

MATLAB®

Desktop Tools and Development Environment

R2012a

MATLAB®

How to Contact 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

© COPYRIGHT 1984–2012 by The MathWorks, Inc.

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

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

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

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 7.9 (Release 2009b)
March 2010	Online only	Revised for MATLAB 7.10 (Release 2010a)
September 2010	Online only	Revised for MATLAB Version 7.11 (Release 2010b)
April 2011	Online only	Revised for MATLAB Version 7.12 (Release 2011a)
September 2011	Online only	Revised for MATLAB Version 7.13 (Release 2011b)
March 2012	Online only	Revised for MATLAB Version 7.14 (Release 2012a)

Startup and Shutdown

1

Starting and Quitting the MATLAB Program	1-2
Starting the MATLAB Program on Windows Platforms ..	1-2
Starting the MATLAB Program on Linux Platforms	1-6
Starting the MATLAB Program on Macintosh Platforms ..	1-6
Quitting the MATLAB Program	1-7
Startup Folder for the MATLAB Program	1-12
What Is the Startup Folder?	1-12
Startup Folder on Windows Platforms	1-13
Startup Folder on Linux Platforms	1-14
Startup Folder on Macintosh Platforms	1-14
Changing the Startup Folder	1-14
Startup Options	1-18
Specifying MATLAB Startup Options	1-18
Commonly Used Startup Options	1-20
Passing Perl Variables on Startup	1-21
Startup and Calling Java Software from the MATLAB Program	1-22
Toolbox Path Caching in the MATLAB Program	1-23
About Toolbox Path Caching in the MATLAB Program ...	1-23
Using the Cache File Upon Startup	1-23
Updating the Cache and Cache File	1-23
Additional Diagnostics with Toolbox Path Caching	1-26

Desktop

2

Adjust Desktop Appearance	2-2
Fonts	2-2

Color Settings	2-8
Toolbar Customization	2-11
Columns — Resize, Rearrange, and Sort in Desktop Tools	2-12
Enable and Disable Confirmation Dialog Boxes	2-13
Open and Rearrange Desktop Tools and Documents	2-13
Keyboard Shortcuts	2-36
Keyboard Shortcuts	2-36
Choose a Set of Keyboard Shortcuts	2-40
Compare Sets of Keyboard Shortcuts	2-43
Display Keyboard Shortcuts	2-45
Customize Keyboard Shortcuts	2-48
Evaluate and Resolve Keyboard Shortcut Conflicts	2-54
Examples of Creating, Modifying, and Deleting Keyboard Shortcuts	2-56
Delete a Set of Keyboard Shortcuts	2-59
Use Keyboard Shortcuts Settings Files Created on Other Systems	2-60
Keyboard Shortcut Restrictions	2-60
Basic Operations	2-63
Cut, Copy, and Paste	2-63
Print Options	2-64
Select Multiple Items	2-66
Add Your Own Toolboxes to the Start Button	2-67
Web Browsers and MATLAB	2-71
About Web Browsers and MATLAB	2-71
Display Pages in Web Browsers	2-73
Specify Proxy Server Settings for Connecting to the Internet	2-73
Specify the System Browser for UNIX Platforms	2-75
License Management and Software Updates	2-76
Manage Your Licenses	2-76
Check for Software Updates	2-77
Macintosh Platform Conventions	2-79
Mouse Instructions and Macintosh Platforms	2-79
Navigating Within the MATLAB Root Folder on Macintosh Platforms	2-79

Accessibility	2-81
Software Accessibility Support	2-81
Documentation Accessibility Support	2-82
Assistive Technologies	2-83
Installation Notes for Accessibility Support	2-84
Troubleshooting	2-87
About Preferences	2-93
Set Preferences for MATLAB	2-93
Where MATLAB Stores Preferences	2-94
Preferences Folder and Files MATLAB Uses When Multiple MATLAB Releases Are Installed	2-95
General Preferences	2-96
MAT-Files Preferences	2-97
Confirmation Dialogs Preferences	2-98
Source Control Preferences	2-100
Java Heap Memory Preferences	2-100
Keyboard Shortcuts Preferences	2-101
Fonts Preferences	2-103
Fonts Custom Preferences	2-104
Colors Preferences	2-105
Colors Programming Tools Preferences	2-106
Toolbars Preferences	2-107
Web Preferences	2-108

Running Functions — Command Window and History

3

Command Window	3-3
Command History Window	3-4
Command History File — history.m	3-4
Open Command Window, Create Variables, and Run Code	3-6
Open the Command Window	3-6
Create Variables and Run Functions	3-6
Find Functions Using the Function Browser	3-7

Enter Statements in the Command Window	3-13
Stop Execution	3-14
Run External Commands, Scripts, and Programs	3-15
Keep a Session Log	3-18
Evaluate or Open Code You Select	3-18
Display Hyperlinks in the Command Window	3-18
Create MATLAB Shortcuts to Rerun MATLAB	
Commands	3-23
What Is a MATLAB Shortcut?	3-23
When to Use MATLAB Shortcuts	3-23
MATLAB Shortcut Creation	3-24
Run MATLAB Shortcuts	3-26
Edit and Organize MATLAB Shortcuts	3-26
MATLAB Toolbar Shortcuts — Hide, Show, and Delete ..	3-27
Open Command History Window	3-29
Create Files from, Run, and Copy Command History	
Commands	3-30
Adjust Command Window Appearance	3-32
Show or Hide Links to Videos, Demos, and Getting Started	3-32
Show or Hide Function Browser Button	3-32
Set the Number of Lines in the Scroll Buffer	3-33
Adjust Command History Window Appearance	3-34
Delete Entries from the Command History Window	3-34
Change the Date Format in the Command History Window	3-35
Command Window Code and Output Display	3-36
Keep or Suppress Blank Lines in Output	3-36
Format Numerals	3-37
Wrap Lines of Code to Fit Window Width	3-38
Matrix Display Width	3-38
Number of Spaces Assigned to Tab Key	3-39
Suppress Code Output	3-39
Page Code Output	3-39
Clear the Command Window	3-40

Avoid Mistakes When Entering Code	3-41
Highlight Syntax to Help Ensure Correct Entries in the Command Window	3-41
Avoid Mismatched Parentheses, Brackets, Braces, and Paired Keywords	3-42
Complete Names in the Command Window Using the Tab Key	3-43
View Function Syntax Hints While Entering a Statement	3-49
Find Text in the Command Window	3-53
Find Text Currently Displayed in the Command Window	3-53
Increase the Amount of Information Available for Searching in the Command Window	3-53
Use Incremental Search in the Command Window	3-54
Find Text in the Command History Window	3-61
Quick Search for Entries Beginning with Specified Letters or Numbers	3-61
Search for Entries Containing Partial Words, Whole Word, or Matching Case	3-61
Command Window Preferences	3-63
Keyboard Preferences	3-65
Command History Preferences	3-67

Help and Product Information

4

Ways to Get Function Help	4-2
Run Examples and Demos	4-3
Search Syntax and Tips	4-5

Adjust Help Browser Fonts	4-7
Bookmark and Share Page Locations	4-9
Contact Technical Support	4-11
Help Preferences	4-13
Documentation on Japanese Systems	4-15
Information About your Installation	4-16

Workspace Browser and Variable Editor

5

MATLAB Workspace Browser	5-2
MATLAB Workspace and Workspace Browser	5-2
Viewing and Editing Values in the Workspace	5-3
Improving Workspace Browser Performance	5-6
Saving the Current Workspace	5-6
Viewing a Saved Workspace	5-7
Loading a Saved Workspace and Importing Data	5-7
Changing and Copying Variable Names in the Workspace	5-8
Deleting Workspace Variables	5-8
Creating Plots from the Workspace Browser	5-9
Opening Variables and Objects for Viewing and Editing ..	5-20
Viewing and Editing Workspace Variables with the Variable Editor	5-22
About the Variable Editor	5-22
Opening the Variable Editor	5-22
Working with Different Data Types in the Variable Editor	5-25
Navigating the Variable Editor Using Shortcut Keys	5-32
Changing Size, Content, and Format of Variables in the Variable Editor	5-33
Selecting and Modifying Values in the Variable Editor ...	5-34

Exchanging Variable Editor Data with Other Tools and Applications	5-39
Creating Graphs and Variables, and Data Brushing in the Variable Editor	5-40
Workspace Browser and Variable Editor	
Preferences	5-45
Workspace Browser Preferences	5-45
Variable Editor Preferences	5-46

Managing Files in MATLAB

6

Introduction to Managing Files in MATLAB	6-2
Ways to Manage MATLAB Files	6-2
Tools for Managing Files	6-2
 Understanding File Locations in MATLAB	6-4
Important MATLAB Folders	6-4
Path Names in MATLAB	6-7
 Working with Files and Folders	6-14
Viewing Folder Contents	6-14
Using the Current Folder Browser	6-20
 Finding Files and Folders	6-28
Finding Files and Folders by Name in the Current Folder	6-28
Simple Search for File and Folder Names in the Current Folder Browser	6-28
Advanced Search for Files — Find Files Tool	6-29
Locating a File or Folder in the Operating System Browser	6-33
Finding Files and Folders Using Functions	6-34
Additional Ways to Find Files	6-34
 Creating, Opening, Changing, and Deleting Files and Folders	6-35

Creating New Files and Folders	6-35
Copying, Renaming, and Deleting Files and Folders	6-40
Opening and Running Files	6-44
Comparing Files and Folders	6-47
Comparing Files and Folders	6-47
Comparing Folders and Zip Files	6-49
Comparing Text Files	6-53
Comparing Files with Autosave Version or Version on Disk	6-58
Comparing MAT-Files	6-58
Comparing Binary Files	6-61
Using Comparison Tool Features	6-61
Function Alternative for Comparing Files and Folders ...	6-63
Making Files and Folders Accessible to MATLAB	6-64
Files and Folders That MATLAB Can Access	6-64
How to Make Files Accessible	6-64
Determining if MATLAB Can Access a File	6-66
Ensuring MATLAB Uses the File You Want	6-67
Using the MATLAB Search Path	6-70
What Is the Search Path?	6-70
Viewing Files and Folders on the Search Path	6-72
Changing the Search Path	6-73
Using the Search Path with Different MATLAB Installations	6-78
Recovering from Problems with the Search Path	6-79
Handling Errors and Unexpected Behavior When Updating Folders	6-81
Related Topics for Managing Files	6-82

File Exchange — Finding and Getting Files Created by Other Users

7

Download Files from File Exchange	7-2
---	-----

About File Exchange	7-2
Steps for Using File Exchange	7-4
Example — Finding and Downloading a File in File Exchange	7-5
Find Files in File Exchange — Searching and Using Tags	7-12
About Finding Files in File Exchange	7-12
Using Search to Find Files in File Exchange	7-12
Finding Files by Product, Author, and Other Attributes in File Exchange	7-13
Using Tags to Find Files in File Exchange	7-13
Clearing Your Criteria	7-25
Getting Better Results Using Search and Tags	7-25
View and Sort the List of Files in File Exchange	7-27
Viewing the List of Files in File Exchange	7-27
Sorting the List of Files in File Exchange	7-28
View Details About a File	7-29
Viewing the File Details Page	7-29
Viewing the Contents of a File	7-29
Get Files from the File Exchange Repository	7-31
About Downloading Files	7-31
Downloading from the List of Files	7-31
Downloading from the File Details Page to a Location You Choose	7-32
The Default Folder for Downloaded Files	7-32
Which Location Should You Choose When Downloading Files?	7-32
Downloading a Submission that Consists of Multiple Files	7-33
Viewing and Locating Files You Downloaded	7-33
Best Practices for Using Files Provided by Other Users	7-36
Ensure MATLAB Can Access the File	7-36
Consult the File Details Page	7-36
Look for Updates to the File	7-36
Read the File	7-37
Ask Questions	7-37

Contribute to the File Exchange Repository	7-38
How You Can Contribute to the Repository	7-38
Adding Tags to a File	7-38
Removing Tags from a File	7-39
Rating a File	7-39
Providing Comments About a File	7-40
Submitting Your Files to the Repository	7-40
Frequently Asked Questions About File Exchange	7-41
What Is File Exchange?	7-41
How Do I Use File Exchange?	7-41
How Does the File Exchange Desktop Tool Relate to File Exchange on the Web Site?	7-42
Why Do I See Only 50 Files and How Can I See More? ...	7-42
What Are Tags and How Do I Use Them?	7-43
What Are the Tags Above the List of Files?	7-43
How Can I See Other Tags?	7-43
Why Are the Tags Changing?	7-44
Is Search Looking Inside Files?	7-44
How Can I Start Over When Looking for Files?	7-44
How Can I Choose Where to Download a File To?	7-45
How Do I Contribute My Files to the Repository?	7-45

Editing and Debugging MATLAB Code

8

Create, Open, Save, and Close Files	8-2
Create New Files	8-2
Open Existing Files	8-2
Save Files	8-4
Close Files	8-7
Close the Editor	8-8
Adjust Editor Appearance	8-9
Line and Column Number Indicators	8-9
Right-Side Text Limit Indicator	8-10
Cursor Location Indicator — Class, Function, or Subfunction	8-11
Display Two Parts of a File Simultaneously	8-11

Editing Code	8-15
Insert Text and Overwrite Existing Text	8-15
Change the Case of Selected Text	8-16
Undo and Redo Editor Actions	8-16
Highlight Current Line	8-16
Indent Code	8-17
Code Folding — Expand and Collapse Code Constructs ...	8-20
Add Comments	8-26
Wrap Comments	8-29
Colors in the MATLAB Editor — What Do They Mean? ..	8-30
Code Contains %#ok — What Does That Mean?	8-33
File Navigation	8-34
Navigate to a Specific Location	8-34
Set Bookmarks	8-38
Navigate Backward and Forward in Files	8-38
Open a File or Variable from Within a File	8-39
Find and Replace Text in Files	8-41
Find Any Text in the Current File	8-41
Find and Replace Functions or Variables in the Current File	8-41
Automatically Rename All Functions or Variables in a File	8-43
Find and Replace Any Text	8-45
Find Text in Multiple File Names or Files	8-45
Function Alternative for Finding Text	8-45
Perform an Incremental Search in the Editor	8-45
Run MATLAB Files in the Editor	8-46
Run Files with No Input Arguments in the Editor	8-46
Run Files with Input Arguments in the Editor	8-46
Create and Use a Run Configuration	8-47
Use a Previously Created Run Configuration	8-48
Create and Execute Multiple Run Configurations for a File	8-49
About the run_configurations.m File	8-52
Find Configurations	8-53
Remove Configurations	8-55
Reassociate and Rename Configurations	8-56
Evaluate Subsections of Files Using Code Cells	8-58

What Are Code Cells?	8-58
Scenarios for Evaluating Sections of Code	8-59
Process for Evaluating Sections of Files	8-60
Define Code Cells	8-61
Nested Code Cells	8-68
Navigate Among Code Cells in a File	8-76
Evaluate Code Cells	8-77
Avoid Mistakes While Editing Code	8-84
Highlight Syntax to Help Ensure Correct Entries in the Editor	8-84
Complete Names in the Editor Using the Tab Key	8-85
Check Code for Errors and Warnings	8-91
Avoid Variable and Function Scoping Problems	8-110
Debugging Process and Features	8-116
Ways to Debug MATLAB Files	8-116
Preparing for Debugging	8-116
Set Breakpoints	8-119
Run a File with Breakpoints	8-123
Step Through a File	8-125
Examine Values	8-127
Correct Problems and End Debugging	8-133
Conditional Breakpoints	8-141
Breakpoints in Anonymous Functions	8-143
Breakpoints in Methods That Overload Functions	8-144
Error Breakpoints	8-145
Debugging Functions	8-150
About MATLAB Code Files	8-151
Editor/Debugger Preferences	8-152
General Preferences for the Editor/Debugger	8-152
Editor/Debugger Display Preferences	8-153
Editor/Debugger Tab Preferences	8-154
Editor/Debugger Language Preferences	8-155
Editor/Debugger Code Folding Preferences	8-158
Editor/Debugger Autosave Preferences	8-159
About Code Analyzer Preferences	8-161

Code Analyzer Preferences	8-161
Searching Messages in the Code Analyzer Preferences	
Dialog Box	8-162

Tuning and Managing MATLAB Code Files

9

Using MATLAB Reports	9-2
Refine and Improve Files Using Reports	9-2
Annotate Files with Reminders and Generate Report	9-3
Generating a Summary View of the Help Components in	
Functions and Scripts	9-7
Displaying and Updating a Report on the Contents of a	
Folder	9-10
Displaying Dependencies Among MATLAB Code Files ...	9-14
Identifying How Much of a File Ran When Profiled	9-19
Using the MATLAB Code Analyzer Report	9-22
Running the Code Analyzer Report	9-22
Changing Code Based on Code Analyzer Messages	9-24
Other Ways to Access Code Analyzer Messages	9-25
Profiling for Improving Performance	9-27
What Is Profiling?	9-27
Profiling Process and Guidelines	9-28
Using the Profiler	9-29
Profile Summary Report	9-36
Profile Detail Report	9-38
The profile Function	9-46

Publishing MATLAB Code

10

Overview of Publishing MATLAB Code	10-2
What Is Meant by Publishing MATLAB Code?	10-2
Using Code Cells	10-2

Process for Publishing MATLAB Code	10-3
Example of Published MATLAB Code	10-4
Adding the Markup for the Example	10-10
Mark Up MATLAB Comments for Publishing	10-17
Publishing Overview	10-17
Document Titles and Introductory Text	10-18
Preformatted Text	10-25
Syntax Highlighted Sample Code	10-27
Bulleted or Numbered Lists	10-29
External Graphics	10-32
HTML Markup	10-35
LaTeX Markup	10-37
Inline LaTeX Math Equations	10-40
LaTeX Display Math	10-41
Force a Snapshot of Output	10-43
Bold, Italic, and Monospaced Text	10-44
Trademark Symbols	10-47
Hyperlinks	10-48
Cleaning Up Text Markup Before Publishing MATLAB Files	10-55
Summary of Markup for Publishing MATLAB Files	10-58
Mark Up MATLAB Code for Publishing	10-62
Overview of Marking Up MATLAB Code for Publishing ..	10-62
Specifying the Display of Code Output	10-62
Example of Marking Up Code	10-62
Specify Output Preferences for Publishing	10-66
About Publishing Configurations	10-66
Creating a Publish Configuration for a MATLAB File	10-68
Running an Existing Publish Configuration	10-95
Creating Multiple Publish Configurations for a File	10-96
About the publish_configurations.m File	10-107
Finding Publish Configurations	10-108
Removing Publish Configurations	10-108
Reassociating and Renaming Publish Configurations	10-108
Summary of Options for Presenting Your Code to Others	10-109

Creating a MATLAB Notebook to Publish to Microsoft Word

11

About MATLAB Notebooks	11-2
Contents of MATLAB Notebooks	11-2
Creating or Opening a MATLAB Notebook	11-2
Entering Commands in a MATLAB Notebook	11-7
Protecting the Integrity of Your Workspace in MATLAB Notebooks	11-8
Ensuring Data Consistency in MATLAB Notebooks	11-8
Debugging and MATLAB Notebooks	11-9
Configuring the MATLAB notebook Software	11-10
Defining MATLAB Commands as Input Cells for a MATLAB Notebook	11-12
Defining Input Cells for a MATLAB Notebook	11-12
Defining Cell Groups for a MATLAB Notebook	11-13
Defining Autoinit Input Cells for a MATLAB Notebook ...	11-14
Defining Calc Zones for a MATLAB Notebook	11-14
Converting an Input Cell to Text in a MATLAB Notebook	11-15
Evaluating MATLAB Commands in a MATLAB Notebook	11-17
Evaluating Input Commands	11-17
Evaluating Cell Groups	11-18
Evaluating a Range of Input Cells	11-20
Evaluating a Calc Zone	11-20
Evaluating an Entire MATLAB Notebook	11-20
Using a Loop to Evaluate Input Cells Repeatedly	11-21
Converting Output Cells to Text	11-22
Deleting Output Cells	11-22
Printing and Formatting a MATLAB Notebook	11-23
Printing a MATLAB Notebook	11-23
Modifying Styles in the MATLAB Notebook Template	11-23
Choosing Loose or Compact Format	11-24
Controlling Numeric Output Format	11-25
Controlling Graphic Output	11-25

Notebook Feature Reference	11-28
Bring MATLAB to Front	11-28
Define Autoinit Cell	11-29
Define Calc Zone	11-29
Define Input Cell	11-30
Evaluate Calc Zone	11-30
Evaluate Cell	11-31
Evaluate Loop	11-32
Evaluate MATLAB Notebook	11-32
Group Cells	11-32
Hide Cell Markers	11-33
Notebook Options	11-33
Purge Selected Output Cells	11-33
Toggle Graph Output for Cell	11-34
Undefine Cells	11-34
Ungroup Cells	11-35

Source Control Interface

12

Source Control Interface on Microsoft Windows	12-2
Setting Up the Source Control Interface on Microsoft	
Windows	12-3
Create Projects in Source Control System	12-3
Specify Source Control System with MATLAB Software ..	12-5
Register Source Control Project with MATLAB Software ..	12-7
Add Files to Source Control	12-9
Checking Files Into and Out of Source Control from the	
MATLAB Desktop on Microsoft Windows	12-11
Check Files Into Source Control	12-11
Check Files Out of Source Control	12-12
Undoing the Checkout	12-13
Additional Source Control Actions on Microsoft	
Windows	12-14
Getting the Latest Version of Files for Viewing or	
Compiling	12-14

Removing Files from the Source Control System	12-15
Showing File History	12-16
Comparing the Working Copy of a File to the Latest Version in Source Control	12-18
Viewing Source Control Properties of a File	12-20
Starting the Source Control System	12-21
Performing Source Control Actions from the Editor, Simulink, or Stateflow File Menu on Microsoft Windows	12-23
Troubleshooting Source Control Problems on Microsoft Windows	12-24
Source Control Error: Provider Not Present or Not Installed Properly	12-24
Restriction Against @ Character	12-25
Add to Source Control Is the Only Action Available	12-25
More Solutions for Source Control Problems	12-25
Source Control Interface on UNIX Platforms	12-26
Specifying the Source Control System on UNIX Platforms	12-27
MATLAB Desktop Alternative	12-27
Function Alternative	12-28
Setting a View and Checking Out a Folder with ClearCase Software on UNIX Platforms	12-29
Checking Files Into the Source Control System on UNIX Platforms	12-30
Checking In One or More Files Using the Current Folder Browser	12-30
Checking In One File Using the Editor, or the Simulink or Stateflow Products	12-31
Function Alternative	12-32
Checking Files Out of the Source Control System on UNIX	12-33
Checking Out One or More Files Using the Current Folder Browser	12-33

Checking Out a Single File Using the Editor, or the Simulink or Stateflow Products	12-34
Function Alternative	12-34
Undoing the Checkout on UNIX Platforms	12-36
Impact of Undoing a File Checkout	12-36
Undoing the Checkout for One or More Files Using the Current Folder Browser	12-36
Undoing the Checkout for a Single File Using the Editor, or the Simulink or Stateflow Products	12-36
Function Alternative	12-37

Internationalization

13

How the MATLAB Process Uses Locale Settings	13-2
Windows Platform-Specific Behavior	13-3
Macintosh Platform-Specific Behavior	13-3
Setting the Locale	13-4
Setting Locale on Windows Platforms	13-4
Setting Locale on Linux Platforms	13-6
Setting Locale on Macintosh Platforms	13-7
Troubleshooting I18n Messages and Settings	13-9
Asian Characters Incorrectly Displayed on Linux Systems	13-9
Characters Incorrectly Displayed on Windows Systems ..	13-10
datenum Might Not Return Correct Value	13-10
Numbers Display Period for Decimal Point	13-10
MATLAB Displays Messages in English	13-11
File or Folder Names Incorrectly Displayed	13-11

Index

Startup and Shutdown

The way you start the MATLAB® program depends on which platform you use. The following topics describe how to startup and shutdown MATLAB software on all supported platforms, with information about how you can customize startup and shutdown.

- “Starting and Quitting the MATLAB Program” on page 1-2
- “Startup Folder for the MATLAB Program” on page 1-12
- “Startup Options” on page 1-18
- “Toolbox Path Caching in the MATLAB Program” on page 1-23

Starting and Quitting the MATLAB Program


In this section...

- “Starting the MATLAB Program on Windows Platforms” on page 1-2
- “Starting the MATLAB Program on Linux Platforms” on page 1-6
- “Starting the MATLAB Program on Macintosh Platforms” on page 1-6
- “Quitting the MATLAB Program” on page 1-7

Starting the MATLAB Program on Windows Platforms

- “Associating Files with MATLAB on Windows Platforms” on page 1-3
- “Speeding Up MATLAB Start Up on Windows Systems” on page 1-6

There are several ways to start the MATLAB program on a Microsoft® Windows® platform:

- Use the shortcut on the Windows Start Menu. .
 - On Windows 7 systems, if chose to have the installer put a shortcut to the MATLAB program on the Windows Start menu, select **Start > MATLAB R2012a**
 - On Windows XP systems, select **Start > Programs > MATLAB > R2012a > MATLAB R2012a**
- If you chose to have the installer create a shortcut, double-click the MATLAB shortcut  on your Windows desktop.
- Double-click a file with any of a number of file extensions in the Windows Explorer tool. The installer sets up associations between these file types and MathWorks® products during installation. For example, double-clicking a file with a `.m` extension in the starts MATLAB and opens the file in the MATLAB Editor. For more information, see “Associating Files with MATLAB on Windows Platforms” on page 1-3.

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 “Adjust Desktop Appearance” on page 2-2. You can

specify other startup options, such the current folder upon startup—for more information, see “Startup Options” on page 1-18 and “Startup Folder for the MATLAB Program” on page 1-12.

If you have trouble starting MATLAB, see troubleshooting information in the “Troubleshooting”.

Associating Files with MATLAB on Windows Platforms

When you install MATLAB software on Windows platforms, the installer sets up associations between certain file types and MathWorks products. When you double-click a particular file type, identified by its file extension, Windows starts MATLAB and opens the file in the appropriate tool. The following table lists some of the file extensions the installer associates with MathWorks products and the behavior that results from this association. To learn how to change this behavior, see “Managing File Associations for MATLAB on Windows Systems” on page 1-5.

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 ¹	Displays icon for MATLAB in Windows Explorer tool
.p	Displays icon for MATLAB in Windows Explorer tool

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 your system associates .mat files with the Access™ application, when

1. MEX-file extensions are platform specific. See “Using Binary MEX-Files”.

you open a MAT-file from within MATLAB, it opens the Import Wizard to load the data.

Managing File Associations for MATLAB on Windows Systems.

You can associate any file types with MATLAB on computers running the Windows operating system. 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.

Occasionally, another program may already own the association with a particular file type. For example, the Microsoft Access program might own the association with files having a `.mat` extension. To reset this file association, use the Windows Default Programs control panel.

Note These instructions might not exactly apply to the version of the Windows operating system you are running on your computer. If you encounter differences or problems, see your Windows documentation.

- 1 Click the Windows Start menu.
- 2 Select **Control Panel**.
- 3 In the Control Panel window, select **Default Programs**.
- 4 In the Default Programs window, select **Associate a file type or protocol with a program**.
- 5 In the Set Associations window, select a file name extension to view which program currently opens it by default. In this example, select `.mat`.
- 6 To change the default association, click **Change Program**. This opens the **Open with** dialog box which lists other programs that might be recommended for this file extension. If you don't see MATLAB in the list, click **Browse** and navigate to the MATLAB installation folder. Open the folder for the version of MATLAB that you want to associate with this file extension and navigate to the `/bin/arch` folder, where `arch` is platform-specific, such as `win64`. Select the MATLAB executable, `MATLAB.exe` and click **OK**. You create file associations with particular versions of MATLAB.
- 7 Click **Close** to close the Set Associations dialog box.

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.

Speeding Up MATLAB Start Up on Windows Systems

On Windows systems, the MathWorks Installer installs and configures a utility program that can speed-up MATLAB startup, called the MATLAB Startup Accelerator. For information about this program, including information about how to modify the configuration, see License Administration Guide on the MathWorks Web site.

Starting the MATLAB Program on Linux Platforms

To start the MATLAB program on Linux[®] 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 path name, `matlabroot/bin/arch`, where `matlabroot` is the name of the folder in which you installed MATLAB and `arch` is an architecture-specific subfolder, such as `glnxa64`.

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 “Adjust Desktop Appearance” on page 2-2.

If the `DISPLAY` environment variable is not set or is invalid, the desktop will not display. If you have trouble starting MATLAB, see troubleshooting information in the “Troubleshooting”.

You can specify the current folder upon startup as well as other options—for more information, see and “Startup Options” on page 1-18.

Starting the MATLAB Program on Macintosh Platforms

There are several ways to start the MATLAB program on Macintosh[®] computers:

- Double-click the MATLAB icon in the Applications folder.
- Open a Terminal window, navigate to your MATLAB installation folder, and type `matlab` at the operating system prompt.

```
/Applications/MATLAB_R2012a.app/bin/matlab
```

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 “Adjust Desktop Appearance” on page 2-2. If the `DISPLAY` environment variable is not set or is invalid, the desktop will not display.

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.

If you have trouble starting MATLAB, see troubleshooting information in the “Troubleshooting”.

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 1-12 and “Startup Options” on page 1-18.

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 1-18.

