsing. MATLAB always sets `LC_NUMERIC` to the `C` locale.

For example, some users expect a comma in a number while other users expect a decimal. The value of pi can be displayed as `3.1415` or `3,1415`, depending on the format used by a locale. MATLAB always uses `3.1415`, regardless of the format specified by the user locale.

Troubleshooting I18n Messages

The term *I18n* is an abbreviation for `internationalization`, where `18` stands for the number of letters between the `i` and the `n`.

MATLAB:I18n:InconsistentLocale Warning

The warning message:

```
MATLAB:I18n:InconsistentLocale - The system locale
setting, system_locale_name,
is different from the user locale setting, user_locale_name.
```

indicates a mismatch between locale setting on Microsoft Windows systems. This may affect your ability to display certain characters. For information about changing the locale settings, see "Setting Locale on Windows Platforms" on page 13-4.

Index

Symbols and Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y