Quitting the MATLAB Program

- “Ways to Quit the MATLAB Program” on page 1-8
- “Confirm Quitting the MATLAB Program” on page 1-8
- “Running a Script When Quitting the MATLAB Program” on page 1-9
- “Abnormal Termination” on page 1-9

Ways to Quit the MATLAB Program

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

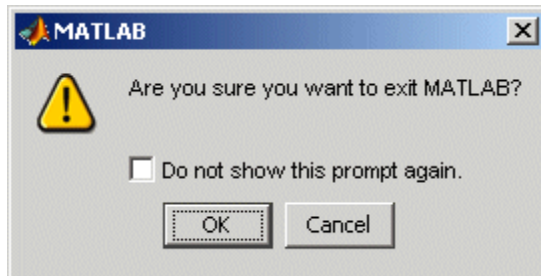
- Click the Close box  in the MATLAB desktop.
- Select **Exit MATLAB** from the desktop **File** menu.
- Type quit at the Command Window prompt.

MATLAB closes after

- Prompting you to confirm quitting, if that preference is specified (see “Confirm Quitting the MATLAB Program” on page 1-8.)
- 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 1-9 .

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 2-98.

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:

- `finishesav.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 1-9
- “Error Log Reporting” on page 1-10
- “Recovering Data After an Abnormal Termination” on page 1-11

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, select the option to send a report to 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 file name. Instead, specify a new file name. 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, select the option to send a report to 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 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, submit a Service Request via http://www.mathworks.com/support/contact_us/ts/help_request_1.html.

Emailing 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 email the reports.

Send email 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 emailed 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.
- 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.
- If you were editing a file in the Editor when MATLAB terminated unexpectedly, and you had the autosave 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.

Startup Folder for the MATLAB Program

In this section...

“What Is the Startup Folder?” on page 1-12

“Startup Folder on Windows Platforms” on page 1-13

“Startup Folder on Linux Platforms” on page 1-14

“Startup Folder on Macintosh Platforms” on page 1-14

“Changing the Startup Folder” on page 1-14

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 Linux 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.
- 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 and “Viewing Files and Folders on the Search Path” on page 6-72.

Startup Folder on Windows Platforms

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

How Started	Startup Folder
Double-click the MATLAB shortcut on your Windows desktop	The startup folder is set to the <code>userpath</code> value, whose default value is <code>My Documents\MATLAB</code> , or <code>Documents\MATLAB</code> on Windows Vista platforms. The <code>userpath</code> 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 <code>userpath</code> is added to the search path. If MATLAB does not find a valid <code>userpath</code> value upon startup, and the

How Started	Startup Folder
	Start in field is empty, the startup folder is the Windows desktop.
Double-click a file type associated with MATLAB	The folder in which the file resides is the startup folder. The <code>userpath</code> folder is automatically added to the search path.
In a DOS window	The folder in which you ran the <code>matlab</code> command is the startup folder. The <code>userpath</code> folder is automatically added to the search path.

Startup Folder on Linux Platforms

On Linux 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 MATLAB `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 default startup folder is the folder from which you started MATLAB.

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” on page 1-15
- “Changing the Startup Folder Using the Shortcut — Windows Platforms Only” on page 1-15
- “Changing the Startup Folder Using the startup.m File” on page 1-17

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 .

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.

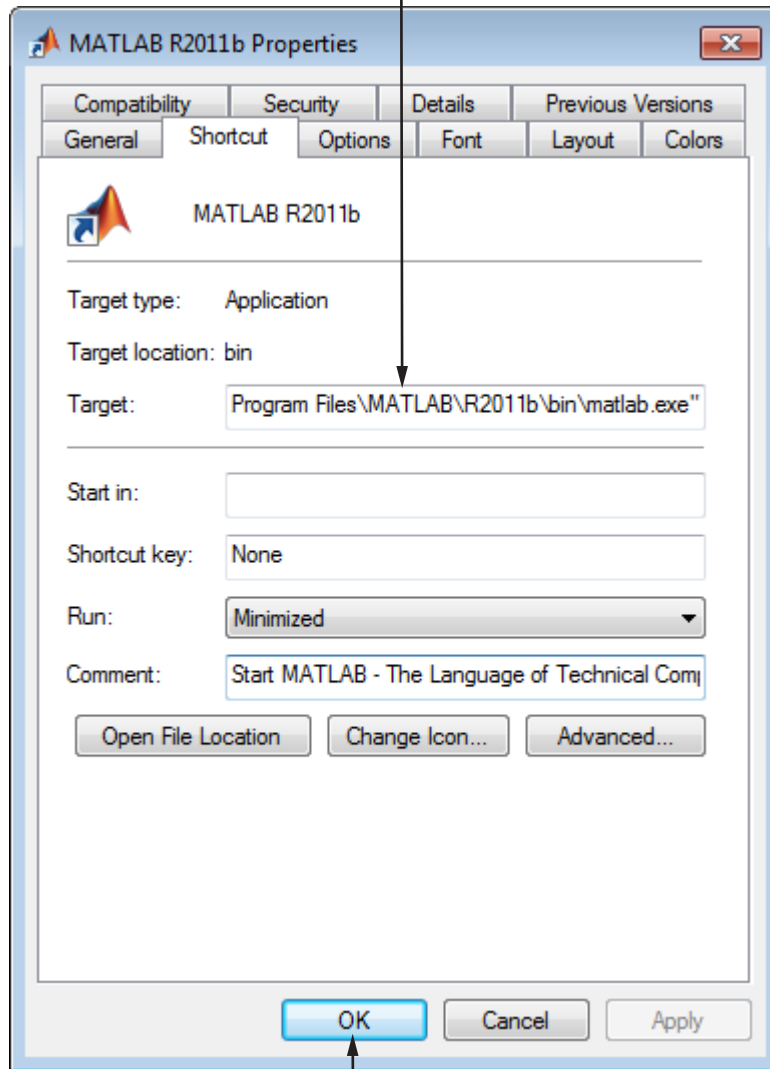
The Properties dialog box for MATLAB opens to the **Shortcut** pane.

- 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 1-13.

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

Enter full path to MATLAB startup folder.



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 in the MATLAB Startup File” on page 1-19.

Startup Options

In this section...

“Specifying MATLAB Startup Options” on page 1-18

“Commonly Used Startup Options” on page 1-20

“Passing Perl Variables on Startup” on page 1-21

“Startup and Calling Java Software from the MATLAB Program” on page 1-22

Specifying MATLAB Startup Options

You can specify startup options (also called command flags or command line switches) that instruct the MATLAB program to perform certain operations when you start it. On all platforms, you specify the options as arguments to the `matlab` command when you start is at the operating system prompt. For example, the following starts MATLAB and suppresses the display of the splash screen.

```
matlab -nosplash
```

On Windows platforms, you can precede a startup option with either a hyphen (-) or a slash (/). For example, `-nosplash` and `/nosplash` are equivalent.


On all platforms, you can also specify startup options using a MATLAB startup file—see “Specifying Startup Options in the MATLAB Startup File” on page 1-19

On Windows platforms, you can specify startup options in the MATLAB shortcut—see “Including Startup Options in a Shortcut on Windows Systems” on page 1-18.

Including Startup Options in a Shortcut on Windows Systems

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. For more information about the options, see “Commonly Used Startup Options” on page 1-20.

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 MATLAB code 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\R2010b\bin\matlab.exe -r "results"
```

Include the statement in double quotation marks ("*statement*"). Use only the file name, not the file extension or path name. 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 MATLAB code 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 MATLAB code file `results`:

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

Specifying Startup Options in the MATLAB Startup File

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 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
```

Place the `startup.m` file in the default or current startup folder, which is where MATLAB first looks for it. For more information, see .

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	<code>-c licensefile</code>	Set <code>LM_LICENSE_FILE</code> to <code>licensefile</code> . It can have the form <code>port@host</code> .
All	<code>-h</code> or <code>-help</code>	Display startup options (without starting MATLAB).
All	<code>-logfile</code> <code>"logfilename"</code>	Automatically write output from MATLAB to the specified log file.
Windows platforms	<code>-minimize</code>	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.

Platform	Option	Description
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 <code>nojvm</code> , 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 <code>-nojvm</code> option. However, you can start MATLAB once <i>without</i> the <code>-nojvm</code> option, set the preference, and quit MATLAB. MATLAB remembers that preference when you start it again, even if you use the <code>-nojvm</code> option.
All	-nosplash	Start MATLAB without displaying its splash screen.
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 path names or file extensions. Enclose the statement in double quotation marks (" <i>statement</i> "). Use semicolons or commas to separate multiple statements
All	-singleCompThread	Limit MATLAB to a single computational thread. By default, Windows makes use of the multithreading capabilities of the computer on which it is running.

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.

Toolbox Path Caching in the MATLAB Program

In this section...

“About Toolbox Path Caching in the MATLAB Program” on page 1-23

“Using the Cache File Upon Startup” on page 1-23

“Updating the Cache and Cache File” on page 1-23

“Additional Diagnostics with Toolbox Path Caching” on page 1-26

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 MATLAB code 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 folders, 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 MATLAB code files you create and any files provided by 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.

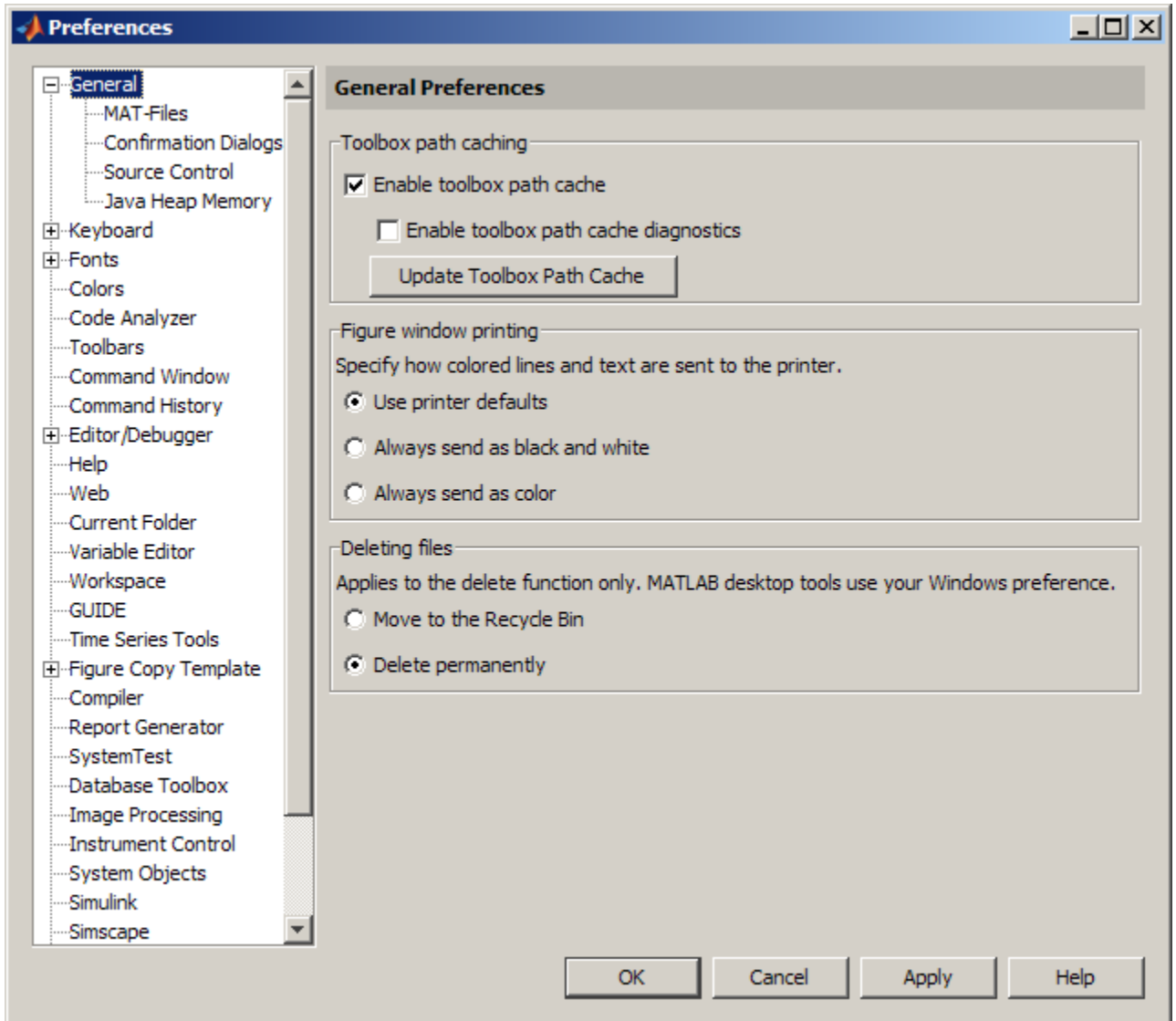
Steps to Update the Cache

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**.

Desktop

- “Adjust Desktop Appearance” on page 2-2
- “Keyboard Shortcuts” on page 2-36
- “Basic Operations” on page 2-63
- “Web Browsers and MATLAB” on page 2-71
- “License Management and Software Updates” on page 2-76
- “Macintosh Platform Conventions” on page 2-79
- “Accessibility” on page 2-81
- “About Preferences” on page 2-93

Adjust Desktop Appearance

In this section...
“Fonts” on page 2-2
“Color Settings” on page 2-8
“Toolbar Customization” on page 2-11
“Columns — Resize, Rearrange, and Sort in Desktop Tools” on page 2-12
“Enable and Disable Confirmation Dialog Boxes” on page 2-13
“Open and Rearrange Desktop Tools and Documents” on page 2-13

Fonts

You can specify the font characteristics (type, style, and size) for:

- Sets of MATLAB desktop tools, based on their predominant content — code, text, or HTML Proportional text
- Each desktop tool individually
- A combination of sets and individual tools.

You can set some font options differently for printing — see “Print Options” on page 2-64.

Setting Fonts for Sets of Tools Based on Content

You can specify fonts for tools based on their predominant content — text, code, or HTML proportional font. For an example of how fonts affect text in the Editor, see “Monospace Font Compared to Proportional Font” on page 2-4.

To set fonts for a tool:

- 1** Select **File > Preferences > Fonts**.
- 2** Under **Desktop code font** select a font type, style, and size from the drop-down menus.

For font size, you can type a size, including a size not shown as a choice in the drop-down menu.

3 Under **Desktop text font**, do one of the following:

- Select **Use system font**.
- Select the font type, style, and size from the drop-down menus.

For font size, you can type a size, including a size not shown as a choice in the drop-down menu.

4 Review the list of tools currently using each font group—**Desktop code font**, **Desktop text font**, and **Custom fonts**.

If you want to move a tool from one group to another, or if you want to customize fonts for a particular tool, follow the steps in “Setting Fonts for Individual Tools and HTML Proportional Text” on page 2-3. Otherwise, click **OK**.

Note If you are running MATLAB on a Linux² or UNIX³ platform, consider selecting **Use antialiasing to smooth desktop fonts**. Note however, that this option requires you to restart MATLAB.

Setting Fonts for Individual Tools and HTML Proportional Text

Custom font preferences enable you to change the font for HTML proportional text and individual desktop tools. For an example of how fonts affect text in the Editor, see “Monospace Font Compared to Proportional Font” on page 2-4.

To set custom fonts:

- 1** Select **File > Preferences > Fonts > Custom**.
- 2** Select the tool you want to customize from the **Desktop tools** list.

2. Linux is a registered trademark of Linus Torvalds.

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

The type of font the tool currently uses appears under **Font to Use**.

3 Under **Font to Use**, select one of the following:

- **Desktop code**.
- **Desktop text**
- **Custom**, and then specify the font characteristics.

4 Click **OK**.

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

About HTML Proportional Text Font

HTML proportional-text fonts present characters with varying widths. Proportional-font text takes less space, but its characters are harder to align. Proportional-text font is easier to read, and is typically used for descriptive and informational text. For example, the MATLAB default is to use proportional fonts for the following:

- 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 Code Analyzer messages

Monospace Font Compared to Proportional Font. A monospace font is useful when you care about alignment, but it does take more space than a proportional font.

With a monospaced font, all characters are the same width.

1234567890
 abcdefghij
 ABCDEFGHIJ

With a proportional font, characters are different widths.

1234567890
 abcdefghij
 ABCDEFGHIJ

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 previously changed settings, and now want to revert to the default values, update the settings using the values in the table.

Font Type	Factory Default Characteristics	Tools Using Font Type by Factory Default
Desktop code font	Monospaced, Plain, 10 point	<ul style="list-style-type: none"> • Command History • Command Window • Editor (which also applies to the Shortcuts Editor)
Desktop text font	Your system's current font.	<ul style="list-style-type: none"> • Current Folder browser (which also applies to the Path browser) • Function Browser in the Command Window and Editor • Help Navigator • Workspace browser • Variable Editor

Example of Changing the Font

This example changes the settings for the desktop code font; changes the Command History font preference so that it uses the desktop text font instead of the code font; and specifies a custom font for the Current 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.

Font Type	Font Characteristics	Tools Assigned Font
Desktop code	Times New Roman® font, Plain, 14 point	Command Window Editor
Desktop text	Same as your current system font, which appears in the dimmed fields below the Use system font check box.	Command History Workspace browser Variable Editor Function Browser
Custom	SanSerif, Plain, 8 point	Help Navigator
Custom	SanSerif, Plain, 10 point	HTML Proportional Text
Custom	Monotype Corporation Arial® Narrow font, Plain, 11 point	Current Folder browser

Font Troubleshooting – Document Displays Boxes and Odd Symbols Instead of Text

If you open a file created by someone else, and 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.

To resolve this problem, 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.

Where MATLAB Gets Fonts

Unless you change it, MATLAB gets fonts from the operating-system font list. Any fonts on the system list that MATLAB cannot display are excluded 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.

If a new compatible font becomes available to MATLAB, consider updating fonts on your Windows platform, as follows:

- 1** Use the Windows Control Panel to install the font.

For more information, refer to the Windows help.

- 2** Restart 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 bitmapped font is compatible font for graphics objects, such as `xlabel`, `ylabel`, `title`, and `text`.

Color Settings

This section describes how to change the colors that desktop tools use.

Changing Text, Background, and Hyperlink Colors in Desktop Tools

To change the colors that MATLAB uses for text and background in desktop tools follow these steps:

Note The colors you specify also apply to the Import Wizard, but do not apply to the Help display pane or the Web Browser.

- 1** Select **File > Preferences > Colors**.

- 2** Clear **Use system colors**.

System colors are the text and background colors that your platform (for example, Microsoft Windows) uses for other applications.

- 3 Select the colors you want to use from the **Text** and **Background** color palettes.

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

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

- 4 Under **Other colors**, select the color you want to use for hyperlinks.
- 5 Click **OK**.

Changing Syntax Highlighting Colors

In the Command Window, Command History, Editor, and Shortcuts callback area, MATLAB conveys syntax information using different colors. This feature, known as syntax highlighting, helps you to identify syntax elements, such as `if/else` statements at a glance. The Editor also provides syntax highlights colors for other languages.

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

To change syntax highlighting colors, follow these steps:

- 1 Select **File > Preferences > Editor/Debugger**, and then click **Language**.
- 2 In the **Language** drop-down menu, select the language for which you want to change syntax highlighting colors.
- 3 In the **Syntax highlighting** area, select **Enable syntax highlighting**.
- 4 Change the colors.
 - If you set the **Language** to MATLAB, click the **Set syntax colors** link, and then change the colors under **MATLAB syntax highlighting colors**.

- If you did not set the **Language** to MATLAB, change the colors under **Syntax highlighting** .

5 Click **OK**.

Changing Code Analyzer Colors

Code Analyzer helps you to identify potential problems and refine your MATLAB code. By default, the Editor indicates:

- Code for which there are warnings, by underlining that code with an orange wavy line and placing an orange line in the message bar.
- Code for which there are errors, by underlining that code with a red wavy line and placing a red line in the message bar.
- Code that MATLAB can fix automatically (autofix), by highlighting that code in tan.

To change code analyzer colors:

- 1** Select **File > Preferences > Colors > Programming Tools**.
- 2** Select the colors you want the code analyzer to use for warnings, autofix highlighting, or both.
- 3** Decide if you want autofix highlights to appear in the Editor.

Clear **Autofix highlight** if you do not want autofix highlights to appear in the Editor; select **Autofix highlight** if you do.

- 4** Click **Apply**.
- 5** Decide if you want to change the color that the code analyzer uses for errors.
 - If you do not, go to step 6.
 - If you do, then:
 - a** In the left navigation pane, click **Colors**.
 - b** Under **MATLAB syntax highlighting colors**, change the color for **Errors**.

In addition to changing the color of code analyzer indicators for errors, this action also changes the color for errors in the Command Window, Command History window, Editor, and Shortcuts callback area.

6 Click **OK**.

For more information, see “Automatically Check Code in the Editor — Code Analyzer” on page 8-91.

Toolbar Customization

The toolbar on the desktop and on undocked tools provides easy access to frequently used operations.

- To display a tooltip that describes a toolbar button action — Position the pointer over a button for a couple seconds and a tooltip appears describing the button action.
- To show or hide a toolbar:
 - ▀ Right-click a toolbar or menu bar, and then select a toolbar from the context menu.
 - ▀ In a figure window, on the **View** menu, select the toolbar of interest.
- To reposition toolbars — If a tool has multiple toolbars, click and hold the toolbar anchor, and then drag the toolbar to a different location.





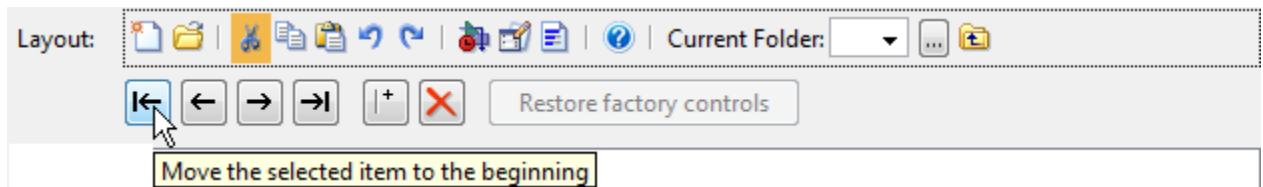
Toolbar anchor

- To show, hide, or rearrange toolbar buttons — follow these steps:
 - 1** Select **File > Preferences > Toolbars**.
 - 2** From the **Toolbar** drop-down menu, select the toolbar that you want to customize.

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

- 3 In the **Controls** list, select or clear the check box for controls that you want to display or remove from the toolbar, respectively.
- 4 In the **Layout** area, rearrange the order of the controls and separator bars on the selected toolbar, by doing either of the following:
 - Drag the icon for a control or separator bar to another position.
 - Select a **Layout** icon, and then click one of the **Layout** buttons below the layout icons.

For instance, to move the MATLAB desktop cut icon to the beginning of the toolbar, select the **Cut** icon , and then click the **Move the selected item to the beginning** button .



- 5 Click **Apply** or **OK**.

Columns – Resize, Rearrange, and Sort in Desktop Tools

Some desktop tools, such as the Current Folder browser, present information in columns. 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 information cropped by narrow column	Position the pointer over an item to view the full value for that item. It displays like a tooltip.

To:	Do This:
Rearrange the columns	Drag a column header to a different position.
Sort the information by a particular column	<p>Click the column header. For example, in the Current Folder browser, click the Date Modified date to sort the items in date order.</p> <p>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.</p>

Enable and Disable Confirmation Dialog Boxes

For many actions you try to perform, MATLAB displays confirmation dialog boxes before performing that action. To specify whether or not you want a given confirmation dialog box to appear:

- 1 Select **File > Preferences > General Confirmation Dialogs**.
- 2 Select the check box for each dialog box you want to appear; clear it if you do not want it to appear.
- 3 Click **OK**.

Note When you perform an action that triggers an enabled confirmation dialog box, that dialog box 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 dialogs preference.

Open and Rearrange Desktop Tools and Documents

- “Desktop Tools — Open and Rearrange” on page 2-14
- “Desktop Documents — Open and Rearrange” on page 2-24

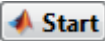
- “Desktop Arrangements — Save, Reuse, Rename, and Delete” on page 2-33

Desktop Tools — Open and Rearrange

Open Desktop Tools. To open a desktop tool, do any one of the following:

- Select a tool from the **Desktop** menu.

A check mark in front of the tool name on the menu indicates that the tool is open.


- Click the desktop  button, select **Desktop Tools**, and then click the tool you want to open.
- Use a function:
 - Command History: `commandhistory`
 - Command Window: `commandwindow`
 - Current Folder Browser: `filebrowser`
 - Editor: `edit`
 - Profiler: `profile` with the `viewer` option.
 - Web Browser: `web`
 - Workspace Browser: `workspace`

Resize and Move Desktop Tools. To resize tools on the MATLAB desktop, use the mouse or the keyboard, as described in the following sections:

Resize Desktop Tools Using the Mouse

To expand or reduce the size of adjacent tool windows, use the pointer to drag the separator bar that appears between them. 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 \leftrightarrow .

- 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 .

Resize 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** — 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.

Determine Current Size of the Command Window

To determine the number of characters and lines that can display in the Command Window, given its current size, issue this command:

```
get(0, 'CommandWindowSize')
```

For example, a result of 50, 25 means that 50 characters can display across the Command Window, and 25 lines can display without scrolling.

Note If you enable the **Set matrix display width to eighty columns** preference, then the result for that same size Command Window is 80, 25. For more information, see “Matrix Display Width” on page 3-38.

Move 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.

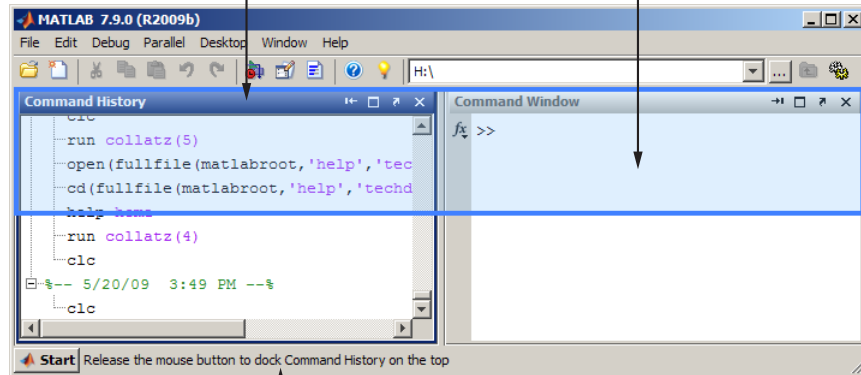
- 2** When the outlined position is where you want the tool to be, release the mouse button.

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.

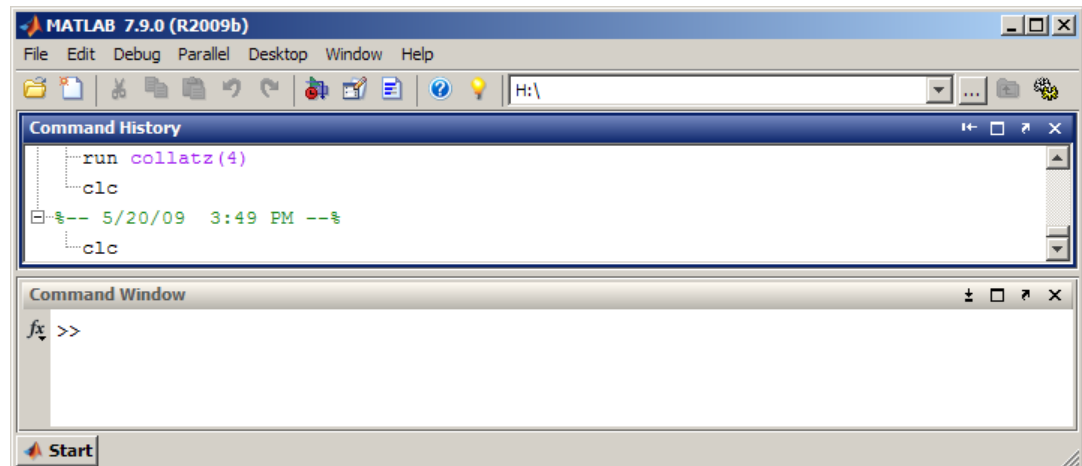
To move the Command History window, drag its title bar.

When the outline of the Command History window appears where you want it, release the mouse.



The status bar displays instructions about moving the tool.

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.



Move 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** — 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.


Undock 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.

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  on 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.

Move 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  on the menu bar for that tool.
- Select **Dock** from the **Desktop** menu for that tool.

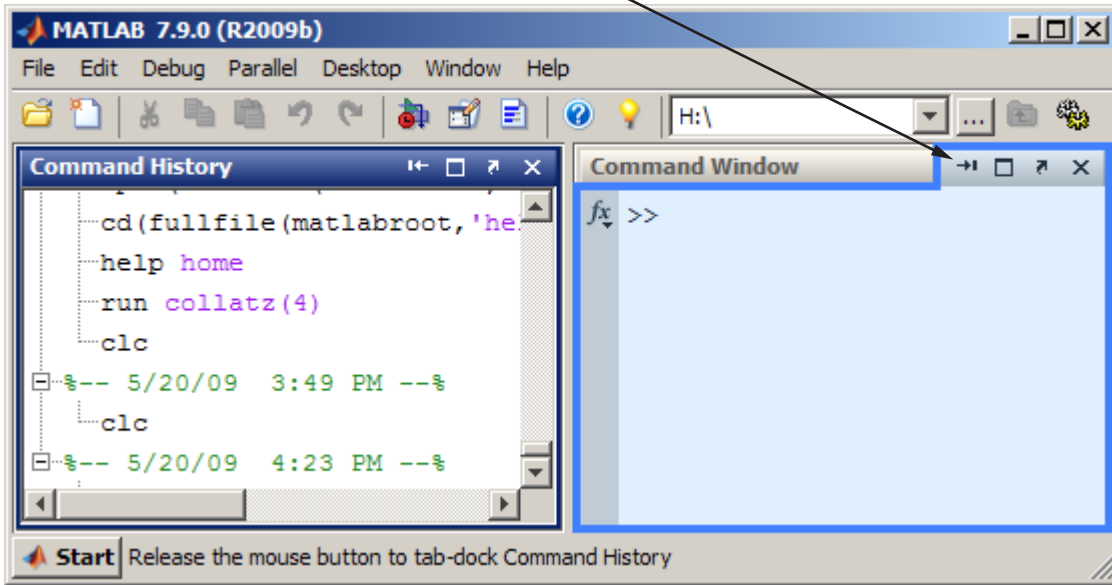
Group Desktop Tools Together. You can group tools so that they occupy the same location on the MATLAB desktop. Basically, you are stacking one tool on top of another. Then, you can access the individual tools using the tool name on the title bar:

To group tools:

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

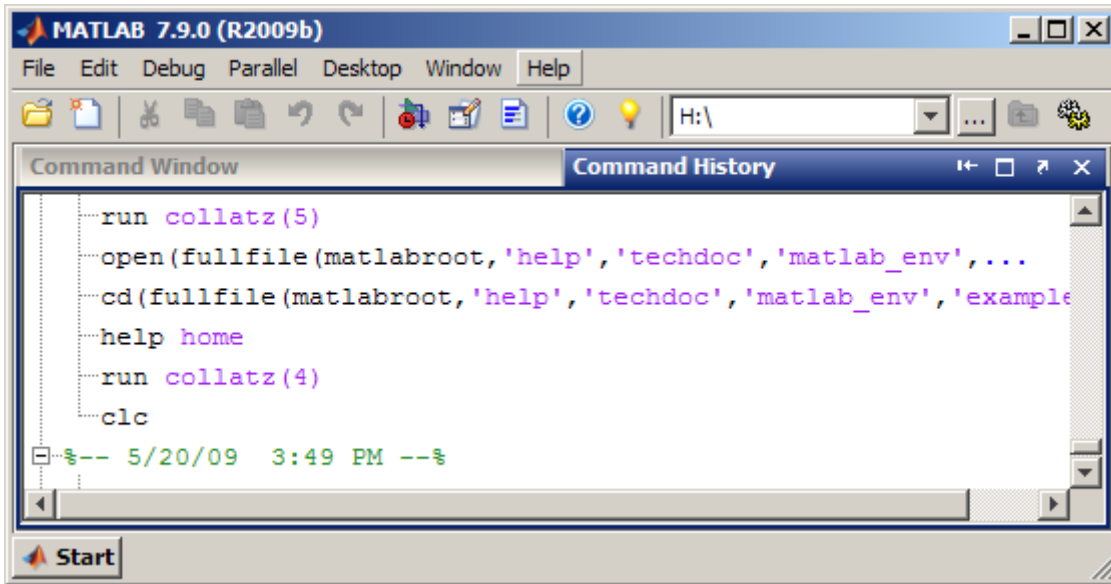
An outline of the tool you are dragging overlies the target tool.

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.

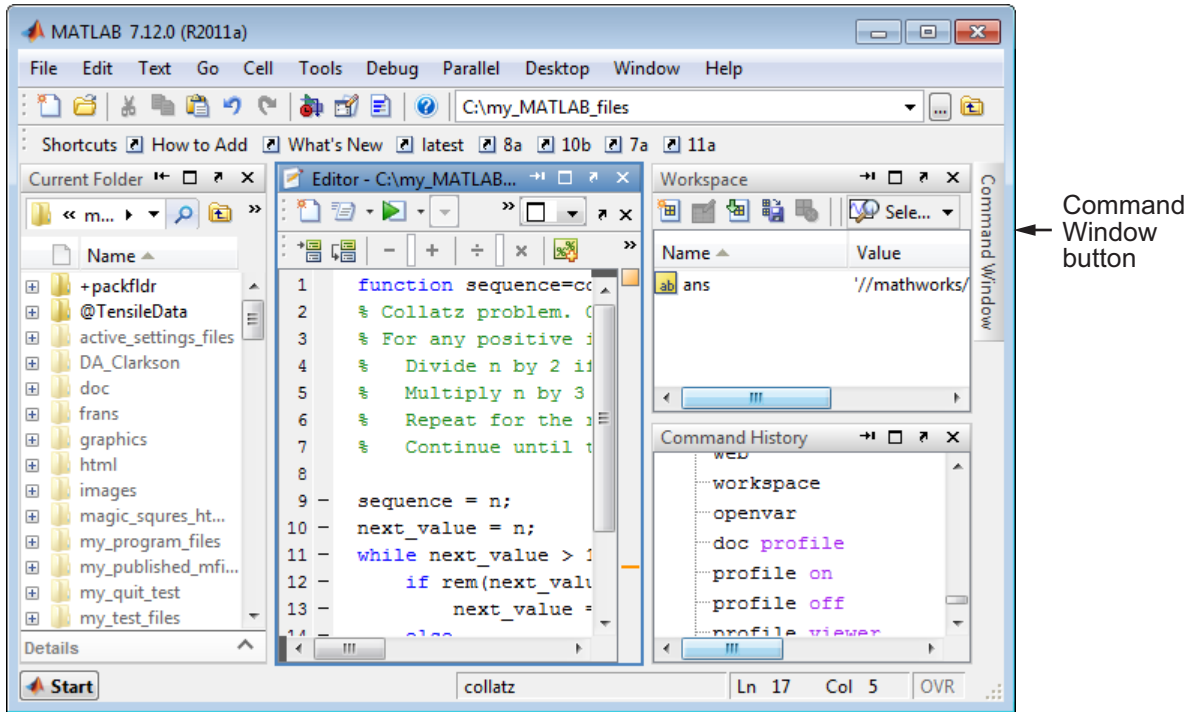
Maximize Available Space on the Desktop. To hide the title bars for desktop tools so they use less space, select **Desktop > Titles**. This action clears the check mark next to the **Titles** menu item. Identify a desktop tool with a hidden title by hovering over the area where the title bar used to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

Maximize 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.

Minimize 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 image, 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 it.

Desktop Documents – Open and Rearrange


- “Open Documents” on page 2-24
- “Navigate Among Open Documents Using the Document Bar” on page 2-27
- “Adjust the Document Bar” on page 2-27
- “Position Documents” on page 2-27
- “Move and Resize Documents” on page 2-31
- “Close Documents” on page 2-32
- “Move Documents Outside of the Desktop (Undock)” on page 2-32
- “Dock Documents and Tools” on page 2-33
- “Group Documents in a Tool Outside the Desktop” on page 2-33

Open 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**.

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.

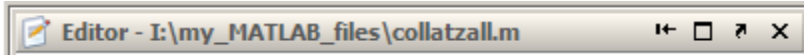
Document Type and Tool	How to Open Document	Where Document Appears by Default	Other Techniques to Open Document
Text file in the Editor	Click the Open file button  on the desktop toolbar and select the file.	In the last location of the Editor. The default location for the Editor is outside the desktop.	“Open Existing Files” on page 8-2
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 5-22
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.	“Web Browsers and MATLAB” on page 2-71
Figure	Use the <code>plot</code> 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.





- Use buttons in the titlebar for a tool. The following image shows the Editor titlebar, for example.



To Accomplish This:	Do This Using the Titlebar for a Tool:
Minimize the tool	Click  .
Maximize the tool	Click  .
Undock the tool from the desktop	Click  .
Close the tool, including all documents in the tool	Click  .

- Use buttons in a toolbar for a tool to arrange documents within a tool. The following image shows the Editor toolbar.









To Accomplish This:	Do This Using the Editor Toolbar:
Arrange documents in the Editor	Use  , as described in “Position Documents” on page 2-27.
Undock a document	Click  .
Close and save the document currently displaying	Click  .
Close and not save the document currently displaying	Click Ctrl +  .


Navigate 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.

Adjust the Document Bar. You can show, hide, move, alphabetize, and adjust the size of the tabs on 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.
Hide the document bar.	Right-click the document bar, and then select Bar Position > Hide .
Move the document bar.	Do one of the following: <ul style="list-style-type: none"> • 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.	Right-click on the document bar and select Alphabetize .
Reorder document names on the document bar.	Drag a document name to a different position on the document bar.

Position 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 table that follows. When you *tile* documents, they are all visible within the tool, arranged in a grid pattern.

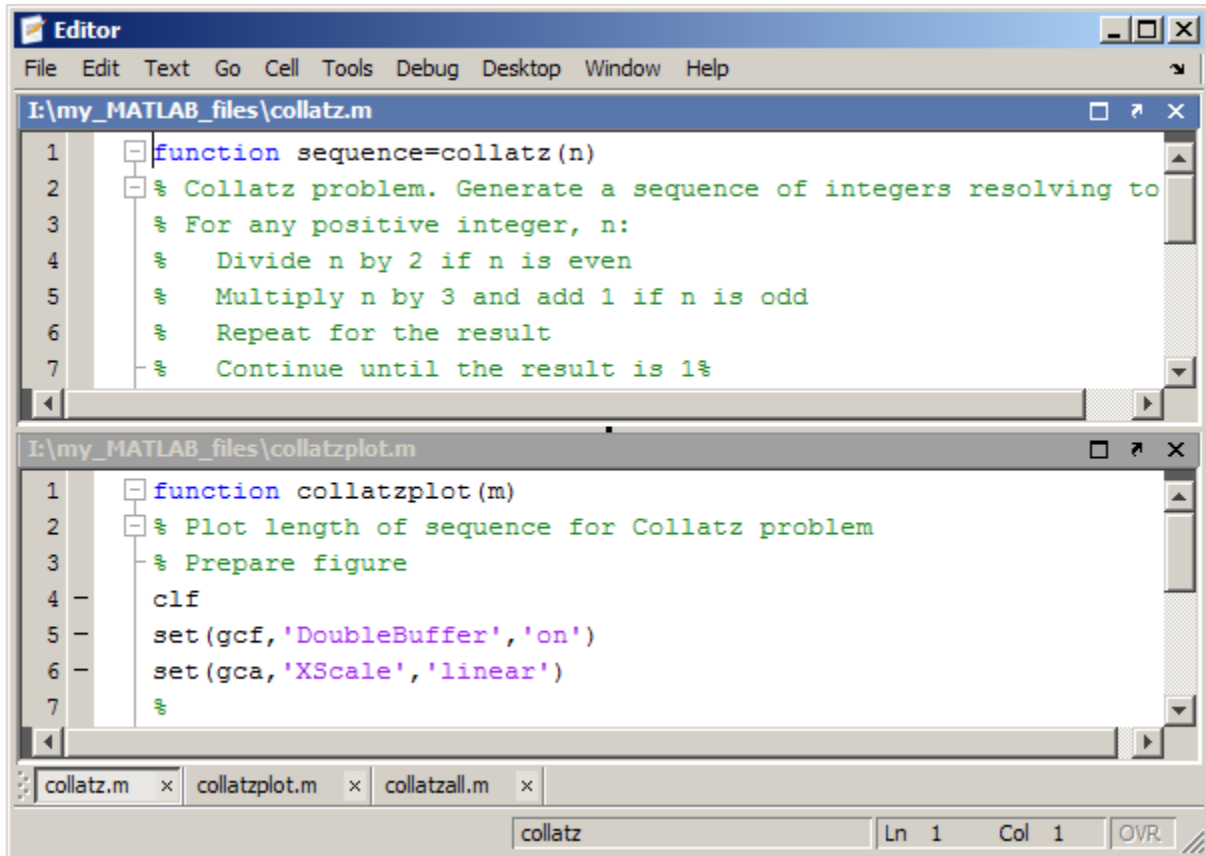
To:	Do This:
View one document (default)	From the Arrange Documents drop-down menu, choose the Maximize option 
View all open documents, layered on top of each other	In the Arrange Documents drop-down menu, choose the Float option  . Optionally, select Window > Cascade to make the document arrangement neater.
View all documents, side-by-side	In the Arrange Documents drop-down menu, choose the Left/Right Tile option 
View open documents, one above the other	In the Arrange Documents drop-down menu, choose the Top/Bottom Tile option 
View open documents, tiled within the tool	<ul style="list-style-type: none"> • In the Arrange Documents drop-down menu, choose the Tile option . • In the grid that opens, move the pointer to highlight the number and position of the tiles. You must choose an even number. <p>The tiles that will contain documents are blue, whereas the tiles that will be empty are gray.</p>
View a subset of open documents, tiled within the tool	<ol style="list-style-type: none"> 1 Select Window > Tile. 2 Indicate the documents you want to view and the grid pattern to use for the arrangement of their display.

To:	Do This:
Replace a tiled document	See Replace a Tiled Document with an Out-Of-View Document on page 29
Close an empty tile	On the separator bar, move the pointer over the handle ■, and then click the Close box  that replaces the handle.

Replace a Tiled Document with an Out-Of-View Document

You can replace a currently tiled document with another that is on the document bar, but not currently in view.

For example, suppose you have three documents open in the Editor — `collatz.m`, `collatzplot.m`, and `collatzall.m`. The first two documents are in view, as shown in the following image.

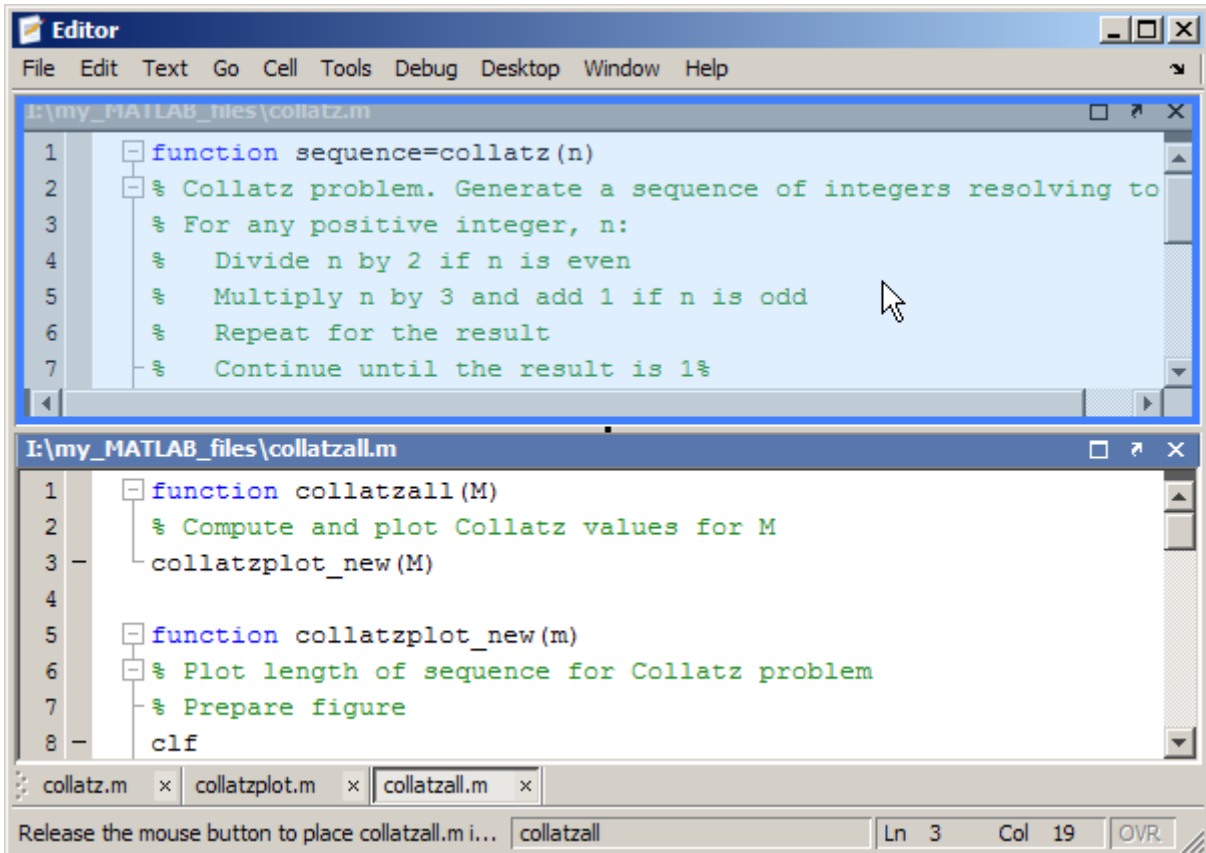


To view `collatzall.m` in the top tile, follow these steps:

- 1 Click the title bar of the file in the top tile, `collatz.m`.
- 2 On the document bar, click `collatzall.m`.

If you previously viewed `collatzall.m` in the bottom tile, `collatzall.m` displays in the bottom tile, regardless of which title bar you click in step 1.


- 3 If `collatzall.m` displays in the bottom tile, drag its title bar to the top tile to get the arrangement you want.




Now, `collatzall.m` displays in the top tile and `collatzplot.m` displays in the bottom tile.

Move and Resize 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 .


To Accomplish This:	Do This:
Minimize (hide) a floating document.	Click the minimize button  on 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.

Close Documents. The following table summarizes the key options for closing documents.

To Close:	Do This:
All documents open within a tool	Click the Close box  on the title bar for the tool.
Documents without saving changes	Hold the Ctrl key while clicking the Close box.
All documents undocked from within their tools	Select Window > Close All Documents from the desktop.

Move Documents Outside of the Desktop (Undock). 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, click the Undock arrow  on the title bar of the tool you want to move outside the desktop.

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

Group 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.

Desktop Arrangements – Save, Reuse, Rename, and Delete

- “Overview of Desktop Arrangements” on page 2-33
- “Save a Desktop Arrangement” on page 2-34
- “Reuse a Saved or Predefined Desktop Arrangement” on page 2-34
- “Rename a Saved Desktop Arrangement” on page 2-34
- “Delete a Saved Desktop Arrangement” on page 2-35

Overview of Desktop Arrangements. 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, tools such as the Help browser, Web browser, and Variable 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 MATLAB Web browser open each time you start MATLAB, add `web` to a `startup.m` file. For more information, see “Startup Options” on page 1-18.

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

Save a Desktop Arrangement. 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. The layout last used in a session is `MATLABDesktop.xml`. The `MATLABDesktop.xml` file loads when you start MATLAB and is overwritten when you close MATLAB.

Reuse a Saved or Predefined Desktop Arrangement. 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.
If you are dissatisfied with your current desktop arrangement, use this option to restore the default desktop arrangement
- **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.

Rename a Saved Desktop Arrangement. 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.

Delete a Saved Desktop Arrangement. 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.

Keyboard Shortcuts

In this section...

“Keyboard Shortcuts” on page 2-36

“Choose a Set of Keyboard Shortcuts” on page 2-40

“Compare Sets of Keyboard Shortcuts” on page 2-43

“Display Keyboard Shortcuts” on page 2-45

“Customize Keyboard Shortcuts” on page 2-48

“Evaluate and Resolve Keyboard Shortcut Conflicts” on page 2-54

“Examples of Creating, Modifying, and Deleting Keyboard Shortcuts” on page 2-56

“Delete a Set of Keyboard Shortcuts” on page 2-59

“Use Keyboard Shortcuts Settings Files Created on Other Systems” on page 2-60

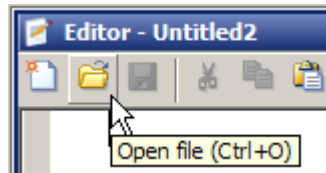
“Keyboard Shortcut Restrictions” on page 2-60

Keyboard Shortcuts

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, it appears next to that action on the menu and as a tooltip for that action on the toolbar. For example, the default keyboard shortcut for opening a file is **Ctrl+O**.



A tooltip for the action also appears on the Editor toolbar.



An action can have multiple keyboard shortcuts. All defined shortcuts work, but only one appears on the desktop menu and as a toolbar tooltip.

You can:

- Choose from a set of shortcuts that install with MATLAB.
- Create customized sets of shortcuts.
- Use a set of shortcuts copied from another system

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.

What Is a Mnemonic?

A *mnemonic* is a means of using keystrokes to perform a desktop action. For instance, 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. This convention helps you to remember the keystrokes required to use the mnemonic.

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.

File

Using Mnemonics

To open a menu or activate a button using mnemonics, press the **Alt** key and the letter key indicated by 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**.

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.

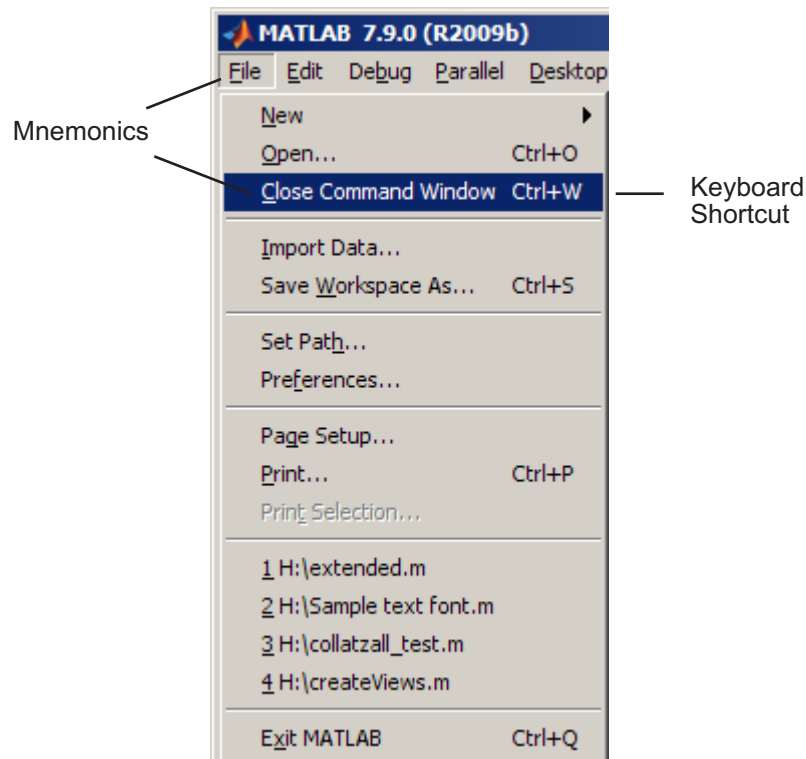
What Is a Keyboard Shortcut?

A *keyboard shortcut* is a means of using keyboard key strokes to perform a desktop action, without opening a desktop menu. For example, the default keyboard shortcut for opening a file is **Ctrl+O**. See “Keyboard Shortcuts” on page 2-36 for more information.

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.

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**.

Choose a Set of Keyboard Shortcuts

By default, MATLAB uses keyboard shortcut settings that correspond to the platform on which you are running. To choose different keyboard shortcut settings, 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 make a selection from the drop-down list, as summarized in this table.

Settings File	Option to Select	Details
Installed with MATLAB	<ul style="list-style-type: none"> • On Mac®, Mac Default Set • On all other systems, Windows Default Set or Emacs Default Set 	For a description of the files that install with MATLAB, see “Installed Settings Files for Keyboard Shortcuts” on page 2-41
Previously added	The file name	No additional information.
On your system, but not in the drop-down list	Browse	“Browse to Keyboard Shortcuts Settings Files” on page 2-41
Created by someone else and uploaded to File Exchange	Search File Exchange for Downloadable Shortcut Sets	“Download Keyboard Shortcut Settings Files from File Exchange” on page 2-42.
MATLAB keyboard shortcuts available in Version 7.9 (R2009a) and earlier releases	Search File Exchange for Downloadable Shortcut Sets	“Download Keyboard Shortcut Settings Files from File Exchange” on page 2-42.

- 3 Click **Apply**.

Installed Settings Files for Keyboard Shortcuts

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

Operating System	Keyboard Shortcut Settings Files Installed with MATLAB
Windows	<ul style="list-style-type: none"> • Windows Default Set (Default) • Emacs Default Set
UNIX	<ul style="list-style-type: none"> • Emacs Default Set (Default) • Windows Default Set
Macintosh	<ul style="list-style-type: none"> • Macintosh Default Set (Default)

Browse to Keyboard Shortcuts Settings Files

Browse to use a keyboard shortcuts settings file that is on your system, but not an **Active settings** choice in the Keyboard Shortcuts Preferences dialog box. This situation typically arises when you copy a settings file from another system to a folder other than the `prefdir` directory. To browse to a settings file and make it your active settings file, follow these steps:

- 1 Choose **File > Preferences > Keyboard > Shortcuts**.
- 2 In the **Active settings** field, click the down arrow, and then select **Browse**.
- 3 In the Open dialog box, navigate to the folder containing the settings file.
- 4 Select the settings file, and then click **Open**.
- 5 In the Keyboard Shortcuts preferences pane, click **OK**.

The settings file you selected in step 4 is now the active settings file for MATLAB.

Future MATLAB sessions will provide this settings file as a choice in the **Active settings** drop-down menu.

Download Keyboard Shortcut Settings Files from File Exchange

Download keyboard shortcut settings files from File Exchange when you want to do either of the following:

- Restore the MATLAB default keyboard shortcuts that were in place for MATLAB Version 7.9 (R2009a) and earlier releases.
- Find and download keyboard shortcuts that others created and uploaded to File Exchange.

Follow these steps:

- 1** Choose **File > Preferences > Keyboard > Shortcuts**.
- 2** In the **Active settings** field, click the down arrow, and then select **Search File Exchange for Downloadable Shortcut Sets**.

MATLAB opens File Exchange.


- 3** Enter your MathWorks account information, and then click **Submit**.


If you do not have a MathWorks account, create one by clicking **Create an account**, and then completing the form that opens in your Web browser.

- 4** Search File Exchange for the keyboard shortcut set that you want to use.

When you follow the steps presented so far, File Exchange lists all files tagged with `keyboard shortcuts configurable` including:

- MATLAB Desktop R2009a Non-Default Keyboard Shortcut sets
- MATLAB Desktop R2009a Default Keyboard Shortcut sets

For a description, click the file name in the **File Summary** column of File Exchange. Click the Back button  to return to the list of files.

- 5** Click the download button  next to the file you want to download.
- 6** Respond to the Download MATLAB Desktop confirmation dialog boxes as follows:
 - a** Click **Download**.


b Click Change Current Folder to Download Location.

The downloaded .ZIP file appears in the Current Folder Browser. Expand it to preview its contents.

- 7** In the Current Folder browser, right-click the downloaded .ZIP file, and then select **Extract**.

MATLAB creates a subfolder with the same name as the .ZIP file and extracts the files from that .ZIP file into the newly created folder.

- 8** In the Current Folder browser, expand the newly created folder, and then double-click the settings file you want to use.

A keyboard key icon  preceding a file name indicates a valid keyboard shortcut settings file.

- 9** In the Keyboard Shortcuts Preferences dialog box, review the settings, and then click **OK**.

The newly downloaded settings file is now in effect.

Compare Sets of Keyboard Shortcuts


Compare sets of keyboard shortcuts to:

- Upgrade MATLAB from a version before Version 7.9 (R2009b).
MATLAB 7.9 made keyboard shortcuts consistent across the desktop. Therefore, you might find that shortcuts you used before Version 7.9 are different.
- See how a set of keyboard shortcuts you found on File Exchange differs from your current set of keyboard shortcuts.
- See how a set of keyboard shortcuts differs from the default set.

Steps for Comparing Keyboard Shortcuts

To compare your current set of keyboard shortcuts to another set:

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

- 2 Click the Actions button .
- 3 From the drop-down menu, choose the set of keyboard shortcuts to which you want to compare the current set.
- 4 The Comparison Tool opens and displays the two keyboard shortcut sets side-by-side.

Read the Results of Comparing Sets of Keyboard Shortcuts



When you compare keyboard shortcut sets, they appear in the Comparison Tool as follows:

- One set displays on the left side of the tool and the other set displays on the right side of the tool.
- Each column header displays the name of the keyboard shortcut set contained within the column.
- Highlighting identifies rows that differ:
 - Rows that exist in one file, but not the other, appear in green highlighting.
 - Rows that appear in both files, but that differ in content appear in pink highlighting.
- When multiple desktop tools support the same keyboard shortcut for a single desktop action, there is a row for each tool. For example, if both the MATLAB desktop and the Editor support the keyboard shortcut **Ctrl+W** for closing a selected window, a column of the Comparison Tool might appear like this:

51	Close	MATLAB Desktop	Ctrl+W	Closes the selected window
52	Close	MATLAB Editor	Ctrl+W	Closes the selected window

- When there are multiple keyboard shortcuts for the same action in a single tool, there is a row for each keyboard shortcut. For example, if there are two different keyboard shortcuts in the Editor for applying a code analyzer autofix, a column of the Comparison Tool might appear like this:

Autofix Message	MATLAB Editor	Alt+Enter	Applies the suggested autofix	11
Autofix Message	MATLAB Editor	Shift+F9	Applies the suggested autofix	12

- On Macintosh platforms, the textual format of keyboard shortcuts is slightly different from other platforms, and also differs from the representation shown on MATLAB desktop menus. These differences are due to the Macintosh platform displaying shortcuts using symbols. For instance, the Macintosh platform uses the symbol  for a keyboard key. Because the Comparison Tool represents symbols as text strings; it specifies the symbol  as CMD.

See also “Using Comparison Tool Features” on page 6-61.


Display Keyboard Shortcuts

The following sections describe the various ways you can display keyboard shortcuts:

- “List All Keyboard Shortcuts in a Set” on page 2-45
- “Display Keyboard Shortcuts on Menus” on page 2-46
- “Display Keyboard Shortcuts in the Preferences Dialog Box” on page 2-46

List All Keyboard Shortcuts in a Set

You can copy all the keyboard shortcuts from a keyboard shortcuts set and paste them in a text file or spreadsheet application, such as Microsoft Excel[®]. To create a list of keyboard shortcuts for easy browsing and future reference, follow these steps:

- 1** Choose **File > Preferences > Keyboard > Shortcuts**.
- 2** Click the Actions button .
- 3** From the drop-down menu, choose **Copy to Clipboard**.
- 4** Open a spreadsheet application or a text editor.

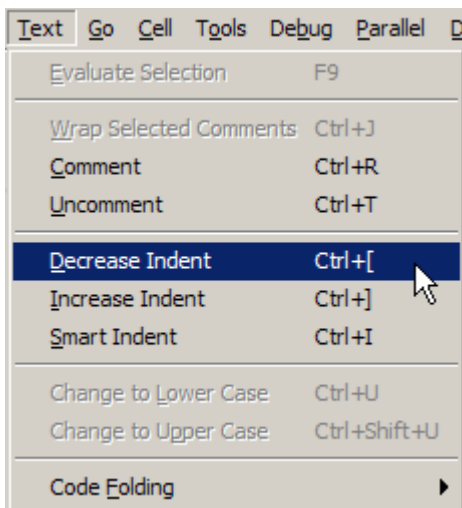
For the best formatting use a spreadsheet application.
- 5** Paste in the data from the clipboard.

In Microsoft Excel, for example, choose **Home > Paste**.

Display 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.

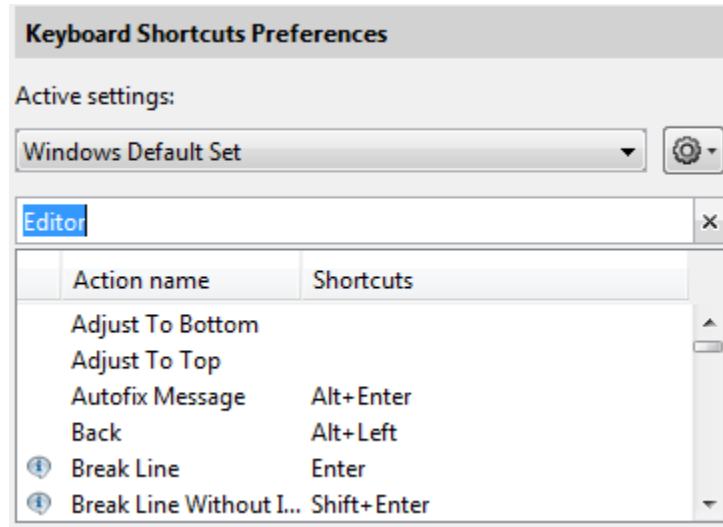


If no keyboard shortcut appears on the menu, one does not currently exist for that action. To create a keyboard shortcut for an action, follow the steps in “Customize Keyboard Shortcuts” on page 2-48.

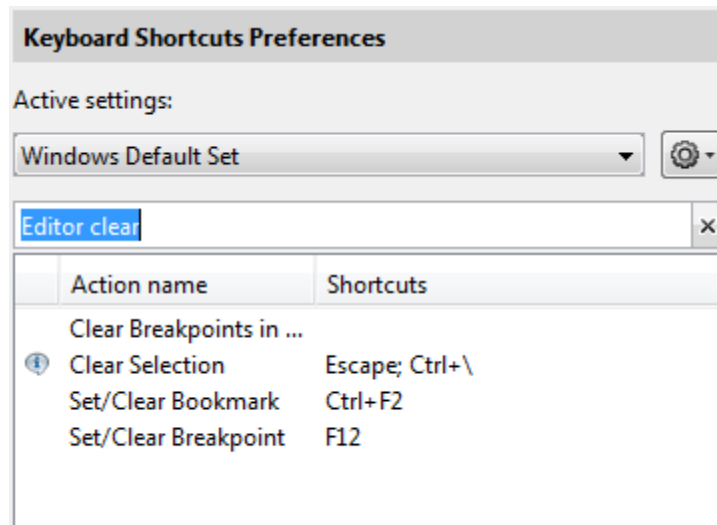
Display Keyboard Shortcuts in the Preferences Dialog Box

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

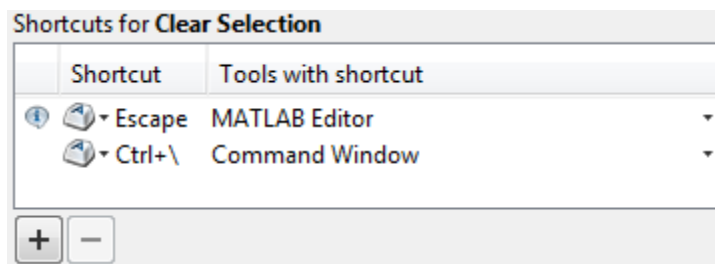
- 1 Choose **File > Preferences > Keyboard > Shortcuts**.
- 2 In the filter field, type the name of the tool for which you want to list the keyboard shortcuts. For example, type **Editor** to see the keyboard shortcuts currently defined for actions you can perform in the Editor.



- 3 Narrow the list of **Action names** that the preferences pane displays by adding a string describing the action. For example, add `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 Select the action name of interest. In this example, select **Clear Selection**.
- 5 View the table labeled **Shortcuts for Clear Selection**. It indicates that the **Escape** key is the current keyboard shortcut for the **Clear Selection** action in the Editor.



Shortcut	Tools with shortcut
Escape	MATLAB Editor
Ctrl+\	Command Window

Customize 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 2-49
- “Filter Keyboard Shortcut Actions” on page 2-51
- “Specify Keystrokes for a Keyboard Shortcut” on page 2-53
- “Evaluate and Resolve Keyboard Shortcut Conflicts” on page 2-54
- “Examples of Creating, Modifying, and Deleting Keyboard Shortcuts” on page 2-56
- “Display Keyboard Shortcuts” on page 2-45

Consider using File Exchange to share your active settings file with others. For more information, see “Submitting Your Files to the Repository” on page 7-40.

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.

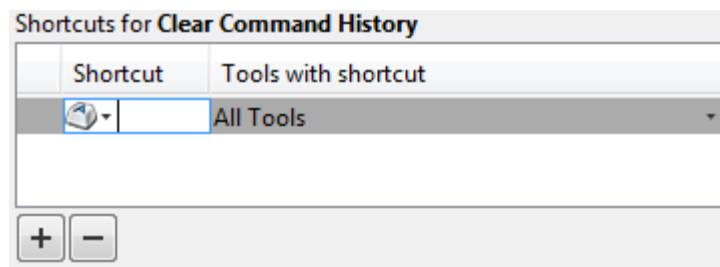
Typically, the first time you modify keyboard shortcuts, you begin with the default settings for your platform. For details, see “Choose a Set of Keyboard Shortcuts” on page 2-40.

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 “Filter Keyboard Shortcut Actions” on page 2-51.

4 Click the Add button .


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



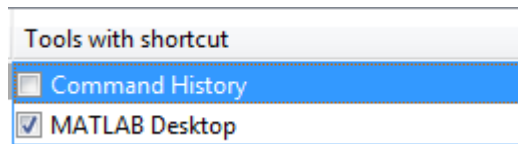
5 Type the shortcut that you want to use for the action you selected in Step 3. Alternatively, you can choose a shortcut from the drop-down menu.



For details, see “Specify Keystrokes for a Keyboard Shortcut” on page 2-53.

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 any conflicts, indicated by the informational  and error  icons.


For more information, see “Evaluate and Resolve Keyboard Shortcut Conflicts” on page 2-54.

- 8** Click **Apply**.

- The keyboard shortcut becomes available immediately.
- If a changed shortcut corresponds to a menu option that previously displayed no keyboard shortcut, MATLAB reflects the new keyboard shortcut on the menu.

Restore Default Keyboard Shortcut Sets

If you modify keyboard shortcuts, and then decide you do not want to keep the changes, you can restore the default shortcuts. To restore the default state of a keyboard shortcut:

- 1** Click the **Actions** button .
- 2** Select **Undo Modifications to Windows Default Set (modified)** or **Undo Modifications to Emacs Default Set (modified)**, as appropriate for your system.
- 3** Click **OK**.

Note Undoing modifications reverts all keyboard shortcuts changes that you made to the set. You cannot undo modifications on a shortcut-by-shortcut basis.

Save Keyboard Shortcuts to a Settings File

Save keyboard shortcuts to a settings file to:

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

MATLAB preserves changes you make to the default sets across sessions. However, if you undo modifications to a default keyboard shortcut set (as described in “Restore Default Keyboard Shortcut Sets” on page 2-50) you lose all changes, unless you first save them to a new set.


- Copy the keyboard shortcuts settings file to another system running MATLAB and use it there.
- Overwrite a settings 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 a keyboard shortcuts settings file with others.

For information on sharing your active settings file with the MATLAB community, see “Submitting Your Files to the Repository” on page 7-40.

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 **Actions** button , and then select **Save As**.
- 3 In the Save dialog box, navigate to the folder where 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 you specified.

Filter Keyboard Shortcut Actions

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

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 a list of the action names associated with the tool or menu you specify. In addition, the list 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 keystrokes or typing the key names.

For example, to enter **Ctrl+S**, use the keystroke (by pressing the **Ctrl** key and the **S** key). Or, type **Ctrl+S** character by character (**C-t-r-l-+-Y**).

- If using keystrokes for a keyboard shortcut does not work, try typing the characters instead. You *must* type some keyboard shortcuts character by character, such as shortcuts including the **Tab**, **Backspace**, or **Delete** keys.
- Type **numpad** to refer to the number pad that is on the far right of some keyboards.
- Type **Up** or **Down** to refer to the **Up arrow** or **Down arrow** keypad keys, respectively.

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.

Action name	Shortcuts
Remove	
Remove Next Word	Ctrl+Delete
Remove Previous Word	Ctrl+Backspace

Deletes the next word


Specify Keystrokes for a Keyboard Shortcut

A *keystroke* can be a single key or the combination of a modifier (**Alt**, **Shift**, or **Ctrl**) and another key. When you create a keyboard shortcut, specify the keystrokes for the 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 number of keystrokes, which is one keystroke, skip to step 3.
 - To specify multiple keystrokes, or to specify explicitly one keystroke follow these steps:
 - a Click the down arrow next to the key icon  in the **Shortcuts** field.
 - b 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* by typing the key names character by character.



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 already have a defined action within dialog boxes. For example, the **Tab** key navigates from one field to the next in dialog boxes.

Evaluate and Resolve Keyboard Shortcut Conflicts


Conflicts arise when two or more different actions have the same shortcut. 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 2-54.
-  —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 2-55.

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.

Although not evident from the preferences pane, **Ctrl+C** presents a similar situation on Windows systems. **Ctrl+C** is the keyboard shortcut for interrupting MATLAB execution. However, the default keyboard shortcut for the copy action is also **Ctrl+C**. Therefore, 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.

Resolve 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 2-57 and “Deleting a Keyboard Shortcut” on page 2-58.

Examples of Creating, Modifying, and Deleting Keyboard Shortcuts

- “Creating a New Keyboard Shortcut” on page 2-56
- “Changing a Keyboard Shortcut” on page 2-57
- “Deleting a Keyboard Shortcut” on page 2-58

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:

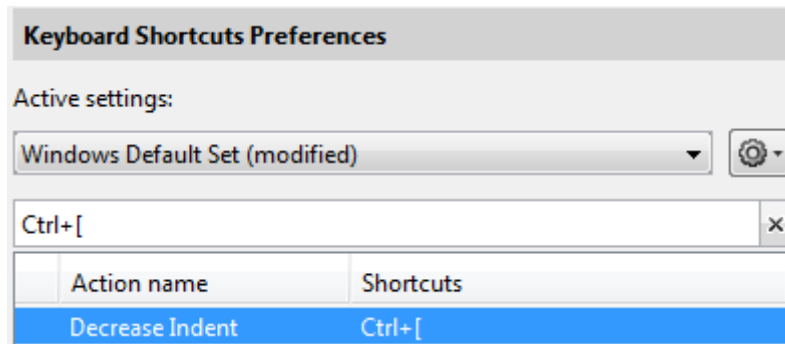
- The Add to Favorites dialog box opens when you press **Ctrl+S, Alt+V** in the Help browser.
- **Ctrl+S, Alt+V** appears next to **Add to Favorites** when you click the **Favorites** menu in the Help browser.

Changing a Keyboard Shortcut

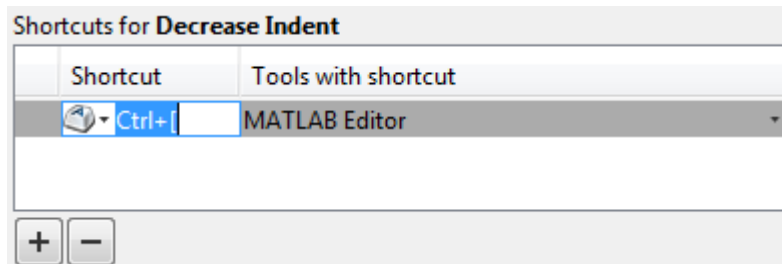
Suppose you frequently adjust indenting in the MATLAB Editor. However, you have difficulty remembering the default keyboard shortcut of **Ctrl+[** for decreasing the indent. So, you decide to change it to something that is easier to remember.

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+[**.
- 4 Under **Action name**, select **Decrease Indent**.



- 5 In the table labeled **Shortcuts for Decrease Indent**, under **Shortcuts**, click **Ctrl+[**. MATLAB makes the field editable.



- 6 In the **Shortcut** field, press **Ctrl+Backspace** twice.

The first time you press the key combination, it deletes **Ctrl+[**. The second time you press it, **Ctrl+Backspace** appears in the field.

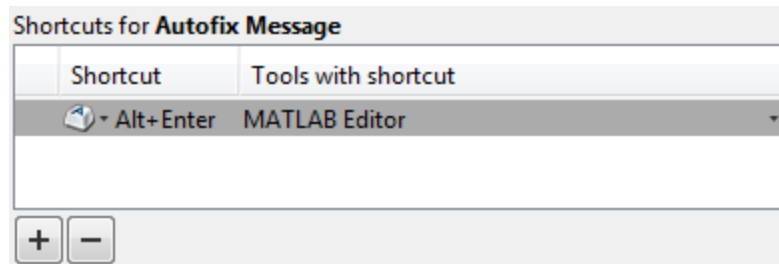
- 7 Click **Apply**.


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, on Windows, you press **Alt+Enter** (to apply a code analyzer autofix) instead of **Ctrl+Enter** (to evaluate the current cell in the MATLAB Editor). To avoid accidentally applying an autofix, delete the **Alt+Enter** shortcut by following 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 Message**.
- 5 In the next table, under **Shortcuts for Autofix Message**, 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.

See also “Delete a Set of Keyboard Shortcuts” on page 2-59.


Delete a Set of Keyboard Shortcuts

If you previously saved or copied a set of keyboard shortcuts to your system and you no longer want it, delete it as follows:

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

2 Under **Active settings**, choose the set of keyboard shortcuts that you want to delete.

You cannot delete default keyboard shortcut sets, such as `Windows Default Set`.

3 Click the Actions button  and choose `Delete filename`, where *filename* is the name of a keyboard shortcut set you previously saved or copied to your system.

For information on deleting a single keyboard shortcut from a set that you want to keep, see “Deleting a Keyboard Shortcut” on page 2-58.

Use 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 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, see “Submitting Your Files to the Repository” on page 7-40 and “Get Files from the File Exchange Repository” on page 7-31.

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” on page 2-60
- “Actions for Which You Cannot Customize Keyboard Shortcuts” on page 2-61

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.	<p>Esc (escape)</p> <p>For example, if you select the Edit menu, the menu items display. Pressing Esc retracts the menu items.</p> <p>In the Function Browser, pressing Esc up to three times has the following effects:</p> <ol style="list-style-type: none"> 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)

Action	Keyboard Shortcut
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
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: <ul style="list-style-type: none">• Command Window: Ctrl+0• Command History: Ctrl+1• Current Folder: Ctrl+2• Editor: Ctrl+Shift+0• Figures: Ctrl+Shift+1• Figure Palette: Ctrl+7• Comparison Tool: 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

Basic Operations

In this section...
“Cut, Copy, and Paste” on page 2-63
“Print Options” on page 2-64
“Select Multiple Items” on page 2-66
“Add Your Own Toolboxes to the Start Button” on page 2-67

Cut, Copy, and Paste

The following table summarizes the various methods for cutting, copying, and pasting data.

To	Do This
Cut a selection from a desktop tool.	Select the text you want to cut, right-click, and then select Cut .
Copy a selection from a desktop tool to the clipboard.	Select the text you want to copy, right-click, and then select Copy .
Paste a selection from the clipboard into a desktop tool.	Copy the text you want to paste, move the mouse pointer to the paste location, right-click and then select Paste .
Copy data from another application (such as Microsoft Excel) and import into MATLAB.	<ol style="list-style-type: none"> 1 Copy data from the other application onto the clipboard. 2 Select Edit > Paste to Workspace. The Import wizard opens. 3 Follow the steps through the Import wizard. For details, see “Tips for Using the Import Wizard”.

For information on copying and pasting programmatically, see the `clipboard` function.

Print Options

- “Print from Desktop Tools” on page 2-64
- “Page Setup Options” on page 2-64
- “Layout Options for Page Setup” on page 2-65
- “Header Options for Page Setup” on page 2-65
- “Fonts Options for Page Setup” on page 2-66

Print from 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 disabled 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:

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 the element.

For example, if you access this dialog box while using the Command Window, you can select **Use Command Window font** for **Body** text. The printed text matches the Command Window font.

- 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.

Tip To change the font that a desktop tool uses, select **File > Preferences > Fonts > Custom**.

Select 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.

Add Your Own Toolboxes to the Start Button

- “Basic Steps” on page 2-67
- “More About Adding Items to the Start Button” on page 2-68

Basic Steps

If you author a collection of MATLAB program files or Simulink blocks, you can provide access to its features through the MATLAB desktop  button. A collection of MATLAB program files is called a *toolbox*. A collection of Simulink blocks is called a *blockset*.

MATLAB determines the information to display on the **Start** button using `info.xml` files that are in folders on the search path. These XML files also contain information to place entries for toolbox and blockset documentation in the **Contents** pane of the Help browser. To add your toolbox to the Start button:

- 1** Make the folder containing your toolbox files your current folder. Use the `cd` command or navigate to the toolbox folder in the Current Folder browser.
- 2** Copy the template file `info_template.xml` to one you name `info.xml` in your current folder. If you have set up HTML files for your toolbox, you have already created an `info.xml` file for it. See “More About Adding Items to the Start Button” on page 2-68 for details.
- 3** Add a `<list>` `</list>` element as the last element within `<productinfo>`. Put the `<list>` after the `<help_location>` element, if any.

Within the `<list>`, add a `<listitem>` for each item you add to the **Start** menu for the toolbox. For example:

```
<list>
  <listitem>
```

```

<!-- The label provides the text for this menu item -->
<label>MyToolbox Documentation</label>
<!-- This callback is a command to open your documentation -->
<callback>web ./helpfiles/mytoolbox_product_page.html -helpbrowser</callback>
<!-- Menu item icon (a toolbox icon from the help browser ) -->
<icon>$toolbox/matlab/icons/bookicon.gif</icon>
</listitem>
<listitem>
...
</listitem>
...
</list>

```

Each `<callback>` element contains a complete MATLAB command specifying the action triggered by selecting that menu item. `<icon>` elements identify **Start** menu item icons. For a table of standard icons that you can use, see “More About the helptoc.xml File”.

- 4 Include the `info.xml` file in your toolbox folder, along with your toolbox program files and documentation files.
- 5 Add the toolbox folder to the search path. Refresh the **Start** button, if a **Toolboxes** item does not appear.
- 6 If you are providing the toolbox to others, instruct them to perform the previous step themselves after they start MATLAB .

If you also include HTML documentation for your toolbox for display in the Help browser, put the Help browser and **Start** button XML code into a single `info.xml` file.

More About Adding Items to the Start Button

The template XML file called `info_template.xml` includes code to add menu items for a toolbox to the **Start** menu. This directory is `matlabroot/help/techdoc/matlab_env/examples/templates`. `<icon>` elements in that file specify icon image files to use in the **Start** menu. You can use an example icon image file, `sampleicon.gif`, contained in the `/examples/templates` folder. To display a custom icon, substitute your own image file name for `sampleicon.gif` in your `info.xml` file. Put your icon file in the same folder that `info.xml` occupies.

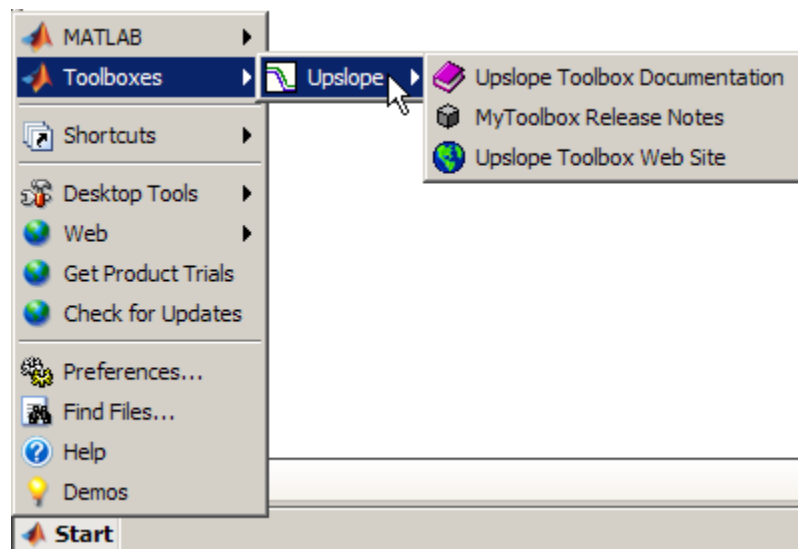
For instructions on copying and using the sample `info_template.xml` file, see “Identifying a Help Folder: the `info.xml` File”. This file contains a `<list>` element that you can modify to customize what appears on the **Start** button for your toolbox. You can remove `<list>` elements or include additional ones.

When you have created a `info.xml` file in your toolbox folder containing `<list>` elements for your **Start** menu items (or have copied the Upslope Toolbox example files, which include a complete `info.xml` file):

- 1 Add the folder containing `info.xml` to the search path. The folder cannot be the current folder when you add it to the path.

See “Adding Folders to the Search Path” on page 6-73.

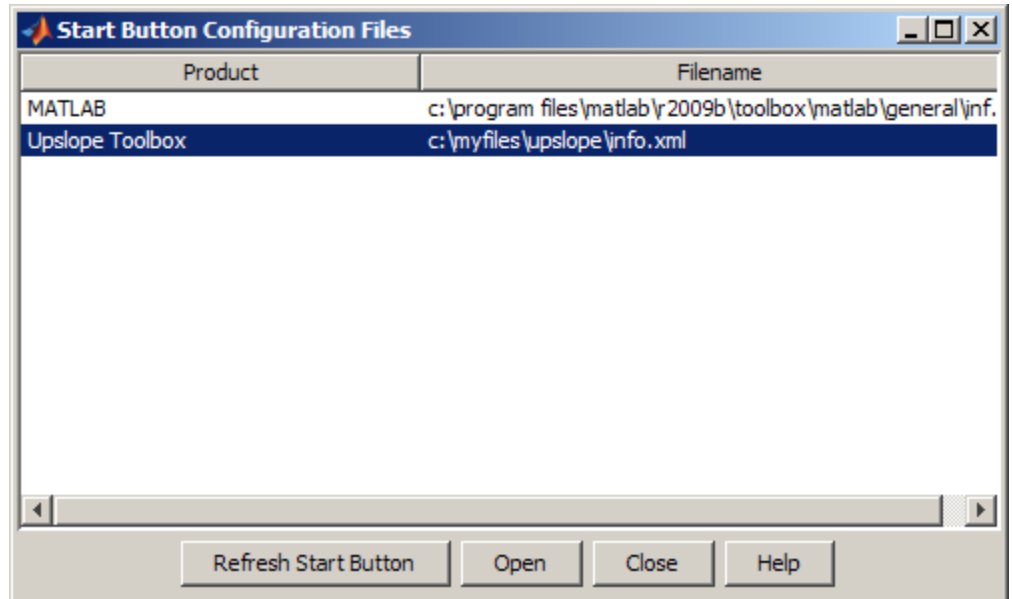
- 2 Click the **Start** button and then click the **Toolboxes** item. If you set up everything properly, you see an entry your toolbox, similar to the following illustration.



If you do not see your toolbox on the **Start** button, you might need to refresh the **Start** button so it can locate your `info.xml` file. Try the following:

- a Select **Start > Desktop Tools > View Start Button Configuration Files**.

The Start Button Configuration Files dialog box opens.



- b** Click **Refresh Start Button**.
 - c** Click **Close** to close the dialog box.
- 3** Ensure there are no errors. If, when interpreting your `info.xml` file, MATLAB detects an invalid construct, it displays an error message in the Command Window. For more information, see “Address Validation Errors for `info.xml` Files”.

For more information about setting up an `info.xml` file, see “Add Documentation to the Help Browser”.

Web Browsers and MATLAB

In this section...
“About Web Browsers and MATLAB” on page 2-71
“Display Pages in Web Browsers” on page 2-73
“Specify Proxy Server Settings for Connecting to the Internet” on page 2-73
“Specify the System Browser for UNIX Platforms” on page 2-75

About Web Browsers and 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 the different browsers 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 a MATLAB program file to HTML, the HTML file displays in the MATLAB Web Browser:

The screenshot shows a web browser window with the title "Square Waves from Sine Waves". The address bar shows the file path: `file:///I:/my_matlab_files/my_mfiles/html/fourier_demo.html`. The page content includes:

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);
```

The plot shows a square wave approximation using a sine wave. The x-axis ranges from 0 to 4π , and the y-axis ranges from 0 to 1. The plot shows a blue sine wave that is zero for the first half of each period and reaches a peak of 1 for the second half. The plot is displayed in a window with a scrollbar.

MATLAB Web and Help Browsers

The MATLAB Web and Help browsers 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.

Display 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 any file type 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 Proxy Server Settings for Connecting to the Internet

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

- MATLAB supports non-authenticated, basic, digest, and NTLM proxy authentication types.
- 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.

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 an integer only, 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.

Specify the System Browser for UNIX Platforms

To specify the system browser:

- 1 Select **File > Preferences > Web**.
- 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.

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

- 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**.

License Management and Software Updates

In this section...

“Manage Your Licenses” on page 2-76

“Check for Software Updates” on page 2-77

Manage 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 “Specify Proxy Server Settings for Connecting to the Internet” on page 2-73 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

Option	Description
	<p>the MathWorks Web site. You will not be able to use MathWorks software with that license on this computer.</p> <p>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 <i>deactivation string</i>. 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.</p>
Update Current Licenses	<p>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 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.</p>
Manage Licenses	<p>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.</p>

Check for Software Updates

To determine if more recent versions of your MathWorks products are available, and to view latest version numbers for all MathWorks products, follow these steps:

- 1 Make sure you have an active Internet connection.
- 2 Select **Help > Check for Updates**. The Check for Updates dialog box displays.
- 3 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 display.

4 Click any column heading to sort or reverse the sort order by that column.

5 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.

6 Decide whether you want to upgrade to the most recent version.

- If you do, click **Download Products at MathWorks.com**
- If you do not, go to step 7.

7 Click **Close**.

Macintosh Platform Conventions

In this section...

“Mouse Instructions and Macintosh Platforms” on page 2-79

“Navigating Within the MATLAB Root Folder on Macintosh Platforms”
on page 2-79

Mouse Instructions and Macintosh Platforms

The documentation typically presents conventions for Microsoft Windows platforms. Therefore, some conventions and operations differ on the Macintosh platform from those that appear in the rest of the documentation. The intended action for the Macintosh platform is typically obvious. Mouse operations follow Macintosh conventions.

Make the following replacements to adjust documented mouse instructions for Macintosh platforms if you are using a one-button mouse:

- Replace right-click with **Ctrl**+click
- Replace middle-click with **Command**+click

Navigating Within the MATLAB Root Folder on Macintosh Platforms

On Macintosh platforms, MATLAB is installed as an application bundle. The root folder, `MATLAB_R2012a`, has a `.app` extension.

To view the contents of the MATLAB root folder in the Mac Finder, right-click the MATLAB application bundle, and then select **Show Package Contents** from the context menu.

To view the content of the MATLAB root folder from within MATLAB:

- 1 Select **File > Open**.
- 2 In the File Browser dialog box, press **Command+Shift+G** to open the Go To Folder dialog box.

3 Enter the full path to the MATLAB folder, for example,
/Applications/MATLAB_R2012a.app.

4 Press **Go**.

To open a file with the MATLAB command, such as `edit`, specify the full path of the MATLAB root folder. For example:

```
edit /Applications/MATLAB_2012a.app/toolbox/matlab/demos/lotka.m
```


Accessibility

In this section...
“Software Accessibility Support” on page 2-81
“Documentation Accessibility Support” on page 2-82
“Assistive Technologies” on page 2-83
“Installation Notes for Accessibility Support” on page 2-84
“Troubleshooting” on page 2-87

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 2-83
- 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, 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.

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/help>

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. Command line help on functions is accessible, however.
- 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

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 MathWorks has not tested other assistive technologies, such as other screen readers or ZoomText® Xtra screen magnifier from Ai Squared, 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, 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 Java Access Bridge to your Windows path.
- 2 Create the `accessibility.properties` file.

These tasks are described in more detail below.

Add Java Access Bridge Software to Your Path. To add Java Access Bridge to your Windows path:

- 1 On your Windows system, locate the Environment Variables dialog box.

The location varies, depending on the version of Windows you are running.

- 2 Under **System variables**, select the Path option.
- 3 Click the **Edit** button.
- 4 At the end of the Path environment variable:
 - a See if a semicolon appears at the end of the existing variable value. If not, add one.
 - b Add the folder that contains `matlab.exe`; for example:

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

- 5 Click **OK** three times.
- 6 Restart your system.
- 7 Restart the JAWS software.

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

8 When the JAWS software indicates it is ready, restart 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 file name `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, 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 folder 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 folder 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.

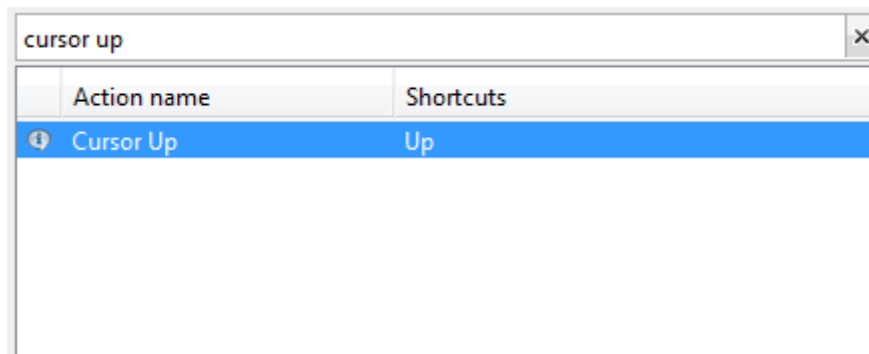
You can check the shortcut setting for the **Up Arrow** key simply by pressing this key while in the Command Window. If the cursor moves up on the screen, then your settings are correct and this is not the cause of the problem. If the **Up Arrow** key displays your most recent command, or performs any 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 2-90 in the documentation below.

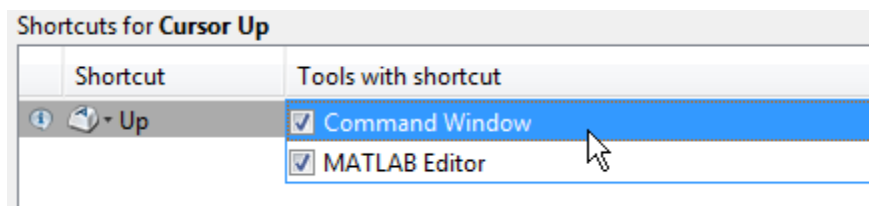
For more information on either of the procedures described below, see “Keyboard Shortcuts” on page 2-36.

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. MATLAB saves your settings and restores them in your next MATLAB session.

- 1** Select **File > Preferences > Keyboard > Shortcuts**.
- 2** In the search field (with dimmed text reading “Search by action name or shortcut”) enter the words “cursor up”, and then click the line that shows the action-to-shortcut key association.

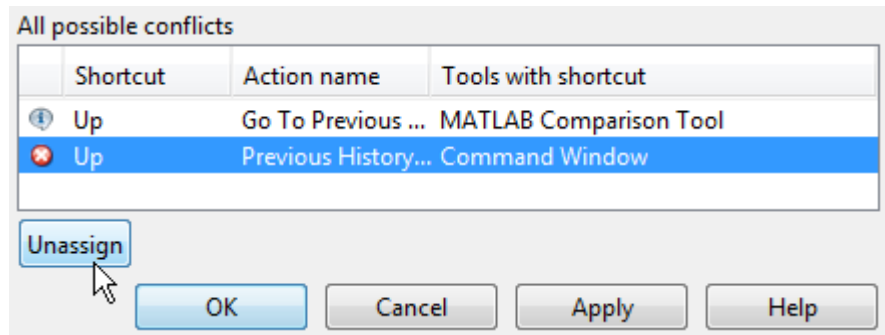


- 3 In the **Shortcuts for Cursor Up** panel click MATLAB Editor. Notice that the check box is selected for MATLAB Editor, but not Command Window.
- 4 Select Command Window.



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

- 5 The **All possible conflicts** panel shows that the shortcut you have just established conflicts with an existing shortcut in the Command Window. The Command Window now has 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:

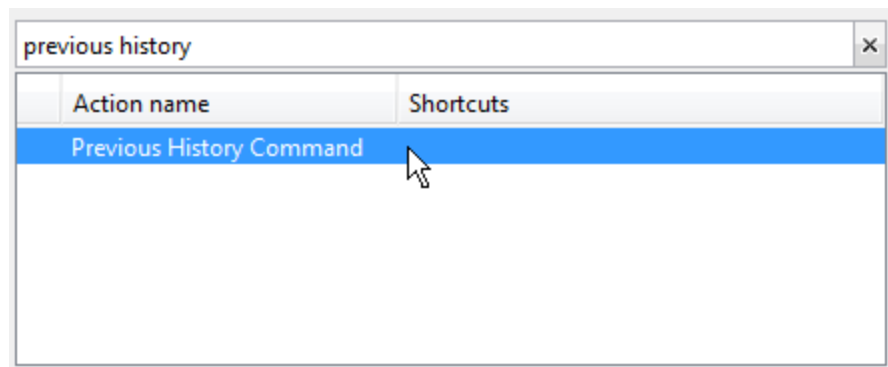



Click **Apply** to make this setting active.

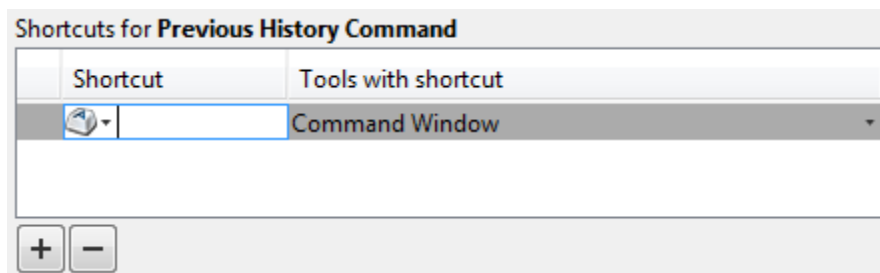
- 6 Repeat steps 2 through 5 to assign the **Down Arrow** key to the **Cursor Down** command in the **Command Window**. This also removes the existing shortcut between **Down Arrow** and **Next History** command.

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:

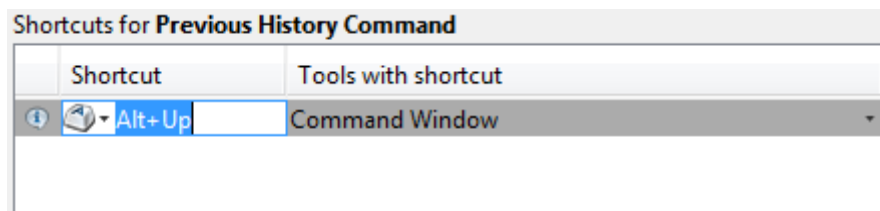
- 1 Type “previous history” in the box above **Action name**.
- 2 Click the line below **Action name** that reads **Previous History Command**.



- 3** Open the key entry field in the **Shortcuts for Previous History Command** panel by clicking the plus  button.



- 4** Press the key or key combination you want to use as the shortcut for the Previous History action. For example, hold down the **Alt** key and press the **Up Arrow** key.



- 5** 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 “Evaluate and Resolve Keyboard Shortcut Conflicts” on page 2-54.
- 6** 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.

About Preferences

In this section...

“Set Preferences for MATLAB” on page 2-93

“Where MATLAB Stores Preferences” on page 2-94

“Preferences Folder and Files MATLAB Uses When Multiple MATLAB Releases Are Installed” on page 2-95

“General Preferences” on page 2-96

“MAT-Files Preferences” on page 2-97

“Confirmation Dialogs Preferences” on page 2-98

“Source Control Preferences” on page 2-100

“Java Heap Memory Preferences” on page 2-100

“Keyboard Shortcuts Preferences” on page 2-101

“Fonts Preferences” on page 2-103

“Fonts Custom Preferences” on page 2-104

“Colors Preferences” on page 2-105

“Colors Programming Tools Preferences” on page 2-106

“Toolbars Preferences” on page 2-107

“Web Preferences” on page 2-108

Set Preferences for MATLAB

MATLAB provides a variety of options called *preferences* for customizing MATLAB. To access and set preferences:

- 1** Select **File > Preferences**.
- 2** From the left pane of the Preferences dialog box, select a tool, product, or an entry revealed when you click a plus sign (+) preceding a tool or product name.
- 3** Change settings in the right pane of the Preferences dialog box.
- 4** Click **Apply** or **OK**.

Preferences take effect immediately. They remain persistent across sessions of MATLAB.

Function Alternative

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

Where MATLAB Stores Preferences

MATLAB and other MathWorks products store their preferences in the file `matlab.prf`. This file loads when you start MATLAB. The folder containing this file is called the preferences folder. The preference folder also contains other related files.

The Path to and File Name for the Preferences Folder

To see the full path for the folder where `matlab.prf` and related files are located, type `prefdir` in the MATLAB Command Window.

On Apple Macintosh platforms, the folder can be in a hidden folder, for example, `myname/.matlab/R2009b`. If so, to access the hidden folder:

- 1** In the Apple Mac OS® Finder tool, select **Go > Go to Folder**.
- 2** In the resulting dialog box, type the path returned by `prefdir`, and then press **Enter**.

The name of the preferences folder, matches the name of the release. For instance, for MATLAB R2010b, the name of the preferences folder is `R2010b`.

Effects of Changing Preferences

When you change preferences using the MATLAB Desktop (**File > Preferences**), it updates `matlab.prf`. When you close MATLAB, it saves those changes to `matlab.prf`.

Effects of Installation and Deinstallation on the Preferences Folder

Installing MATLAB has no effect on the preferences folder. That is, MATLAB creates, checks, copies, and writes to the preferences folder when you start

up MATLAB, not when you install it. When you deinstall MATLAB, there is an option in the uninstaller to remove the preferences folder. However, this option is not selected by default.

Preferences Folder and Files MATLAB Uses When Multiple MATLAB Releases Are Installed

The files in the preferences folder that MATLAB uses depends on the version of MATLAB you are starting up. How and if MATLAB migrates (reuses) preferences files from one version to the next also depends on the version.

Process MATLAB Uses to Create and Migrate the Preferences Folder and its Files

When you start it up, MATLAB looks for a preferences folder name that matches the release starting up, and then does one of the following:

- If MATLAB finds a preferences folder name matching the release starting up, it uses that folder and the files within it.

If that folder is empty, MATLAB recreates the default files for the release starting up.

- If MATLAB does not find a preferences folder name matching the release starting up, it creates one. Then, MATLAB checks to see if the release of MATLAB that immediately precedes the one you are starting up is installed.
 - If that previous release is not installed, MATLAB recreates the folder and default files for the version starting up.

For example, if you start up R2010b and R2010a is not installed, then MATLAB recreates the default files for the R2010b preferences folder. This is true even if R2009b or earlier is installed.

- If that previous release is installed, MATLAB migrates the files from the preferences folder corresponding to that previous release to the preferences folder for the release starting up.

For example, if you start up R2010b and R2010a is installed, then MATLAB migrates the files from R2010a preferences folder to the R2010b preferences folder.

Control the Preferences Files MATLAB Uses

This table describes how to control which versions of preferences files MATLAB uses.

To Use:	Do This:
Default preference files for a given release of MATLAB	Make sure the preferences folder for that release exists, but is empty before starting up that MATLAB version.
All the preference files from the release of MATLAB immediately preceding the release you plan to start up.	Ensure that the preferences folder exists for that preceding release. If so, delete the entire preferences folder for the release of MATLAB you plan to start up.
The release-specific default for just a particular file in the preferences folder	Delete just that file from the preferences folder for the release of MATLAB you plan to start up. One file to consider keeping is <code>history.m</code> . For more information, see “Command History File — <code>history.m</code> ” on page 3-4.

General Preferences

You can set preferences for toolbox path caching, figure window printing, and deleting files.

Select **File > Preferences > General**, and then adjust preference options as described in this table.

Preference	Usage
Toolbox path caching	Select Enable toolbox path caching to have MATLAB cache toolbox folder information across sessions for quicker startup performance.
	Select Enable toolbox path cache diagnostics to display information about startup time when you start MATLAB.
	Click Update Toolbox Path Cache to add files to the toolbox folders under the <code>matlabroot</code> folder. (Use after you use tools not provided with MATLAB to create MATLAB files.) For details, see “Toolbox Path Caching in the MATLAB Program” on page 1-23.
Figure window printing	Select an option to specify how colored lines and text appear in printed output.
Deleting Files	Select an option to specify what MATLAB does with files you delete using the <code>delete</code> function. Selecting Delete permanently makes the <code>delete</code> function run faster. For details, see “Deleting Files and Folders Using Functions” on page 6-42.

MAT-Files Preferences

You can set the default MATLAB version for MAT-files and FIG-files. These preferences apply to both the `save` function and the **Save** menu options. However, the `matfile` function creates only Version 7.3 MAT-files.

To set MAT-Files preferences, select **File > Preferences > General > MAT-Files**, and then adjust preference options as described in the table below.

For more details on the features supported in each version, see the [save](#) reference page.

Option	Use to:
MATLAB Version 7.3 or later (save -v7.3)	Load or save parts of variables, or save variables larger than 2 GB on 64-bit systems. As with Version 7, files are compressed and use Unicode® character encoding.
MATLAB Version 7 or later (save -v7)	Save compressed MAT-files that use Unicode character encoding. This is the default on new installations of MATLAB software and upgrades from versions earlier than 7.3.
MATLAB Version 5 or later (save -v6)	Save MAT-files for use with versions prior to MATLAB Version 7, or create uncompressed files.

Confirmation Dialogs Preferences

You can specify whether or not MATLAB displays specific confirmation dialog boxes.

Select **File > Preferences > General > Confirmation Dialogs**, and then adjust preference options as described in the table below.

This table summarizes the core MATLAB confirmation dialog boxes. There might be additional confirmation dialog boxes for other products you install.

Option	Confirmation Dialog Box Appears
Warn before deleting Command History items	When you delete entries from the Command History window. For details, see “Delete Entries from the Command History Window” on page 3-34 .
Warn before clearing the Command Window	When you clear the Command Window content by selecting Edit > Clear Command Window . Does not appear when you use the <code>clc</code> function.

Option	Confirmation Dialog Box Appears
Confirm when overwriting variables in MAT-files	<p>When you save variables by dragging them from the Workspace browser onto a MAT-file in the Current Folder browser.</p> <p>For details, see “Creating and Updating MAT-Files with the Current Folder Browser” on page 6-36.</p>
Prompt when editing files that do not exist	<p>When you type <code>edit filename</code> and <code>filename</code> does not exist in the current folder or on the search path.</p>
Prompt to exit debug mode when saving file	<p>When you try to save a modified file while in debug mode.</p> <p>For details, see “End Debugging” on page 8-133.</p>
Prompt to save on activate	<p>When you have unsaved changes to a figure and program file and you activate the GUI by clicking the Run button, for example.</p> <p>For details, see “GUIDE Preferences”.</p>
Prompt to save on export	<p>When you have unsaved changes to a figure and program file and you select File > Export.</p> <p>For details, see “GUIDE Preferences”.</p>
Confirm changing default callback implementation	<p>When you have modified a callback signature in GUIDE.</p> <p>For details, see “GUIDE Preferences”.</p>
Confirm before exiting MATLAB	<p>When you quit MATLAB.</p> <p>For details, see Quitting MATLAB.</p>
Warn about missing search databases	<p>When you have help files in the Help browser for products not produced by MathWorks and the search database for those files has not been updated for the version of MATLAB you are running.</p> <p>For more information, 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.</p>
Confirm when deleting variables	<p>When you delete variables from the workspace using menu items. Does not appear with the <code>clear</code> function.</p> <p>For details, see “Deleting Workspace Variables” on page 5-8.</p>

Source Control Preferences

You can select which previously installed and configured source control system to use with MATLAB.

Select **File > Preferences > General > Source Control**, and then select an option from the drop-down menu.

For detailed information on setting up and using a source control system with MATLAB, see “Setting Up the Source Control Interface on Microsoft Windows” on page 12-3 and “Source Control Interface on UNIX Platforms” on page 12-26.

Java Heap Memory Preferences

You can adjust the amount of memory that MATLAB software allocates for Java objects.

Note The default heap size is sufficient for most cases.

To adjust the Java heap size:

- 1** Select **File > Preferences > General > Java Heap Memory**.
- 2** Select a Java heap size value using the slider or spin box.

Note Increasing the Java heap size decreases the amount of memory available for storing data in arrays.

- 3** Click **OK**.
- 4** Restart MATLAB.

If the amount of memory you specified is not available upon restart, MATLAB resets the value to the default, and displays an error dialog box. To readjust the value, repeat the previous steps.


If increasing the heap size does not eliminate memory errors, check your Java code for memory leaks. Eliminate references to objects that are no longer useful. For more information, see the Java SE Troubleshooting guide at <http://www.oracle.com/technetwork/java/javase/index-138283.html>.


Keyboard Shortcuts Preferences

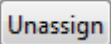
You can set keyboard shortcuts for actions you perform using MathWorks software. You can specify or import sets of predefined keyboard shortcuts, set individual shortcuts on an action-by-action basis, or use a combination of both approaches.

Select **File > Preferences > Keyboard > Shortcuts**, and then adjust preference options as described in the table below.

For step-by-step instructions, see “Customize Keyboard Shortcuts” on page 2-48.

Preference	Usage
Active settings	Select or import a set of predefined keyboard shortcuts. For details, see “Choose a Set of Keyboard Shortcuts” on page 2-40 and “Use Keyboard Shortcuts Settings Files Created on Other Systems” on page 2-60.
	Select any one of these options: <ul style="list-style-type: none"> • Save As—Save active settings to a file. • Copy to clipboard— so you can import into Microsoft Excel, for example. For details, see, “Display Keyboard Shortcuts” on page 2-45. <ul style="list-style-type: none"> • Compare active settings to another set.

Preference	Usage
	<p>For details, see, “Compare Sets of Keyboard Shortcuts” on page 2-43.</p> <ul style="list-style-type: none"> • Undo Modifications to a default keyboard shortcut set. • Delete a set of keyboard shortcuts you previously saved or added. <p>For details, see “Delete a Set of Keyboard Shortcuts” on page 2-59.</p>
<p>Search by action name or shortcut</p>	<p>Search the list of displayed actions.</p>
<p>Shortcuts for <action-name></p>	<p>View the keyboard shortcut assigned to a selected action.</p>
	<p>Add or delete a keyboard shortcut to a selected action.</p> <p>For details, see, “Examples of Creating, Modifying, and Deleting Keyboard Shortcuts” on page 2-56.</p>

Preference	Usage
All possible conflicts	Display conflicts when two or more different actions have the same shortcut. For details, see “Evaluate and Resolve Keyboard Shortcut Conflicts” on page 2-54.
	Remove the keyboard shortcut from the selection in the All possible conflicts list. For details, see “Evaluate and Resolve Keyboard Shortcut Conflicts” on page 2-54.

Fonts Preferences

You can set and specify different fonts for desktop tools.

Select **File > Preferences > Fonts**, and then set options as described in the table below.

For step-by-step instructions on setting these preferences, see “Fonts” on page 2-2.

Preference	Usage
Desktop code font	Specify the font (type, style, and size in points) for tools assigned to use the desktop code font. To change the list of tools that use the desktop code font, see “Fonts Custom Preferences” on page 2-104.
Desktop text font	Specify the font (type, style, and size in points) for tools assigned to use the desktop text font. To have the desktop text font use the same font as your system, select Use system font . Otherwise, clear Use system font and specify the type, style, and size in points in the fields provided. To change the list of

Preference	Usage
	tools that use the desktop text font, see “Fonts Custom Preferences” on page 2-104.
Custom fonts	See which tools currently do not use the desktop text font or the desktop code font. By default, HTML proportional text uses a custom font.
Use antialiasing to smooth desktop fonts Linux ⁴ and UNIX ⁵ platforms only.	Give the desktop a smoother appearance. This option is not provided on Microsoft Windows or Apple Macintosh platforms, because MATLAB follows the operating system’s font settings on these platforms.

Fonts Custom Preferences

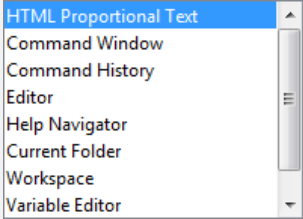

You can specify the font for HTML Proportional Text, and to override font settings for individual desktop tools, as described in the table that follows. Desktop tools otherwise use the settings that the “Fonts Preferences” on page 2-103 specify.

Select **File > Preferences > Fonts > Custom**, and then set options as described in the table below.

For step-by-step instructions on setting these preferences, see “Fonts” on page 2-2.

4. Linux is a registered trademark of Linus Torvalds.

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

Preference	Usage
Desktop tools	<p>Select HTML Proportional Text or the desktop tool for which you want to view or customize fonts.</p> 
Font to use	<p>Indicates the font currently being used in the selected desktop tool. Use one of these fonts to change it.</p> <ul style="list-style-type: none"> • Desktop code Uses the characteristics of the desktop code font, as described in “Fonts Preferences” on page 2-103. • Desktop text Uses the characteristics of the desktop text font, as described in “Fonts Preferences” on page 2-103. • Custom Uses the type, style, and size you specify in the fields.  <p>If you change the font style (for example, to bold or italic) for HTML Proportional Text, it has no effect. However, if you change the font size HTML Proportional Text, it affects both code and text for tools using the HTML Proportional Text font.</p>

Colors Preferences

You can specify the text and background color for desktop tools, as well as colors for highlighting syntax elements of MATLAB code.

Select **File > Preferences > Colors**, and then set options as described in the table below.

Preference	Usage
<p>Desktop tool colors</p>	<p>Specify that desktop tools use the same text and background colors that your platform uses for other applications by selecting Use system colors.</p> <p>Customize colors by clearing Use system colors, and then choose Text and Background colors from the drop-down menus .</p> <p>These colors do not apply to the Help display pane, nor to the Web Browser.</p> <p>For details, see “Changing Text, Background, and Hyperlink Colors in Desktop Tools” on page 2-8.</p>
<p>MATLAB syntax highlighting colors</p>	<p>Set colors to help you quickly identify elements of MATLAB syntax in the Editor, Command Window, Command History window, and the MATLAB shortcuts callback area.</p> <p>For details, see “Changing Syntax Highlighting Colors” on page 2-9.</p>
<p>MATLAB Command Window colors</p>	<p>Set colors to help you quickly identify errors, warnings, and hyperlinks in the Command Window.</p>

Colors Programming Tools Preferences

You can specify options used for editing and debugging code, including code analysis colors, variable and function colors, and cell display options.

Select **File > Preferences > Colors > Programming**, and then set options as described in the table below.

Preference	Usage
Code analyzer colors	<ul style="list-style-type: none"> • Warnings—Specifies the color Code Analyzer uses to identify code in the Editor for which there are warning messages. • Autofix highlight—Specifies the color Code Analyzer uses to identify code in the Editor for which there is an automatic fix. <p>For details, see, “Automatically Check Code in the Editor — Code Analyzer” on page 8-91.</p>
Variable and function colors	<ul style="list-style-type: none"> • Automatically highlight—Specifies the color the Editor uses to highlight all occurrences of a specific variable or function. For details, see “Find and Replace Functions or Variables in the Current File” on page 8-41. • Variables with shared scope—Specifies the color of variables with shared scope. The text is colored, not shaded. For details, see “Avoid Variable and Function Scoping Problems” on page 8-110
Cell display options	<p>Highlight cells—Specifies the color the Editor uses to shade code cells.</p> <p>Show lines between cells—Specifies that code cells divisions appear with a gray line between each cell in the Editor. These lines do not appear in the published or printed file.</p> <p>See also, “Evaluate Subsections of Files Using Code Cells” on page 8-58.</p>

Toolbars Preferences

You can customize some toolbars in the MATLAB application.

Select **File > Preferences > Toolbars**, and then set options as described in the table below.

For step-by-step instructions on setting these preferences, see “Toolbar Customization” on page 2-11.

Preference	Usage
Toolbar	Select the toolbar you want to customize.
Layout	Rearrange the order of controls in the toolbar by dragging and dropping them to a new location in the Layout .
Controls	Select which buttons appear on the selected toolbar.

Web Preferences

Web preferences enable you to specify Internet connection information to MATLAB.

Note

Limitations

- MATLAB supports nonauthenticated, basic, digest, and NTLM proxy authentication types.
- You cannot specify proxy server settings using a script.
- There is no automated way to provide MATLAB with the proxy server settings that your system browser uses.

To set Web preferences, select **File > Preferences > Web**, and then adjust preference options as described in the table below.

Preference	Usage
Use a proxy server to connect to the Internet	Provide information that MATLAB needs to access the internet when your network uses a firewall or another method of protection that restricts Internet access.
Proxy host	Specify a value for the Proxy host . For example, 172.16.10.8 or ourproxy. If you do not know the values for your proxy server, ask your system or network administrator for the information.

Preference	Usage
Proxy port	Specify an integer value for the Proxy port . For example, 22. If you do not know the values for your proxy server, ask your system or network administrator for the information.
Use a proxy with authentication	Specifies that your proxy server requires a user name and password.
Proxy username	Specify the proxy server user name.
Proxy password	Specify the proxy server password. Note MATLAB stores the password without encryption in your <code>matlab.prf</code> file.
Test connection	Ensure that your settings work. 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.
System Web browser ⁶ UNIX ⁶ platforms only — excluding Macintosh	<ul style="list-style-type: none"> • Command—Specifies the system command to open the browser. For example, <code>opera</code>, opens the Opera Web browser. • Options—Specifies options for the system browser. For example, <code>geometry 1064x860</code> specifies the size of the window for Opera.

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

Running Functions — Command Window and History

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

- “Command Window” on page 3-3
- “Command History Window” on page 3-4
- “Open Command Window, Create Variables, and Run Code” on page 3-6
- “Create MATLAB Shortcuts to Rerun MATLAB Commands” on page 3-23
- “Open Command History Window” on page 3-29
- “Create Files from, Run, and Copy Command History Commands” on page 3-30
- “Adjust Command Window Appearance” on page 3-32
- “Adjust Command History Window Appearance” on page 3-34
- “Command Window Code and Output Display” on page 3-36
- “Avoid Mistakes When Entering Code” on page 3-41
- “Find Text in the Command Window” on page 3-53
- “Find Text in the Command History Window” on page 3-61
- “Command Window Preferences” on page 3-63
- “Keyboard Preferences” on page 3-65

- “Command History Preferences” on page 3-67

Command Window

Use the Command Window to enter data, run MATLAB code, and display results. The Command Window includes a **New to MATLAB?** getting started message bar that provides quick access to videos, demos, and getting started information.

If you have an active Internet connection, you can watch the video demo, *Working in the Development Environment* for an overview of the major features.

The Command Window prompt indicates that MATLAB is ready to accept input. Enter statements at the prompt, which is also known as the command line. The appearance of the prompt varies:

- `>>` — Indicates that the Command Window is in normal mode.
- `EDU>>` — Indicates that the Command Window is in normal mode, in MATLAB Student Version.
- `K>>` — Indicates that MATLAB is in debug mode.

Type `dbquit` to return to normal mode. For information on debugging, see “Debugging Process and Features” on page 8-116.

Command History Window

The Command History window displays a log of statements you ran in the current and previous MATLAB sessions. The time and date for each session appear at the top of the statements listed for that session. All entries remain until you delete them, or until the command history file exceeds its maximum size of 200,000 bytes. When the file exceeds its maximum size, MATLAB automatically deletes the oldest entries. If you have an active Internet connection, you can watch the video demo, *Working in the Development Environment* for an overview of the major functionality.

Command History File — `history.m`

When you recall lines in the Command Window, MATLAB uses the command history file, `history.m`.

The history file:

- Resides in the folder returned when you type `prefdir` in the Command Window
- Loads when MATLAB starts
- Stores a maximum of 200,000 bytes
- Deletes the oldest entries, as needed, to maintain the maximum number of bytes

MATLAB saves statements that run in the Command Window to the to the history file. 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 file does not include every action taken in MATLAB. For example, suppose you run the following statement:

```
a = 1:10
```

If you then modify the value of `a` in the Variable Editor, there is no record of that modification in the history.

By default, MATLAB automatically saves the command history file after each command. This allows you to more easily recover your commands and state in the event of an abnormal termination. To change when MATLAB saves the history file and adjust which statements MATLAB saves, select **File > Preferences > Command History**.

Open Command Window, Create Variables, and Run Code

In this section...

“Open the Command Window” on page 3-6

“Create Variables and Run Functions” on page 3-6

“Find Functions Using the Function Browser” on page 3-7

“Enter Statements in the Command Window” on page 3-13

“Stop Execution” on page 3-14

“Run External Commands, Scripts, and Programs” on page 3-15

“Keep a Session Log” on page 3-18

“Evaluate or Open Code You Select” on page 3-18

“Display Hyperlinks in the Command Window” on page 3-18

Open the Command Window

To open the Command Window:

- From the desktop, select **Desktop > Command Window**
If you do not want the other desktop tools visible, select **Desktop > Desktop Layout > Command Window Only**.
- Programmatically, include the `commandwindow` function in a MATLAB function or script.

Create Variables and Run Functions

This example shows how to use the Command Window to create variables and run functions. For detailed information on using MATLAB functions and creating your own, see “Scripts and Functions”.

Create A, a 3-by-3 matrix:

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

MATLAB returns:

```
A =
```

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

Run a function that MathWorks provides. Type the function name including all arguments. For example, typing:

```
magic(2)
```

returns:

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

- If an error message appears, click the underlined portion of the error message. The offending file opens in the Editor, scrolled to the line containing the error.
- If you type additional statements while MATLAB is busy running another statement, the additional statements are buffered in a queue. The next statement runs when the current statement finishes.
- To search for functions provided by MathWorks, use the function browser. For details, see “Find Functions Using the Function Browser” on page 3-7.

Find Functions Using the Function Browser

- “What Is the Function Browser?” on page 3-7
- “Steps for Using the Function Browser” on page 3-8
- “Interpreting Search Results in the Function Browser” on page 3-12
- “Customizing the Function Browser” on page 3-12

What Is the Function Browser?

The Function Browser provides quick access to the syntax and description for a function by using the reference pages for installed MathWorks products.


Using the Function Browser, you can browse or search for a MATLAB function.

Be aware that:

- 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 6-28.

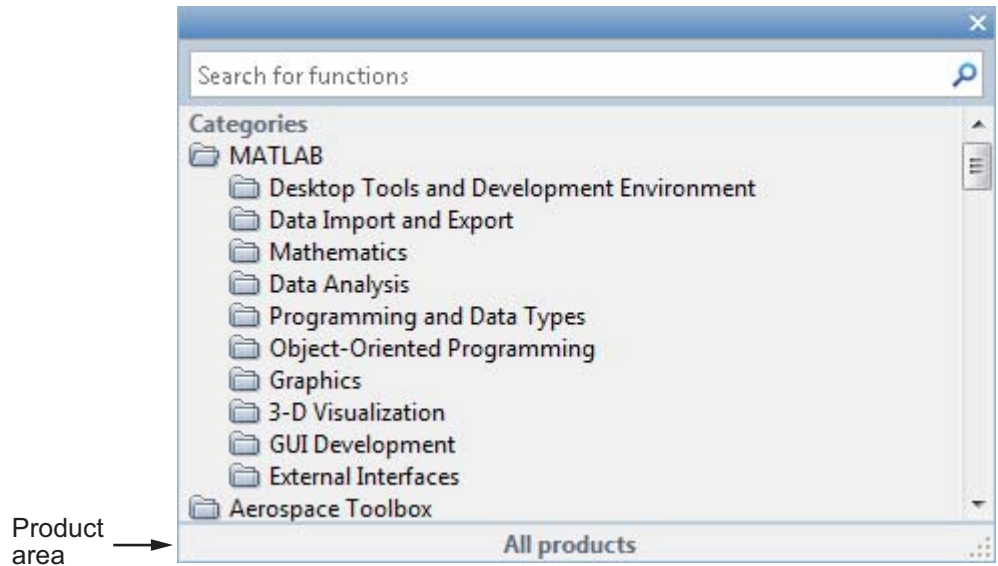
Steps for Using the Function Browser

1 Open the Function Browser by doing either of the following:

- Select **Help > Function Browser**.
- Click the **Browse for functions** button,  in the Command Window or Editor.

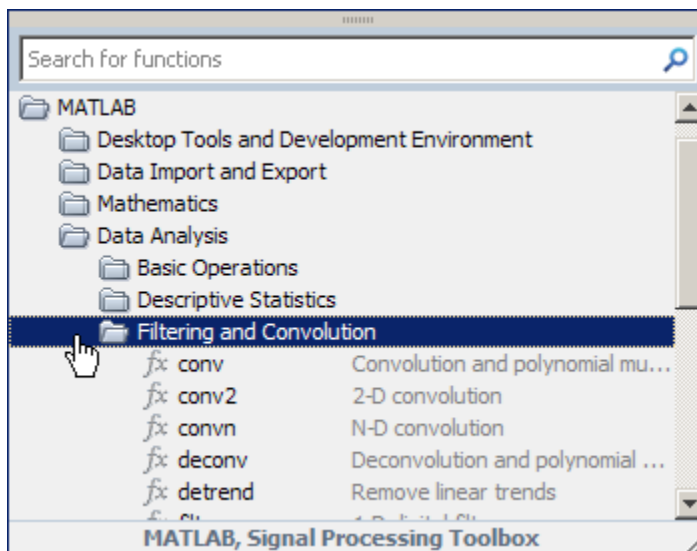
The Function Browser closes when you move the pointer outside of it. To keep the Function Browser open, drag it by the top edge to a different location.

2 Restrict the products the Function Browser looks in, click the product area at the bottom of the Function Browser, and then set the **Filter by Product** options in the Help Preferences dialog box that opens.

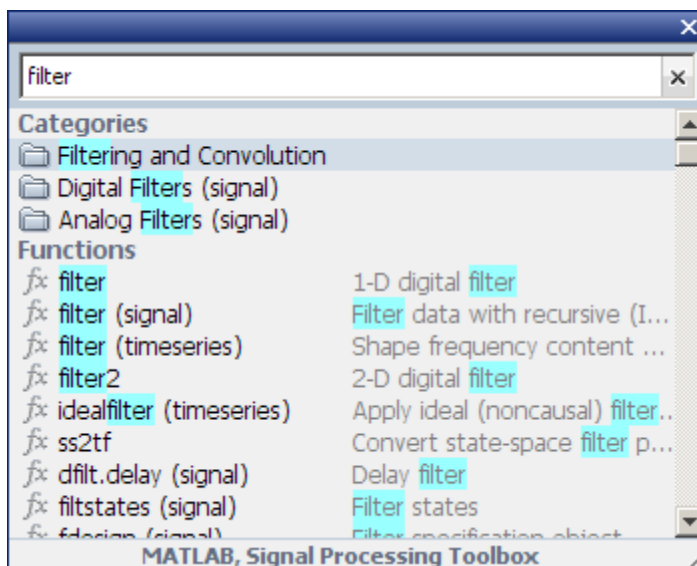


- 3 Find functions by browsing in the list of categories or by typing a search term.

The following illustration shows how you browse for functions by expanding categories of interest.

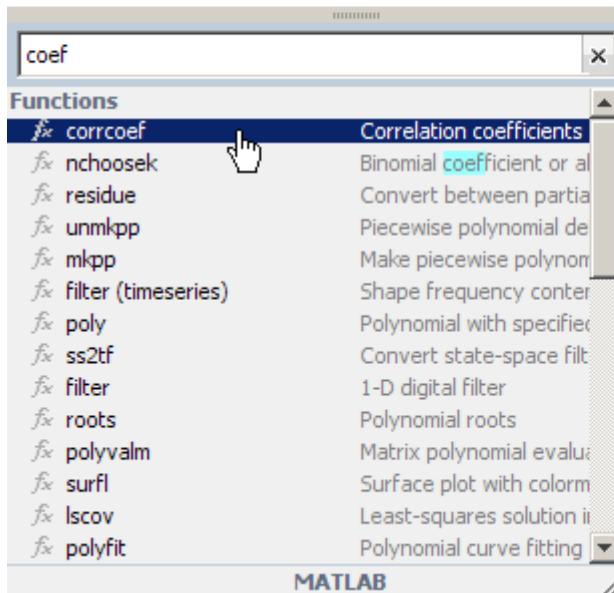


The following illustration shows results when you search, for example, for filter.



- 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.



corrcoef [More Help...](#)

Correlation coefficients

`R = corrcoef(X)` returns a matrix `R` of correlation coefficients calculated from an input matrix `X` whose rows are observations and whose columns are variables. The matrix `R = corrcoef(X)` is related to the covariance matrix `C = cov(X)` by

$$R(i, j) = \frac{C(i, j)}{\sqrt{C(i, i)C(j, j)}}$$

`corrcoef(X)` is the zeroth lag of the normalized covariance function, that is, the zeroth lag of `xcov(x, 'coeff')` packed into a square array.

- 5 Choose a function you want to use and then:

- Add a function name after the cursor in the Command Window or Editor by double-clicking the name in the Function Browser results list.

- Drag the function name from the Function Browser into any tool or application.
- Right-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 Image 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.

Customizing the Function Browser

- Show or hide the Browse for functions button  in the Command Window by selecting **File > Preferences > Command Window**, and then selecting or clearing **Show Function Browser button**.

- Show or hide the Browse for functions button  in the Editor by customizing the Editor toolbar. See “Toolbar Customization” on page 2-11 for details.
- Change the font that the Function Browser uses by selecting **File > Preferences > Fonts**. By default, the Function Browser uses the desktop text font and the pop-up window uses the HTML proportional text font.

Enter Statements in the Command Window

You can use a variety of methods to enter statements in the Command Window. If keyboard shortcuts do not behave as documented, see “Keyboard Shortcuts” on page 2-36.

To:	Do This:
Enter multiple statements on multiple lines before running any of the statements	Use Shift+Enter , the default keyboard shortcut for Break Line Without Code Execution between statements. This action is unnecessary when you enter a paired keyword statement on multiple lines, such as <code>for</code> and <code>end</code> .
Enter multiple functions on one line	Separate function calls with a comma (,) or semicolon (;). The semicolon suppresses the output for the command preceding the semicolon. For example: <code>format short; x = (1:10)';</code> MATLAB runs the functions in order, from left-to-right.

To:	Do This:
Enter multiple-line (long) statements	Use the line continuation character, <code>...</code> , also called dots, stops, or ellipsis between statement fragments. For strings in single quotation marks, you must complete the string in the line on which it was started. For example: <pre>headers = ['Last Name, First Name, ' ... 'Middle Initial']</pre>
Recall previous lines in the Command Window	To recall previous lines, press the up-arrow key \uparrow . To recall only lines that begin with certain characters, type the characters, and then press the up-arrow key.

Stop Execution

To stop file execution:

- **All platforms**—press **Ctrl+C** or **Ctrl+Break** .

Ctrl+C does not always effectively stop execution for files that run a long time, or that call built-ins or MEX-files that run a long time. If you experience this problem, include a `drawnow`, `pause`, or `getframe` function in your file, for example, within a large loop.

Also, **Ctrl+C** might be less responsive if you start MATLAB with the `-nodesktop` option.

- **Apple Macintosh platforms** —You can also use **Command+.** (the Command key and the period key).

Note For certain operations, stopping the program might generate errors in the Command Window.

Run External Commands, Scripts, and 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, the following code invokes the vi editor for a file named `yearlystats.m`:

```
!vi 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.

Restrictions maintained within the operating system determine the maximum length of the argument list you can provide as input to the bang (!) command. 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.

Running UNIX Programs That Are Off the System Path

You can run a UNIX program from MATLAB when the folder containing that file is not on the UNIX system path that is visible to MATLAB. To determine the system path that is visible to MATLAB, type the following in the Command Window:

```
getenv('PATH')
```

You can make modifications to the system path that persist for the current MATLAB session or across subsequent MATLAB sessions, as described in the sections that follow.

Modify the System Path for the Current MATLAB Session. Do one of the following:

- Change the current folder in MATLAB to the folder that contains the program you want to run.
- Issue these commands using the Command Window:

```
path1 = getenv('PATH')
path1 = [path1 ':/usr/local/bin']
setenv('PATH', path1)
!echo $PATH
```

If you restart MATLAB, the folder is no longer on the system path visible to MATLAB.

Modify the System Path Across MATLAB Sessions Within the Current Shell Session. To add a folder to the system path from the shell:

- 1 Stop MATLAB.
- 2 Depending on the shell you are using, type one of the following at the system command prompt, where *myfolder* is the folder that contains the program you want to run:

- Type this if you are using `bash` or a related shell:

```
export PATH="$PATH:myfolder"
```

- Type this if you are using `tcsh` or a related shell:

```
setenv PATH "${PATH}:myfolder"
```

- 3 Start MATLAB.
- 4 In the MATLAB Command Window, type:

```
!echo $PATH
```

If you restart MATLAB within the current shell session, the folder remains on the system path visible to MATLAB. However, if you restart the shell session, and then restart MATLAB, the folder is no longer on the system path visible to MATLAB.

Modify the System Path Across All MATLAB Sessions. To make adjustments that persist across shell and MATLAB sessions, add the following commands to the MATLAB startup file as described in “Specifying Startup Options in the MATLAB Startup File” on page 1-19:

```
path1 = getenv('PATH')
path1 = [path1 ':/usr/local/bin']
setenv('PATH', path1)
!echo $PATH
```

Keep a Session Log

To keep a log of previously issued commands, do one of the following:

- Use the `diary` function.
- Use the `logfile` startup option—see “Startup Options” on page 1-18.

Evaluate or Open Code You Select

You can evaluate or open code you select in the Command Window, as described in this list:

- To append text to code currently at the prompt, and then execute that code — select the code, and then press **Enter**
- To evaluate a statement — select it from any MATLAB desktop tool, right-click, and then select **Evaluate Selection**.
- To open a function, file, variable, or Simulink model from the Command Window — select the name in the Command Window, right-click, and then select **Open Selection**.

MATLAB runs the `open` function on your behalf to open the selected item in the appropriate tool. Specifically:

- 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`.

Display 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 one or more MATLAB functions by prefixing them with `matlab:` (called *matlabcolon* syntax)

Creating 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 following command:

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

displays the following hyperlink in the Command Window:

The MathWorks Web Site

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

Transferring Files Using FTP

To create a link to an FTP site, enter the site address as input to the `disp` command as follows:

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

This command displays the following as a link in the Command Window:

The MathWorks FTP Site

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

Running MATLAB Functions from Hyperlinks

The special keyword `matlab:` lets you embed commands in other functions. Most commonly, the functions that contain it display hyperlinks, which execute the commands when you click the hyperlink text. Functions that support `matlab:` syntax include `disp`, `error`, `fprintf`, `help`, and `warning`.

Use `matlab:` syntax to create a hyperlink in the Command Window that runs one or more functions. For example, you can use `disp` to display the word Hypotenuse as an executable hyperlink as follows:

```
disp(' <a href="matlab:a=3; b=4;c=hypot(a,b)">Hypotenuse</a> ')
```

Clicking the hyperlink executes the three commands following `matlab:`, resulting in

```
c =  
    5
```

Executing the link creates or redefines the variables `a`, `b`, and `c` in the base workspace.

The argument to `disp` is an `<a href>` HTML hyperlink. Include the full hypertext string, from `' <a href= to '` within a single line, that is, do not continue a long string on a new line. No spaces are allowed after the opening `<` and before the closing `>`. A single space is required between `a` and `href`.

You cannot directly execute `matlab:` syntax. That is, if you type

```
matlab:a=3; b=4;c=hypot(a,b)
```

you receive an error, because MATLAB interprets the colon as an array operator in an illegal context:

```
??? matlab:a=3; b=4;c=hypot(a,b)
```

```
      |  
Error: The expression to the left of the equals sign  
      is not a valid target for an assignment.
```

You do not need to use `matlab:` to display a live hyperlink to the Web. For example, if you want to link to an external Web page, you can use `disp`, as follows:

```
disp(' <a href="http://en.wikipedia.org/wiki/Hypotenuse">Hypotenuse</a> ')
```

The result in the Command Window looks the same as the previous example, but instead opens a page at `en.wikipedia.org`:

Hypotenuse

Using `matlab:`, you can:

- “Run a Single Function” on page 3-21

- “Run Multiple Functions” on page 3-21
- “Provide Command Options” on page 3-21
- “Include Special Characters” on page 3-22

Run a Single Function. Use `matlab:` to run a specified statement when you click a hyperlink in the Command Window. For example, run this command:

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

It displays this link in the Command Window:

[Generate magic square](#)

When you click the link, MATLAB runs `magic(4)`.

Run Multiple Functions. You can run multiple functions with a single link. For example, run this command:

```
disp(' a href="matlab: x=0:1:8;y=sin(x);plot(x,y)">Plot x,y</a>')
```

It displays this link in the Command Window:

[Plot x,y](#)

When you click the link, MATLAB runs this code:

```
x = 0:1:8;  
y = sin(x);  
plot(x,y)
```

Redefine `x` in the base workspace:

```
x = -2*pi:pi/16:2*pi;
```

Click the hyperlink, `Plot x,y` again and it changes the current value of `x` back to `0:1:8`. The code that `matlab:` runs when you click the `Plot x,y` defines `x` in the base workspace.

Provide Command Options. Use multiple `matlab:` statements in a file to present options, such as

```
disp('<a href = "matlab:state = 0">Disable feature</a>')  
disp('<a href = "matlab:state = 1">Enable feature</a>')
```

The Command Window displays the links that follow. Depending on which link you click, MATLAB sets `state` to 0 or 1.

[Disable feature](#)
[Enable feature](#)

Include Special Characters. MATLAB correctly interprets most strings that include special characters, such as a greater than symbol (>). For example, the following statement includes a greater than symbol (>).

```
disp('<a href="matlab:str = ''Value > 0''">Positive</a>')
```

and generates the following hyperlink.

[Positive](#)

Some symbols might not be interpreted correctly and you might need to use the ASCII value for the symbol. For example, an alternative way to run the previous statement is to use ASCII 62 instead of the greater than symbol:

```
disp('<a href="matlab:str=[''Value ' char(62) '' 0'']'>Positive</a>')
```

Create MATLAB Shortcuts to Rerun MATLAB Commands

In this section...

“What Is a MATLAB Shortcut?” on page 3-23

“When to Use MATLAB Shortcuts” on page 3-23

“MATLAB Shortcut Creation” on page 3-24

“Run MATLAB Shortcuts” on page 3-26

“Edit and Organize MATLAB Shortcuts” on page 3-26

“MATLAB Toolbar Shortcuts — Hide, Show, and Delete” on page 3-27

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 a MATLAB script, MATLAB does not save it as a script.

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.

MATLAB Shortcut Creation

You can create a MATLAB shortcut to run from the desktop **Start** button, or from the shortcuts toolbar, as described in the sections that follow.

Create MATLAB Start Button Shortcuts

To create a start button shortcut, follow these steps:

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

You can also drag statements from a desktop tool, such as the Command History, onto the **Start** button.

- 2 Complete the Shortcut Editor dialog box.

- a Provide a shortcut name in the **Label** field.

For this example, `sea_temp_environment`.

- b Enter the statements you want to run in the **Callback** field.


For this example, enter these statements:

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

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.

Create MATLAB Toolbar Shortcuts

Follow these steps:

- 1** Right-click the shortcuts toolbar, and then select **New Shortcut**.

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

- 2** Complete the Shortcut Editor dialog box:
 - a** In the **Label** field, enter a name for the shortcut.
 - b** In the **Callback** field, type statements you want the shortcut to run.

You can also drag and drop statements from the Command Window, Command History Window, or a file.

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

- c** Do not change the **Category** field value, **Toolbar Shortcuts**.
- d** In the **Icon** field, select an icon.
- 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.

Run 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.



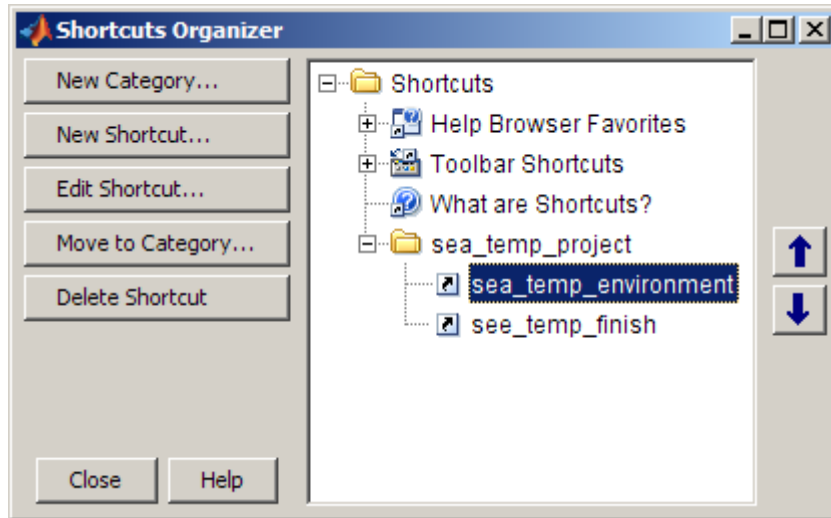
All the statements in the shortcut **Callback** field execute. It is as if you ran those statements from the Command Window, although they do not appear in the Command History window.

Edit and Organize 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 then select **Shortcuts > Organize Shortcuts**.
 - From the right-click the shortcuts toolbar, and then select **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.
- Select one or more items in the navigation tree, right-click, and then select an action from the context menu.

Changes take effect immediately.

3 Click **Close**.

MATLAB Toolbar Shortcuts – Hide, Show, and Delete

For step-by-step instructions on how to add a shortcut to the toolbar, see “Create MATLAB Toolbar Shortcuts” on page 3-25.

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.

Open Command History Window

To open the Command History window:

- From the desktop, select **Desktop > Command History**
- Programmatically, use the `commandwindow` function.

Create Files from, Run, and Copy Command History Commands

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
Create a file from a statement or statements.	Select an entry or entries, and then right-click and select Create Script from the context menu. The Editor opens a new file that contains the statements you selected from the Command History window.
Run Command History commands in the Command Window	Do one of the following: <ul style="list-style-type: none"> • Double-click an entry or entries in the Command History window. For example, double-click <code>edit myfile</code> to open <code>myfile.m</code> in the Editor. • Right-click an entry and select Evaluate Selection from the context menu. • Select an entry and press Enter or Return.
Copy statements to another window.	Do either of the following: <ul style="list-style-type: none"> • Select an entry or entries, and then select Copy from the context menu. Paste the selection into an open file in the Editor or any application. • Drag the selection from the Command History window to an open file or another application.
Create a shortcut from a statement or statements.	Do either of the following: <ul style="list-style-type: none"> • Select an entry or entries, and then right-click and select Create Shortcut from the context menu. • Drag the selection to the Shortcuts toolbar. The Shortcut Editor opens and the selected statements appear in the Callback field.

Action	How to Perform the Action
	For more information, see “Create MATLAB Shortcuts to Rerun MATLAB Commands” on page 3-23.

Adjust Command Window Appearance

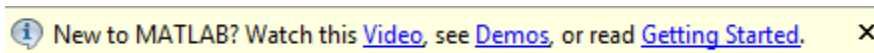
In this section...
“Show or Hide Links to Videos, Demos, and Getting Started” on page 3-32
“Show or Hide Function Browser Button” on page 3-32
“Set the Number of Lines in the Scroll Buffer” on page 3-33

Note See also:

- “Fonts” on page 2-2
 - “Color Settings” on page 2-8
 - “Open and Rearrange Desktop Tools and Documents” on page 2-13
-

Show or Hide Links to Videos, Demos, and Getting Started


By default, MATLAB displays a message bar at the top of the Command Window:



To change the message bar display:

- 1** Select **File > Preferences > Command Window**.
- 2** Select or clear **Show getting started message bar**.
- 3** Click **OK**.

Show or Hide Function Browser Button

By default, MATLAB displays a the Function Browser button, , to the left of the prompt in the Command Window.

To change the Function Browser button display:

- 1** Select **File > Preferences > Command Window**.
- 2** Select or clear **Show Function Browser button**.
- 3** Click **OK**.

Set the Number of Lines in the Scroll Buffer

The size of the scroll buffer controls the number of lines you see when you scroll vertically in the Command Window. A larger scroll buffer provides a larger base for search features, but requires more memory. By default, the scroll buffer is set to 5,000 lines.

Note The scroll buffer size does not impact the number of lines you can recall in the Command Window. 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.

To change the size of the scroll buffer:

- 1** Select **File > Preferences > Command Window**.
- 2** In the **Number of lines in command window scroll buffer** field, enter a new value.
- 3** Click **OK**.

Adjust Command History Window Appearance

In this section...
“Delete Entries from the Command History Window” on page 3-34
“Change the Date Format in the Command History Window” on page 3-35

Note See also:

- “Fonts” on page 2-2
 - “Color Settings” on page 2-8
 - “Open and Rearrange Desktop Tools and Documents” on page 2-13
-

Delete Entries from the Command History Window

Consider deleting entries from the Command History window when it becomes difficult to find the lines you want.

Note You cannot recall entries you delete from the Command History window.

To delete specific entries in the Command History window:

- 1** Select the entries to delete.

Tip To select all entries for a session, select the timestamp for that session.

- 2** Right-click and select **Delete Selection** from the context menu, or press the **Delete** key.

To delete all entries, right-click in the Command History window, and then select **Clear Command History** from the context menu.

Change the Date Format in the Command History Window

MATLAB uses your operating system's short date format to display dates in the Command History window. To change the date format, for instance from MM/DD/YYYY to DD/MM/YYYY:

- 1 Change the short date format for your operating system as described in its documentation.
- 2 Right-click in the Command History window and select **Clear Command History**.

Note Clearing the command history deletes all entries from the Command History window. This means you can no longer recall those entries in the Command Window.

Command Window Code and Output Display

In this section...

“Keep or Suppress Blank Lines in Output” on page 3-36

“Format Numerals” on page 3-37

“Wrap Lines of Code to Fit Window Width” on page 3-38

“Matrix Display Width” on page 3-38

“Number of Spaces Assigned to Tab Key” on page 3-39

“Suppress Code Output” on page 3-39

“Page Code Output” on page 3-39

“Clear the Command Window” on page 3-40

Note See also:

- “Fonts” on page 2-2
- “Color Settings” on page 2-8

Keep or Suppress Blank Lines in Output

By default, MATLAB displays blank lines in command output.

To control the display of blank lines in output:

1 Select **File > Preferences > Command Window**.

2 Select a **Numeric format**:

- `1oose`—Keeps the display of blank lines

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

```
x =
```

```
1.3333    0.0000
```

- **compact**—Suppresses the display of blank lines

```
>> x = [4/3 1.2345e-6]
x =
    1.3333    0.0000
```

3 Click **OK**.

Alternatively, use the `format` function with the `loose` or `compact` option.

Format Numerals

You can change the way numbers display. By default, MATLAB uses the short format (5-digit scaled, fixed-point values).

For example suppose you enter `x = [4/3 1.2345e-6]` in the Command Window:

- If you use the default format, MATLAB returns `x = 1.3333 0.0000`.
- If you set the format to `short e`, then MATLAB returns `1.3333e+00
1.2345e-06`.
- If you set the format to `+`, then MATLAB returns `++`.

Note The text display format affects only how numbers are shown, not how MATLAB computes or saves them.

To change the numeric display option, do one of the following:

- Select **File > Preferences > Command Window**, and then choose a **Numeric format** option.
- Use the `format` function.

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` function.

Wrap Lines of Code to Fit Window Width

A line of code or its output can exceed the width of the Command Window, requiring you to use the horizontal scroll bar to view the entire line. In console mode, it can be useful to make a single line of input or output in the Command Window break into multiple lines to fit within the current width of the Command Window.

To wrap lines of code:

- 1 Select **File > Preferences > Command Window**.
- 2 Select **Wrap Lines**.
- 3 Click **OK**.

Matrix Display Width

By default, a row of matrix output fills the width of the Command Window, and then continues displaying output in a new row.

To determine the number of characters and lines that will display in the Command Window, given its current size, use:

```
get(0, 'CommandWindowSize')
```

For example, a result of 50, 25 means 50 characters will display across the Command Window, and 25 lines will display.

To set the matrix output to 80 characters per row (and then continue displaying output in a new row, regardless of the width of the Command Window) :

- 1 Select **File > Preferences > Command Window**.
- 2 Select **Set matrix display width to eighty columns**.

Note If you also select **Wrap lines**, 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.

3 Click **OK**.

Run `get(0, 'CommandWindowSize')` again. The result for the same size Command Window is 80, 25.

Number of Spaces Assigned to Tab Key

By default, a tab is set to four spaces, except on UNIX⁷ platforms where the default is eight spaces.

To change the setting:

1 Select **File > Preferences > Command Window**.

2 Type a new value in the **Tab size** field.

3 Click **OK**.

Note This setting does not apply if you select **File > Preferences > Keyboard** and enable tab completion in the Command Window.

Suppress Code Output

To suppress code output, add a semicolon (;) to the end of a command. This is particularly useful when code generates large matrices. For example, running the following code creates A, but does not show the resulting matrix in the Command Window:

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

Page Code Output

When output in the Command Window exceeds the visible portion of the window, follow these steps to view the output, one screen full at a time:

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

- 1 In the Command Window, type `more` on.
- 2 View the remaining output:
 - Advance to the next line by pressing **Enter**.
 - Advance to the next page by pressing **Space Bar**.
 - Stop displaying the output by pressing **q**.

Clear the Command Window

If the Command Window seems cluttered, you can clear all the text (without clearing the workspace) by doing one of the following:

- Select **Edit > Clear Command Window**
- Use the `clc` function.
- Use the `home` function.

Avoid Mistakes When Entering Code

In this section...

“Highlight Syntax to Help Ensure Correct Entries in the Command Window” on page 3-41

“Avoid Mismatched Parentheses, Brackets, Braces, and Paired Keywords” on page 3-42

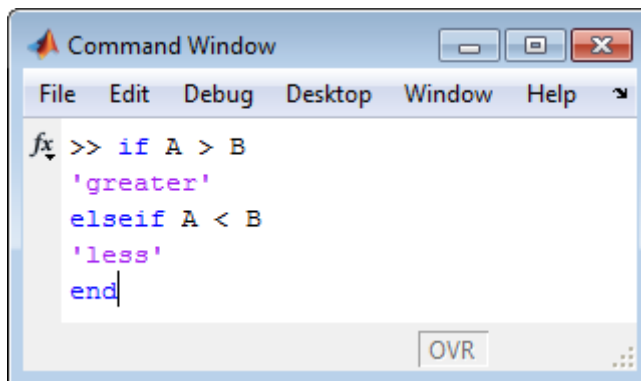
“Complete Names in the Command Window Using the Tab Key” on page 3-43

“View Function Syntax Hints While Entering a Statement” on page 3-49

Highlight Syntax to Help Ensure Correct Entries in the Command Window

To help you find MATLAB elements, such as matching if/else statements, some entries appear in different colors in the Command Window. This is known as syntax highlighting. By default:

- Keywords are blue.
- Strings are purple.
- Unterminated strings are maroon.



The screenshot shows the MATLAB Command Window with a menu bar (File, Edit, Debug, Desktop, Window, Help) and a toolbar. The command prompt is at the top left. The code entered is: `if A > B`, `'greater'`, `elseif A < B`, `'less'`, and `end`. The keywords `if`, `elseif`, and `end` are highlighted in blue. The strings `'greater'` and `'less'` are highlighted in purple. The cursor is at the end of the `end` line. A status bar at the bottom right shows 'OVR' and a small icon.

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

Except for errors, output in the Command Window does *not* appear with syntax highlighting.

To change syntax highlighting preferences, select **File > Preferences > Editor/Debugger > Languages**.

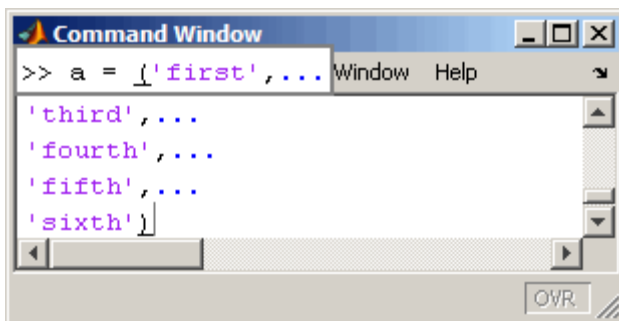
Avoid Mismatched Parentheses, Brackets, Braces, and Paired Keywords

When you type code at the Command prompt or in the Editor, MATLAB helps you avoid mismatched delimiters, such as parenthesis, brackets, and braces. In addition, MATLAB indicates paired language keywords in the Editor. Paired language keywords include `for`, `if`, `while`, `else`, and `end` statements.

By default, MATLAB indicates matched and mismatched delimiters and paired language keywords as follows:

- Type a closing delimiter—MATLAB briefly highlights the corresponding opening delimiter.
- Type more closing delimiters than opening delimiters—MATLAB beeps.
- Use the arrow keys to move the cursor over one delimiter—MATLAB briefly underlines both delimiters in a pair. If no corresponding delimiter exists, MATLAB puts a strike line through the unmatched delimiter.

If a matching delimiter exists, but it is not visible on the screen, a pop-up window appears and shows the line containing the matching delimiter. Click in the pop-up window to go to that line.



Change Delimiter Matching Indicators

To change delimiter matching indicators, and when and if they appear:

- 1 Select **File > Preferences > Keyboard**.
- 2 Under **Delimiter Matching**, choose when you want MATLAB to notify you of matches and mismatches in pairs of delimiters.
- 3 Choose how you want MATLAB to indicate matches by selecting one of the following from the **Show match with** drop-down menu associated with your selection or selections in step 2:
 - **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.
- 4 Choose how you want MATLAB to indicate mismatches by selecting one of the following from the **Show mismatch with** menu associated with your selection or selection in step 2:
 - **Beep** — MATLAB beeps.
 - **Strikethrough** — The delimiter you typed is crossed out briefly.
 - **None** — There is no action.

Complete Names in the Command Window Using the Tab Key

MATLAB helps you complete the names of functions, models, MATLAB objects, files, folders, variables, structures, and Handle Graphics property names, which can help you to avoid typographical errors.

To complete names in the Command Window:

- 1 Select **File > Preferences > Keyboard**.
- 2 Under **Tab completion**, select **Enable in Command Window**, and **Tab key narrows completions**.
- 3 Click **OK**.

- 4** In the Command Window, type the first few characters of the name you want to complete, and then press the **Tab** key.

For MATLAB to complete a file or folder name, it must be on the search path or in the current folder. Variables and structures must be in the current workspace.

- 5** Do one the following, depending on how MATLAB responds in step 3.

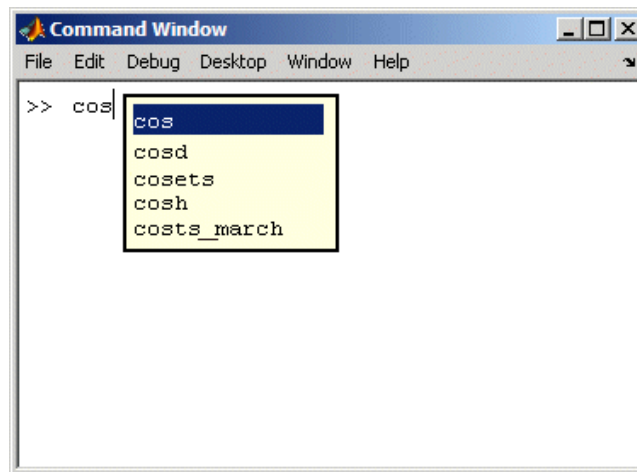
If MATLAB:	Then:
Completes the name you want	You are done.
Presents a list of possible matches	Use the arrow keys to select the name you want, and then press the Tab key. See “Example of Name Completion for a Variable in the Command Window” on page 3-45. In addition, you can: <ul style="list-style-type: none"> • Clear the list without selecting anything, by pressing the Esc (escape) key. • Narrow a long list before making a selection, by adding additional characters to your original term.
Completes the first part of a name that uses dot notation	At the command line, type a dot, and then press the Tab key. Repeat until the name is complete. See “Example of Name Completion for a Class Folder” on page 3-46.
Completes one property of several	At the command prompt, type a comma and the first part of the next property, and then press the Tab key. Repeat for all properties you want to specify. See “Example of Name Completion for Handle Graphics Properties” on page 3-48.

Example of Name Completion for a Variable in the Command Window

This example shows how to use tab completion for a variable name when there is more than one name that starts with the characters you type. The process is similar for other names that do not use dot notation.

- 1 Select **File > Preferences > Keyboard**.
- 2 Under **Tab completion**, select **Enable in Command Window**.
- 3 Click **OK**.
- 4 In the Command Window, create a variable, `costs_march`, by typing the following in the Command Window.

```
costs_march = 100;
```
- 5 Type `cos`, and then press the **Tab** key.



MATLAB lists all possible completions, including MATLAB functions and the variable name you created, `costs_march`.

- 6 Navigate the list of possible completions using the up and down arrow keys to find and highlight `costs_march`.

7 Choose `costs_march` by pressing the **Tab** key.

The Command Window displays `costs_march`.

8 At the command prompt, add arguments, and press **Enter**.

MATLAB runs the statement.

Example of Name Completion for a Class Folder

This example shows how to complete the name for a class folder. The steps for completing other names that use dot notation are similar.

Consider the `containers` package in the `matlabroot` folder. The `containers` package contains a `Map @-` folder and a `Map.m` file. The folder structure appears as follows:

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

To use tab completion:

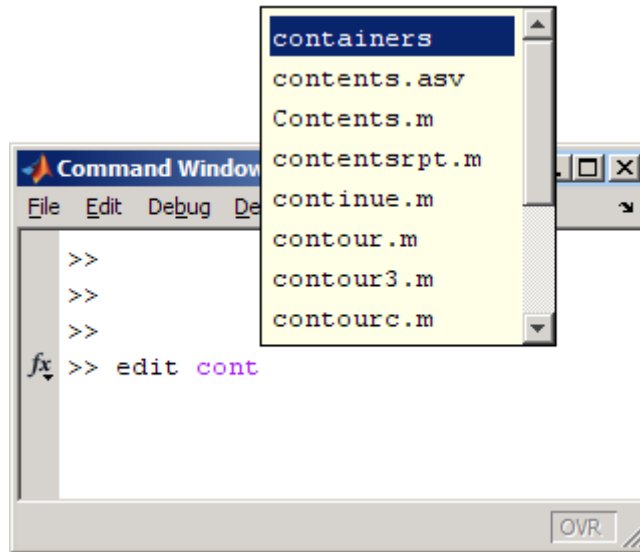
1 Select **File > Preferences > Keyboard**.

2 Under **Tab completion**, select **Enable in Command Window**.

3 Click **OK**.

4 In the Command Window, type `edit cont`, and then press **Tab**.

MATLAB displays a list from which to choose, such as the following.



- 5 Select **containers**, and then press **Tab**.

The Command Window displays `edit containers`.

- 6 At the command prompt, add a dot after **containers**, and then press **Tab**.

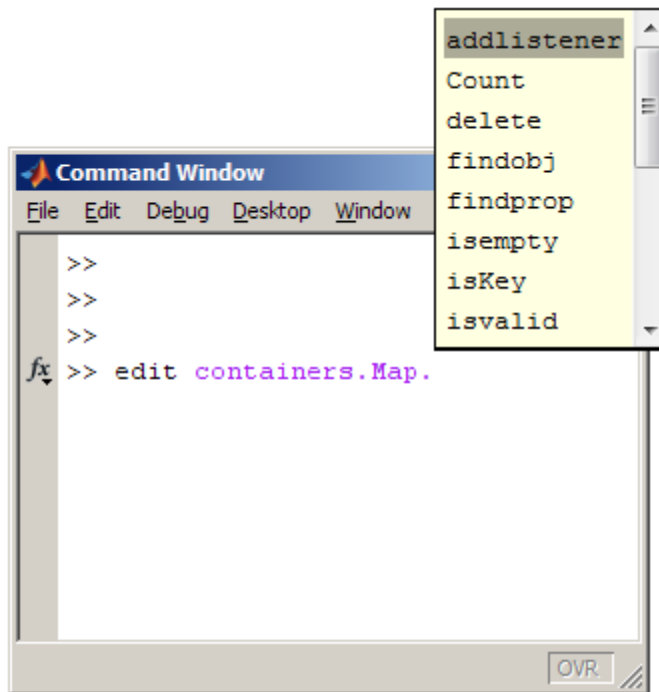
The Command Window displays:

```
edit containers.Map
```

```
.
```

- 7 At the command prompt, add a dot after **Map**, and then press **Tab**

MATLAB displays a new list.



- 8 Scroll down the list, select `isvalid`, and then press the **Tab** key.

The Command Window displays `edit containers.Map.isvalid`.

Example of Name Completion for Handle Graphics Properties

This example shows how to complete the names of MATLAB Handle Graphics properties using tab completion. The steps for completing properties for other objects that use dot notation are similar.

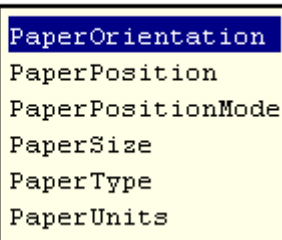
- 1 Select **File > Preferences > Keyboard**.
- 2 Under **Tab completion**, select **Enable in Command Window**.
- 3 Click **OK**.
- 4 In the Command Window, create a figure:

```
scrsz = get(0,'ScreenSize');
f=figure('Position',[1 scrsz(4)/2 scrsz(3)/2 scrsz(4)/2])
```

- 5** Type `set(f, 'pap`, and then press the **Tab** key.

MATLAB displays

```
set(f, 'paper|
```



- 6** Type `u`, and then press the **Tab** key.

MATLAB completes the property, including the closing quote:

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

- 7** Type `, 'c`, and then press the **Tab** key.

Because `centimeters` is the only possible completion, MATLAB automatically completes the property and displays:

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

- 8** Complete the command by typing a closing parenthesis at the command prompt.

View Function Syntax Hints While Entering a Statement

As you enter a function in the Command Window or Editor, syntax hints open in a pop-up window to display allowable input arguments for a function. These function hints appear for both MATLAB installed functions and functions you create. The syntax comes from the function definition statement

(first executable line) in the MATLAB program file. That file must be on the search path or in the current folder.

Be aware that:

- 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 black text when they should be blue.

To use function syntax hints in the Command Window, follow these steps, as shown for the `size` function:

- 1 Ensure function hints are enabled.

Select **File > Preferences > Keyboard**, and then setting the options for **Function hints**.

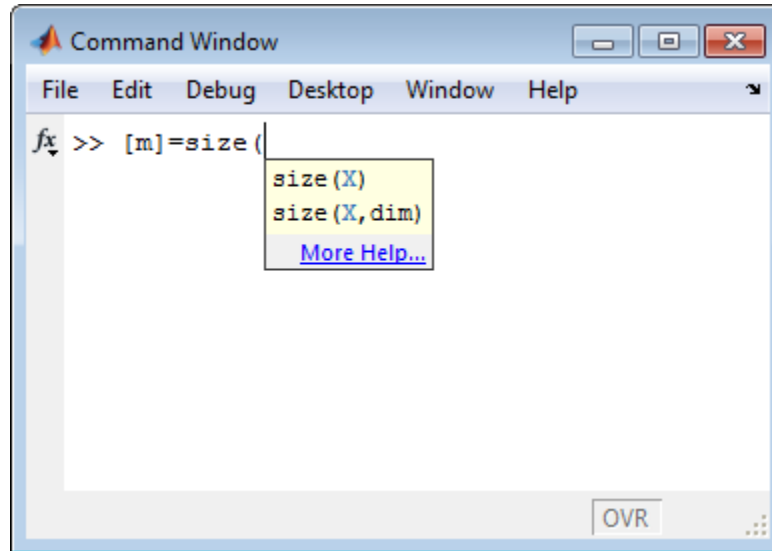
- 2 Type any output arguments and the equal sign. For example:

```
[m]=
```

- 3 Type the function name and the opening (left) parenthesis, and then pause. For example:

```
size(
```

A yellow window appears, displaying the syntax options for the function.



- 4 Type a variable name for the first input argument. You can type a variable for any argument that appears in blue.

Note Enter your variable names, and not the argument names shown in the window.

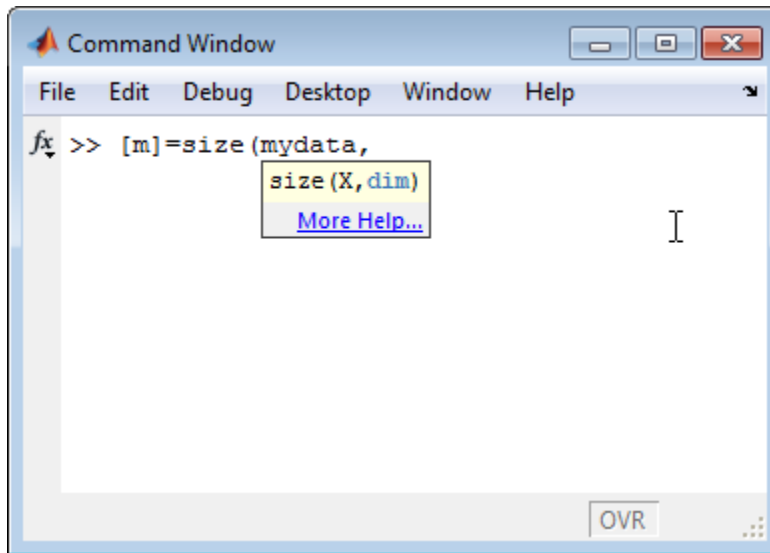
In this example, for the input argument `X`, type your variable:

```
my_data
```

- 5 Enter more arguments:
 - a Type a comma after the argument you just entered.

The displayed syntax options change, based on the argument you just entered.

For this example, only the `X,dim` option displays. `dim` is blue now, because you can enter it.



- b** Type the next input argument. It can be any argument that is blue.

For example, to run `size` for the third dimension, enter `3` for `dim`.

Continue entering more input arguments in the same way.

- 6** When you finish entering arguments, type the closing (right) parenthesis.

The pop-up window closes.

Find Text in the Command Window

In this section...

“Find Text Currently Displayed in the Command Window” on page 3-53

“Increase the Amount of Information Available for Searching in the Command Window” on page 3-53

“Use Incremental Search in the Command Window” on page 3-54

Find Text Currently Displayed in the Command Window

To search for specified text currently displayed in the Command Window:

- 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.

Increase the Amount of Information Available for Searching in the Command Window

You can increase the amount of information displayed in the Command Window so that more text is available for searching. Be aware that doing so requires more memory.

1 Select **File > Preferences > Command Window**, and then increase the setting for **Number of lines in the command window scroll buffer**.

2 Click **OK**.

Do *not* clear the Command Window.

In other words, do not enter `clc` or select **Edit > Clear Command Window**.

Use 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. The following sections provide details:

- “Example of Using Incremental Search” on page 3-54
- “Summary of Keyboard Shortcuts for Incremental Searches” on page 3-58
- “Case Sensitivity in Incremental Searches” on page 3-59

Example of Using Incremental Search

You control incremental search using keyboard shortcuts. The following example demonstrates how to use the incremental search feature in the Command Window when:

- MATLAB is running on a Windows or UNIX system and the **Active settings** field in the Keyboard Shortcuts Preference dialog box specifies the Emacs Default Set.
- MATLAB is running on a Macintosh system and **Active settings** field in the Keyboard Shortcuts Preference dialog box specifies the Macintosh Default Set.

For details, see “Choose a Set of Keyboard Shortcuts” on page 2-40.

1 Clear the Command Window to clear it of existing text and add this text:

```
qty = {15, 'Berlin'; 15, 'Boston'; 15, 'London'; ...  
15, 'Melbourne'; 21, 'Berlin'; 21, 'Boston'; 21, 'London'; ...  
21, 'Melbourne'; 22, 'Berlin'; 22, 'Berlin'; 22, 'Boston'; ...
```

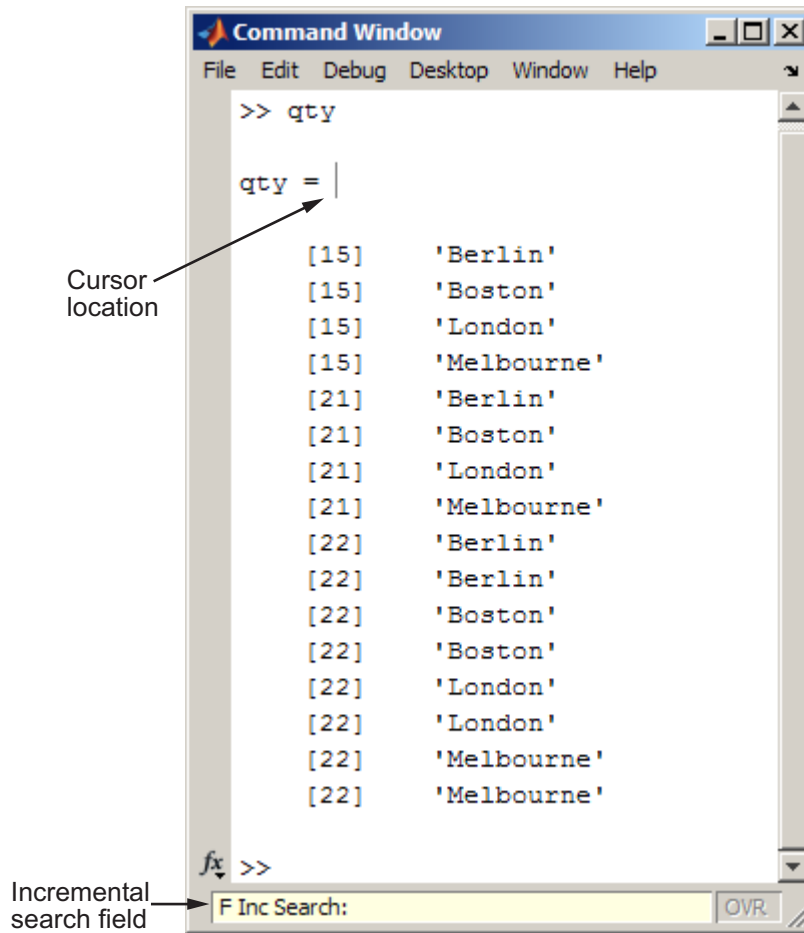
```
22, 'Boston'; 22, 'London'; 22, 'London'; 22, 'Melbourne'; ...  
22, 'Melbourne';}}
```

```
clc
```

```
qty
```

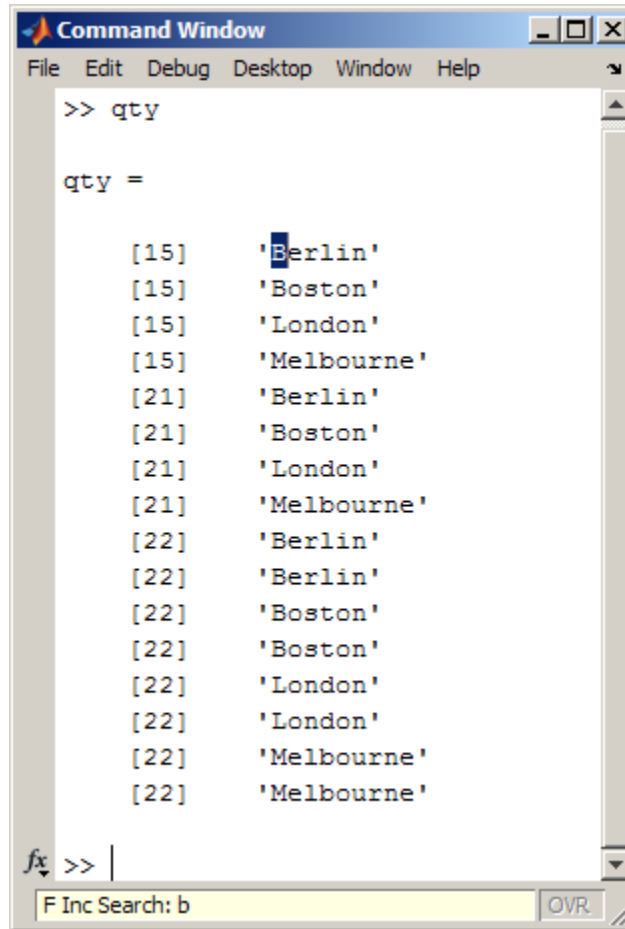
- 2** In the Command Window, position the cursor after the equal sign (=).
- 3** Begin an incremental search by pressing **Ctrl+Shift+S**.

An incremental search field appears at the bottom of the Command Window with the text `F incSearch`. The `F` indicates a forward search.



4 Begin a search for Boston by typing b.

The first occurrence of b highlights.



The screenshot shows a 'Command Window' with a menu bar (File, Edit, Debug, Desktop, Window, Help) and a search icon. The command prompt shows '>> qty' followed by 'qty ='. Below this is a list of city names with their corresponding counts in square brackets: [15] 'Berlin', [15] 'Boston', [15] 'London', [15] 'Melbourne', [21] 'Berlin', [21] 'Boston', [21] 'London', [21] 'Melbourne', [22] 'Berlin', [22] 'Berlin', [22] 'Boston', [22] 'Boston', [22] 'London', [22] 'London', [22] 'Melbourne', [22] 'Melbourne'. The letter 'b' in the first 'Berlin' entry is highlighted. At the bottom, the search field 'F Inc Search: b' is visible, and the 'OVR' button is on the right.

```
>> qty
qty =
[15] 'Berlin'
[15] 'Boston'
[15] 'London'
[15] 'Melbourne'
[21] 'Berlin'
[21] 'Boston'
[21] 'London'
[21] 'Melbourne'
[22] 'Berlin'
[22] 'Berlin'
[22] 'Boston'
[22] 'Boston'
[22] 'London'
[22] 'London'
[22] 'Melbourne'
[22] 'Melbourne'
```

If you mistype in the **Inc Search** field, use the **Backspace** key to remove the previous letter or letters and make corrections.

- 5 Type the next letter of the text you want to find. Type **o** to specify the next letter in Boston.

The **Inc Search** field contains **bo**, and the **bo** in Boston highlights.

- 6 To complete the highlighted word, press **Ctrl+W**.

Boston highlights and appears in the **Inc Search** field.

- 7** Find subsequent occurrences of the word by pressing **Ctrl+S** one or more times.

If MATLAB beeps, it means either that the text was not found, or the search wrapped past the bottom (or top) of the Command Window and is continuing at the top (or bottom).

- 8** Find previous occurrences of the word by pressing **Ctrl+R**.

The incremental search field begins with R to indicate a backward (reverse) search.

- 9** Search for a string that does not appear in the Command Window text by typing **x**.

MATLAB beeps and **Failing** appears in the incremental search field.

- 10** Automatically remove characters back to the last successful search by pressing **Ctrl+G**.

- 11** End incremental searching by pressing **Esc** (escape) or **Enter**.

The **Inc Search** field disappears. The cursor remains at the position where the text was last found, with the search text highlighted.

Summary of Keyboard Shortcuts for Incremental Searches

The following table summarizes the keyboard shortcuts you can use for performing incremental search actions. Except for the keyboard shortcuts that initiate an incremental search, you cannot customize these shortcuts. For information on choosing a keyboard shortcuts **Active settings** file, see “Choose a Set of Keyboard Shortcuts” on page 2-40.

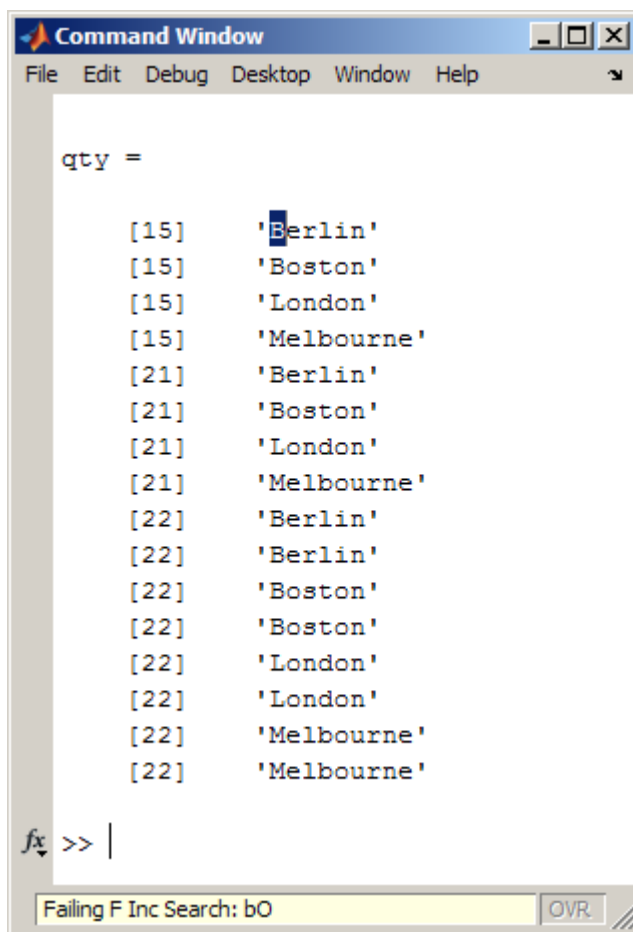
Action	Active Settings File	Keyboard Shortcut
Initiate a forward incremental search.	Windows Default Set	Ctrl+Shift+S
Initiate a forward incremental search.	Emacs Default Set or Macintosh Default Set	Ctrl+S

Action	Active Settings File	Keyboard Shortcut
Initiate a backward incremental search.	Windows Default Set	Ctrl+Shift+R
Initiate a backward incremental search.	Emacs Default Set or Macintosh Default Set	Ctrl+R
Complete a partially highlighted string of characters.	Any	Ctrl+W
Find the next occurrence of a string of characters.	Any	Ctrl+S
Remove characters from the Inc Search field, back to the last successful search	Any	Ctrl+G
End an incremental search.	Any	Esc (escape), or Enter , or any other key that is not a character or number

Case Sensitivity in Incremental Searches

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

For example, suppose you enter **b0** in the **Inc Search** field when the Command Window contains the text shown in the image that follows. Incremental search finds the **b** in **Berlin**, but does not find any additional matching text.



Find Text in the Command History Window

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

- “Quick Search for Entries Beginning with Specified Letters or Numbers” on page 3-61
- “Search for Entries Containing Partial Words, Whole Word, or Matching Case” on page 3-61

Quick Search for Entries Beginning with Specified Letters or Numbers

To quickly find entries in the Command Window based on the first few letters or numbers in the entry:

- 1 Type the first few letters or numbers of the entry you want to find in the Command History window.

The Command History window searches backwards and selects the previous entry that begins with the letters you typed. 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.

- 2 Search for additional instances by doing one of the following:
 - Find the previous or next occurrence of the entry with the up and down arrow keys, respectively.
 - Highlight each occurrence of the entry found, while you search for additional instances, press the **Ctrl** key with the up or down arrow key.
 - Highlight all instances of the entry, press **Ctrl+A**

Search for Entries Containing Partial Words, Whole Word, or Matching Case

To find text that matches the case, whole word, or partial word, follow these steps:

- 1 Select **Edit > Find**.

2 Complete the fields and select the options you want in the Find dialog box.

3 Click **Find Next** or **Find Previous**.

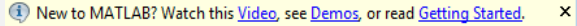
The search begins at the current cursor position. Find does not identify entries in collapsed nodes.


Tip To search for the specified text in other MATLAB desktop tools, change the selection in the **Look in** field.

Command Window Preferences

You can customize the visual display of the Command Window and command output within it.

Select **File > Preferences > Command Window**, and then adjust preference options as described in the table below.

Preference	Usage
Text display	Select a Numeric format option to specify the output format of numeric values in the Command Window. For details, see “Format Numerals” on page 3-37.
	Select a Numeric display option to specify whether blank lines appear in Command Window output. To suppress blank lines, select compact . To display blank lines, select loose .
Display	Select Wrap lines to make each line of input or output in the Command Window break into multiple lines to fit within the current width of the Command Window. For details, see “Wrap Lines of Code to Fit Window Width” on page 3-38.
	Select Set matrix display width to eighty columns to limit the width of matrix output. For details, see “Matrix Display Width” on page 3-38.
	Select Show getting started message bar to display the Command Window message bar that provides links to introductory information.
	 <p>New to MATLAB? Watch this Video, see Demos, or read Getting Started. ×</p>

Preference	Usage
	<p>Select Show function browser button to display this button  to the left of the prompt in the Command Window.</p> <p>See also, “Find Functions Using the Function Browser” on page 3-7.</p> <hr/> <p>Number of lines in command window scroll buffer specifies the maximum number of lines retained in the Command Window.</p> <p>For details, see “Set the Number of Lines in the Scroll Buffer” on page 3-33.</p>
Tab key	<p>Tab size specifies the number of spaces assigned to the tab key.</p> <p>For details, see “Number of Spaces Assigned to Tab Key” on page 3-39.</p>

Keyboard Preferences

Keyboard preferences enable you to set tab completion, function hints, and delimiter matching in the Command Window and Editor, as described in the table that follows.

To set Keyboard Preferences, select **File > Preferences > Keyboard**, and then adjust preference options as described in the table below.

Preference	Usage
Tab completion	<p>Select the tool or tools in which you want the Tab key to complete names known to MATLAB after you type the first few letters of the name.</p> <p>For details, see “Complete Names in the Command Window Using the Tab Key” on page 3-43 and</p> <hr/> <p>Select Tab key narrow completions to have MATLAB continue to reduce the list of possible names for completion as you type each additional character and press the Tab key.</p>
Function hints	<p>Specify, the selected tool or tools that you want to display syntax function hints.</p> <p>When enabled, if you type a function name with an opening parenthesis, and then pause, a tooltip opens showing the basic syntax for the function. For example:</p> <pre data-bbox="669 1156 1302 1459"> x = edit(edit('fun.m') edit('file.ext') edit('fun1','fun2','fun3',...) edit('classname/fun') edit('private/fun') edit('classname/private/fun') edit('+packagename/classname/fun') edit('my file.m') </pre> <p style="text-align: right;">More Help...</p>

Preference	Usage
	<p>For details, see “View Function Syntax Hints While Entering a Statement” on page 3-49.</p>
<p>Delimiter Matching</p>	<p>Specify when and if MATLAB alerts you to matched and mismatched delimiters. Delimiters include parentheses, brackets, braces, and, in the Editor only, paired keywords.</p> <p>If you select Match while typing, MATLAB alerts you to matched and mismatched delimiters as you type.</p> <p>If you select Match on arrow key, MATLAB alerts you to matched and mismatched delimiters when you move the cursor over a delimiter using an arrow key. For details, see “Avoid Mismatched Parentheses, Brackets, Braces, and Paired Keywords” on page 3-42.</p> <hr/> <p>Select one of these Show match with options to specify how MATLAB indicates matching delimiters:</p> <ul style="list-style-type: none"> • Balance — The corresponding delimiter highlights briefly. • Underline — Both delimiters in the pair display underlines briefly. • Highlight — Both delimiters in the pair highlight briefly. <hr/> <p>Select one of these Show mismatch with options to specify how MATLAB indicates mismatched delimiters</p> <ul style="list-style-type: none"> • Beep — MATLAB beeps. • Strikethrough — The delimiter you type appears crossed out briefly. • None — There is no alert.

Command History Preferences

You can exclude statements from the command history and specify how often to save the file in which the command history is stored, `history.m`. MATLAB uses the command history file for both the Command History window and statement recall in the Command Window. See also, “Command History File — `history.m`” on page 3-4

Note When you exclude statements from the command history file, you cannot recall them in the Command Window, nor can you view them in the Command History window.

To set Command History preferences, select **File > Preferences > Command History**, and then adjust preference options as described in the table below.

Preference	Usage
Settings	<p>Save exit/quit commands saves exit and quit commands in the command history.</p>
	<p>Save consecutive duplicate commands saves consecutive executions of the same statement in the command history.</p> <p>With this option selected, if you run <code>magic(5)</code> two times in a row, both entries for <code>magic(5)</code> remain in the command history.</p> <p>With this option cleared, the command history retains only one entry for <code>magic(5)</code>. If you then run <code>magic(10)</code>, the command history retains both entries.</p>

Preference	Usage
Saving	Save history file on quit saves the command history file only when you end a MATLAB session. If the session ends abnormally, such as due to a power failure, then the MATLAB does not save the history file for that session.
	Save after n commands saves the command history file after MATLAB adds n statements to it. Use this option instead of Save history file on quit if you do not want to risk losing entries to the saved history because of an abnormal termination.
	Don't save history file is useful when multiple users share the same machine. It prevents each user from viewing the statements others have run. Any entries already in the command history remain unless you first follow the instructions in “Delete Entries from the Command History Window” on page 3-34.

Help and Product Information

- “Ways to Get Function Help” on page 4-2
- “Run Examples and Demos” on page 4-3
- “Search Syntax and Tips” on page 4-5
- “Adjust Help Browser Fonts” on page 4-7
- “Bookmark and Share Page Locations” on page 4-9
- “Contact Technical Support” on page 4-11
- “Help Preferences” on page 4-13
- “Documentation on Japanese Systems” on page 4-15
- “Information About your Installation” on page 4-16

Ways to Get Function Help

Each MATLAB function has supporting documentation that includes examples and describes the function inputs, outputs, and calling syntax.

There are several ways to access this information from the command line:

- Open the function documentation in the Help browser by calling the `doc` command. For example,

```
doc mean
```


- Display function hints (the syntax portion of the function documentation) in the Command Window by pausing after you type the open parentheses for the possible function inputs.

```
mean(
```

- Open the Function browser by clicking the function icon to the left of the command prompt, f_x .
- View an abbreviated text version of the function documentation in the Command Window by calling the `help` command.

```
help mean
```

To access function documentation from the Editor, Command Window, or Help browser, select a function name from any part of the text, right-click, and then select **Help on Selection**.

Access the complete product documentation by clicking the help icon  or using the `doc` command.

Related Examples

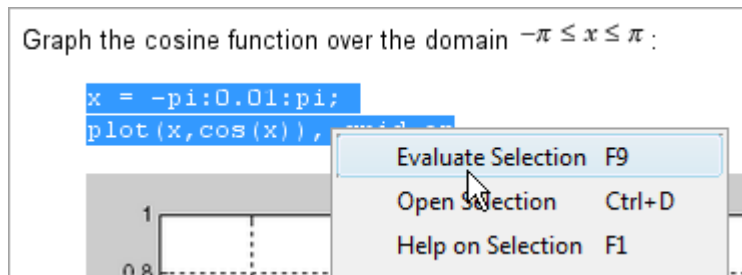
- “Run Examples and Demos” on page 4-3

Run Examples and Demos

This topic shows how to run examples and demos from the Help browser.

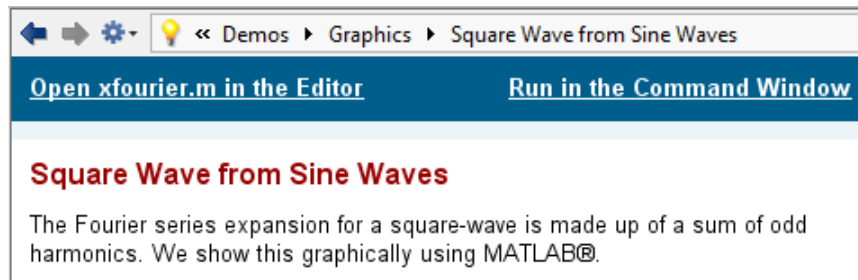
Examples



Run code from any page in the documentation by selecting the code, right-clicking, and then selecting **Evaluate Selection**.



Demos

There are two ways to run demos that are MATLAB scripts: from the Editor or in the Command Window. The Help browser provides links at the top of each demo to open or run the file.



Open *filename* in the Editor — Open the demo file in the Editor. For best results, run one cell at a time and view the incremental results. Select the first cell, and then step through the demo by clicking the Evaluate cell and advance icon, . Alternatively, run the entire file by clicking Run, .

Run in the Command Window — Step through the demo and view incremental results in the Command Window. Use links at the start of each step to continue or stop the demo (sometimes you must scroll within the Command Window to find the start of the step).

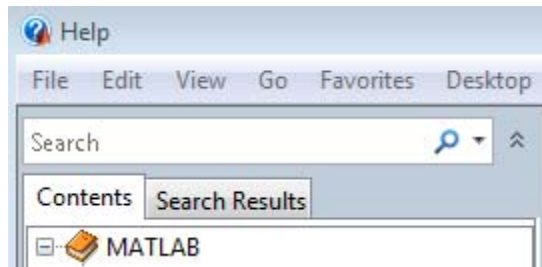
See Also demo

**More
About**

- “What Are Code Cells?” on page 8-58

Search Syntax and Tips

Find keywords in the documentation by entering text in the search field of the Help browser.



When you view the search results, search terms appear with highlights on the page. To clear the highlights, navigate to a different page or refresh the page.

The search engine ignores common, insignificant words such as *a*, *an*, *the*, and *of*, unless they are part of an exact phrase in quotation marks. It also ignores capitalization, punctuation, and special characters such as *+*. To find a symbol or special character:

- Search for the word instead of the symbol or character, such as **plus** instead of **+**.
- View the documentation on Operators and the Symbol Reference.
- Search the PDF documentation, available from the product page in the Help browser.

Searches can include the following operators:

- " " Exact phrase
Example: "plot tools" finds pages that contain *plot tools*, in that sequence, with no words between them.
- * Wildcard
Requires at least two nonwildcard characters, and cannot appear at the start of a keyword or in an exact phrase.
Example: plot* finds *plot*, *plot3*, and *plotting*.
- OR Boolean OR
Example: plot OR graph finds pages with either *plot* or *graph*.
- NOT Boolean NOT
Example: "plot tools" NOT "time series" finds pages with *plot tools* but excludes pages with *time series*.
- AND Boolean AND
Implied when no operator is present between keywords.
Example: plot AND tools is equivalent to "plot" "tools".

The Help browser search evaluates NOT operators first, OR operators second, and AND operators last. For example,

"plotting tool" OR "plot tools" NOT "time series" AND workspace
finds pages that contain either *plotting tool* or *plot tools* and contain *workspace*, but do not contain *time series*.

The search engine searches the following text in the documentation and demos:

- Documentation — Text and code shown in the Help browser
- Product demos — Comments and code in program files and models
- GUI-based demos — Help comments in the program file
- Video demos — Title

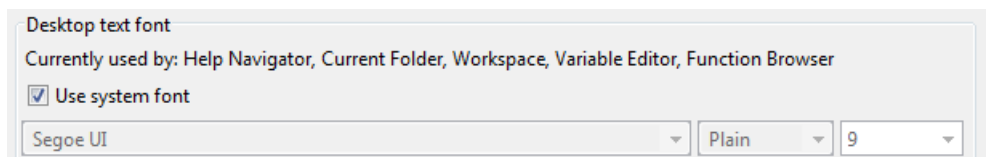
Adjust Help Browser Fonts

This topic shows how to change the fonts that appear in the panes of the Help browser. The preference that you set depends on whether you want to change the appearance of text in the Help Navigator pane (the **Contents** and **Search Results**) or the content viewing pane.

Help Navigator Pane

There are two ways to adjust the font for the Help Navigator Pane.

- Select **File > Preferences > Fonts** and set the **Desktop text font**. If you change the desktop text font, all of the tools listed in the pane inherit the change.



- Select **File > Preferences > Fonts > Custom**, and then select the **Help Navigator** tool. Specify custom settings for the name, style, or size of the font, as shown.



Content Viewing Pane

Specify the font for the Help browser viewer pane by selecting **File > Preferences > Fonts > Custom**, and then selecting the **HTML Proportional Text** tool. By default, HTML Proportional Text tools use

a custom font (Sans Serif, 10 pt.). You cannot specify colors for the text, background, or hyperlinks in the viewing pane.

Not all of the options affect the display of the content:

- The font family applies to text, but not to code. However, the font size applies to both text and code.
 - Bold and italic styles do not affect text in the viewing pane.
-
- “Fonts” on page 2-2
 - “Color Settings” on page 2-8

More About

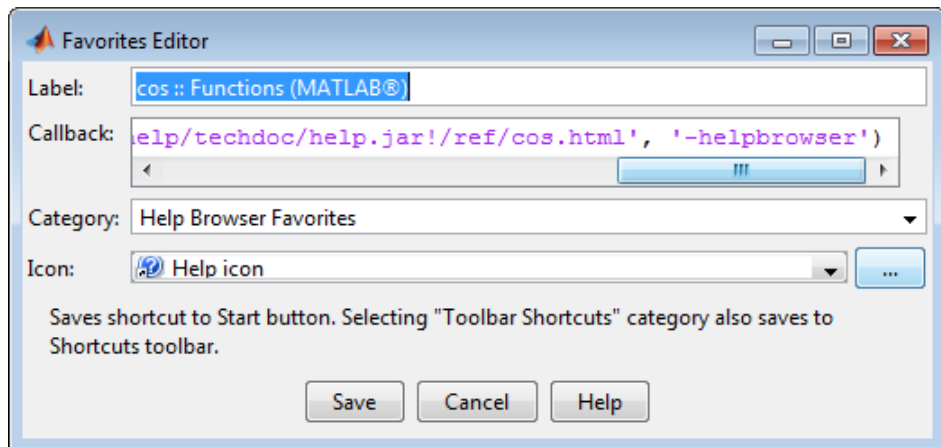
Bookmark and Share Page Locations

This topic shows how to store or identify the location of a page in the Help browser for future reference.

Bookmark Favorite Pages

In MATLAB, bookmarks are called *favorites*. Add, find, and organize favorites from the **Favorites** menu in the Help browser.

When you add a favorite, do *not* change the **Callback**. MATLAB requires special values to create a shortcut that opens the page in the Help browser. In addition, if you want the bookmark to appear in your list of favorites, keep the **Category** set to **Help Browser Favorites**, as shown.



Note You cannot migrate favorites that you save in one MATLAB release to a new release.

View Page Locations

Identify the location of a page in the Help browser to share with someone else by selecting **View > Page Location**. The Help Page Location window provides two ways to access the page:

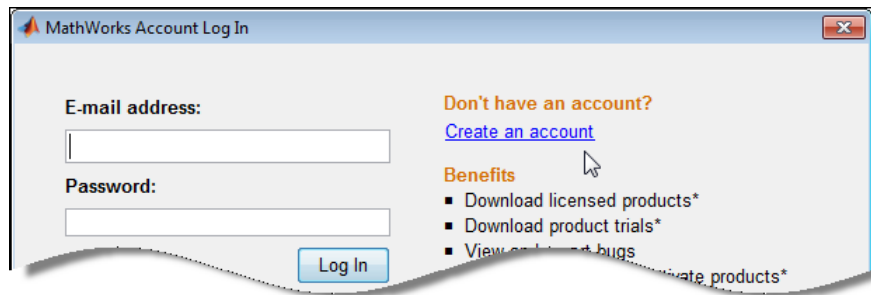
- A **web** command to run from the command line that opens the page from the installed documentation. This command is subject to change between releases, so it is not always accurate for someone running a different version of MATLAB.
- A URL for the page corresponding to your product version at the MathWorks Web site. This documentation is available to anyone, even if they do not have MathWorks products.

Note If you are running a prerelease version, the URL is invalid because the documentation does not yet exist on the Web site.

Contact Technical Support

This example shows how to contact MathWorks Technical Support to report a bug or request help.

- 1 Select **Help > Submit a MathWorks Support Request**.
- 2 When requested, log in using your MathWorks Account email address and password. If you do not have a MathWorks Account, create one.



- 3 Provide information to help technical support reproduce your issue, such as a description of the steps you followed or a code excerpt. Optionally, you can attach up to five files to your request, where each file is no larger than 3 MB. To submit files larger than 3 MB, upload them to the MathWorks FTP site.

Submit a MathWorks Support Request

Logged in as: I

Summary:

Function foo produces unexpected results

Description: ⓘ

When I call function foo as follows:

```
% code start  
myinput1 = 1;  
myinput2 = 2;  
myoutput = foo(myinput1, myinput2)  
%code end
```

I get an error message: Undefined function 'foo' for input argu

Product:

Please attach your related files:

- 4 Specify the product that is related to the issue.
- 5 Submit the request. If you do not have Internet access, save the support request form, and then email the form to support@mathworks.com.

External Links

- How do I access the MathWorks FTP site?
- MathWorks Support Page

Help Preferences

To set Help preferences, select **File > Preferences > Help**, and then adjust preference options as described in the table below.

Preference	Usage
Filter by Product	<p>Select the products that you want to include in the Help browser and Function browser.</p> <p>The Release Notes entry in the product list refers to the general Release Notes document, not the product-specific release notes.</p>
Quick Help Display	<p>Specify whether help links open content in the Help browser or in a small window. This preference applies to links from:</p> <ul style="list-style-type: none"> • Context menus, such as Help on Selection • Function hints or the Function browser • Error messages <p>Links to help from the Current Folder browser or the Help browser always open in the Help browser.</p>
Search History	<p>Specify how many terms to store in your search history, which persists from previous sessions. Recent search terms are visible in the search drop-down list in the Help browser.</p>
PDF reader (UNIX systems only)	<p>Specify the full path to the application for reading the PDF version of documentation, which you can access from each product page.</p> <p>On Microsoft Windows and Apple Macintosh platforms, MATLAB obtains the PDF reader location from the operating system.</p> <p>On UNIX platforms, MATLAB searches for Adobe® Acrobat® software to set the default reader.</p>
Language (Japanese systems only)	<p>Specify whether documentation in the Help browser and context-sensitive help should appear in English or Japanese. The installed Japanese documentation is not always current.</p>

**More
About**

- “Documentation on Japanese Systems” on page 4-15

Documentation on Japanese Systems

Many MathWorks products provide Japanese versions of the documentation. However, the current version of the documentation usually is not available in Japanese until about two months after the initial release of a new product version.

The new version of most products installs the Japanese documentation from the *previous* version and the English documentation for the *current* version. To view the English documentation, select **File > Preferences > Help** and set **Language** to English.

The **Language** preference is available when the system locale is Japanese and the Japanese documentation is installed. The preference changes the language only in the Help browser, context-sensitive help, and some of the text in the **Start** menu. (The `help` command provides help in Japanese.) If the documentation for a product is not translated, the Help browser displays the English documentation.

When the translations of the latest documentation are available, you can download them from the MathWorks Web site at <http://www.mathworks.co.jp/help>.

For information about documentation in other languages, contact your MathWorks sales and service office.

Information About your Installation

MATLAB software can tell you what products are installed, their versions, and other information about your license and platform. This information is important to have in the event you contact technical support.

Type of Information You Want	To Get the Information
Version and license for Installed product	From the product, select Help > About . Or use functions: <ul style="list-style-type: none"> • <code>license</code> — for the license number • <code>ver</code> — for version numbers for MATLAB and libraries • <code>version</code> — for version numbers for MathWorks products
MATLAB platform	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 arch value, for example win32. Or use the <code>computer</code> function.
Passcodes and licenses	From any desktop tool, select Help > Web Resources > MathWorks Account .

For related information concerning system requirements, and procedures for installing, activating, and uninstalling products, see the installation guide for your computer platform.

Workspace Browser and Variable Editor

- “MATLAB Workspace Browser” on page 5-2
- “Viewing and Editing Workspace Variables with the Variable Editor” on page 5-22
- “Workspace Browser and Variable Editor Preferences” on page 5-45

MATLAB Workspace Browser

In this section...

“MATLAB Workspace and Workspace Browser” on page 5-2

“Viewing and Editing Values in the Workspace” on page 5-3

“Improving Workspace Browser Performance” on page 5-6

“Saving the Current Workspace” on page 5-6

“Viewing a Saved Workspace” on page 5-7

“Loading a Saved Workspace and Importing Data” on page 5-7

“Changing and Copying Variable Names in the Workspace” on page 5-8

“Deleting Workspace Variables” on page 5-8

“Creating Plots from the Workspace Browser” on page 5-9

“Opening Variables and Objects for Viewing and Editing” on page 5-20

MATLAB Workspace and Workspace Browser

The MATLAB workspace consists of the set of variables built up and stored in memory during a MATLAB session. You add variables to the workspace by using functions, running MATLAB code, 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*.

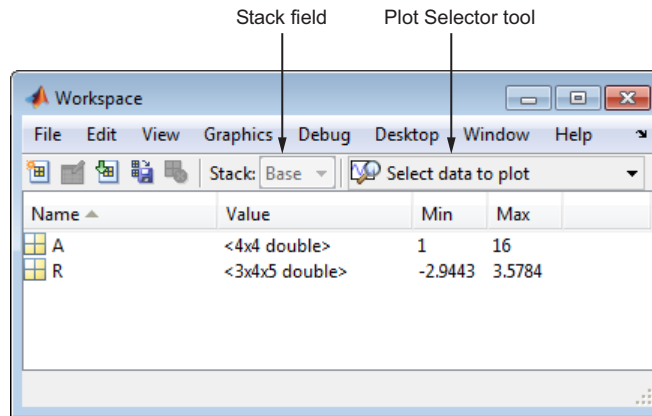
The Workspace browser enables you to view, change, and plot MATLAB workspace values.

To open the Workspace browser, do either of the following:

- Select **Desktop > Workspace**.
- Type `workspace` at the Command Window prompt.

By default, the Workspace browser displays the base workspace. The **Stack** field enables you to view function workspaces if MATLAB is in debug mode, but otherwise appears dimmed. For more information, see “Debugging Process and Features” on page 8-116 and the `dbstack` and `evalin` functions.

To generate a plot of one or more variables in the Workspace browser, use the Plot Selector tool. For more information, see “Creating Plots from the Workspace Browser” on page 5-9.



Viewing and Editing Values in the Workspace

For each variable or object, the Workspace browser shows:

- The name
- The class (also represented by the icon)
- The Value
- Statistics — such as the **Min**, **Max**, and **Mean** calculations, when relevant

MATLAB performs these calculations using the `min`, `max`, and `mean` functions, and updates the results automatically.

The following table describes how to edit values and adjust columns in the Workspace browser table:

To:	Do this:
Edit variable values in the Value column	Position the pointer in the Value column at the row you want to edit, click, and then type the new value.
View more of the data for a variable (as well as to more easily edit it)	Double-click a variable name. The variable opens in the Variable Editor. For more information, see “Viewing and Editing Workspace Variables with the Variable Editor” on page 5-22.
Display additional columns Such as size (dimensions), size in bytes, mode and standard deviation	Select View > Choose Column.. On Microsoft Windows systems, you can also right-click any column header.
Specify how NaNs are considered in statistical calculations	Select File > Preferences > Workspace , and select an option for Handling NaNs when calculating statistics .
Limit the array size for which statistics are calculated	See “Improving Workspace Browser Performance” on page 5-6.
Resize a column	Drag the border of the column header.
Reorder columns	Drag a column header to a new position.
Sort columns	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.

Include or Exclude NaN Values in Statistical Calculations

If your data includes NaNs, you can specify that the Workspace browser statistical calculations consider or ignore the NaNs. Select **File > Preferences > Workspace**, then select one of the following:

- **Use NaNs when calculating statistics**

If a variable includes a NaN, and you select this option, the values for **Min**, **Max**, **Var** and some other statistics will appear as NaN. However, **Mode**, for example, shows a numeric result.

- **Ignore NaNs when calculating statistics**

If a variable includes a NaN, and you select this option, numeric results appear for most statistics including **Min** and **Max**. **Var**, however, is still appears as NaN.

Function Alternative

- To list the current workspace variables, use `who`.

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	

- To list the variables and information about size and class, use `whos`.

For example:

```
>> whos
```

Name	Size	Bytes	Class	Attributes
A	4x4	128	double	
C	1x3	348	cell	
R	3x4x5	480	double	

Improving Workspace Browser Performance

If you show statistical columns in the Workspace browser, and you work with very large arrays, you might experience performance issues when the data changes as MATLAB updates the statistical results. To improve performance, consider one or both of the following:

- Show only the statistics of interest to you.

Select **View > Choose Columns**, and then clear the statistics you do not want MATLAB to calculate.

- Exclude large arrays from statistical calculations.


Select **File > Preferences > Workspace**, and then use the arrow buttons to change the value of the maximum array size for which you want the Workspace browser to perform statistical calculations. Any variable exceeding the maximum array size reports `<Too many elements>` in Workspace browser statistics columns instead of statistical results.

Saving the Current Workspace

The workspace is not maintained across sessions of MATLAB. When you quit MATLAB, the workspace clears. 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.

You can save all or selected workspace variables, as follows:

- Save all workspace variables—click the Save button  in the Workspace browser toolbar.
- Save selected workspace variables—Select the variables in the Workspace browser, right-click, and from the context menu, select **Save As**.

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, select **File > Preferences > General > MAT-Files**.

Function Alternative

To save workspace variables, use the `save` function followed by the file name you want to save to. For example, to save all current workspace variables to the file `june10.mat`:

```
save('june10')
```


Viewing a Saved Workspace

To view a MAT-file without loading it into the workspace, do one of the following:

- Select the MAT-file in the Current Folder browser.
The contents display in the Details panel.
- Use `whos` with the `-file` option.

Loading a Saved Workspace and Importing Data

You can load entire MAT-files or selected variables using one of these methods:

- To load MAT-files using the desktop:
 - 1 Click the Import Data button  on the Workspace browser toolbar.
 - 2 Select the MAT-file you want to load and click **Open**.

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.

See also, “Tips for Using the Import Wizard”.

- To load MAT-files programmatically
Use `load`.

For example, to load all workspace variables from the file `june10.mat`:

```
load('june10')
```

- To load selected variables:
Drag them from the Current Folder browser Details panel to the Workspace browser.
- To import data you previously copied to the clipboard
Select **Edit > Paste to workspace**, or use **Ctrl+V**.
MATLAB imports the clipboard data using the Import Wizard.

Changing and Copying Variable Names in the Workspace


To rename a variable in the workspace:

- 1 Right-click the variable in the Workspace browser, and then select **Rename** from the context menu.
- 2 Type the new variable name over the existing name, and then press **Enter**.

To copy variables from the clipboard, select the workspace variables, and then 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 variables:
 - To select multiple variables, use **Shift+click** or **Ctrl+click**.
 - To select all variables, select **Edit > Select All**.
- 2 Press the **Delete** key on your keyboard or click the Delete button  on the Workspace browser toolbar.

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, to clear the variables `A` and `M` from the workspace:

```
clear A M
```

Use the `clearvars` function with the `-except` option to keep specified variables and clear all other variables.

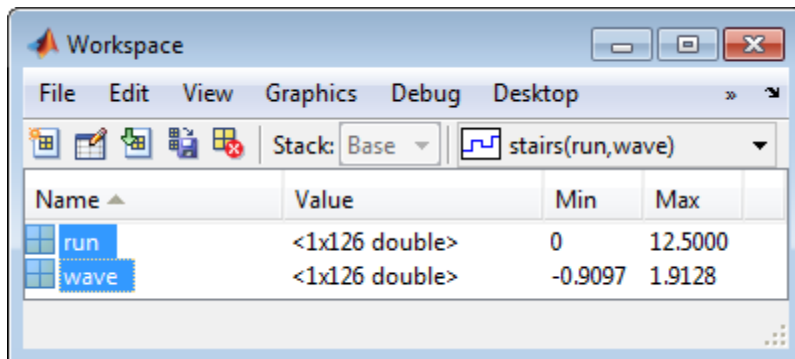
Creating Plots from the Workspace Browser

You can generate a plot of one or more variables with the *Plot Selector* tool  `Select data to plot` 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:

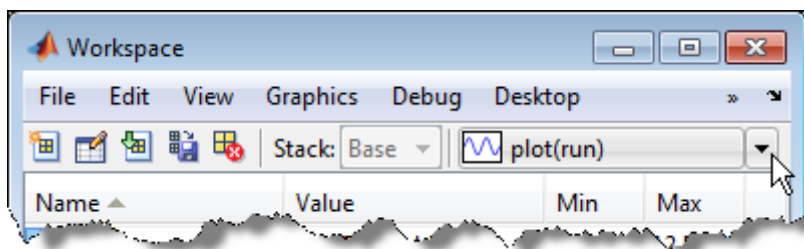
- 1 Create two vector variables:

```
run = 0:.1:4*pi;  
wave = (sin(run.^2) + cos(run).^2);
```

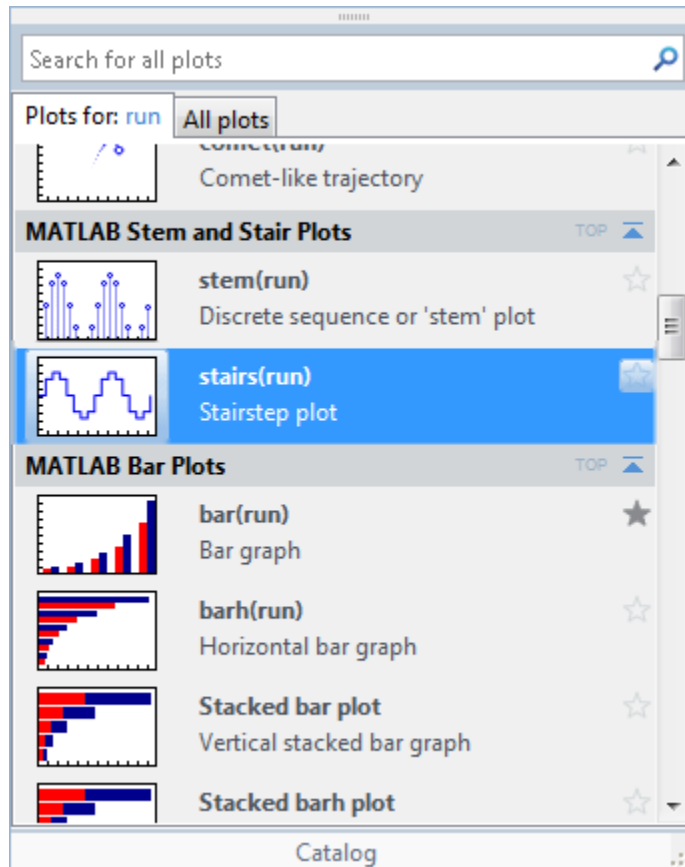
- 2 Select one or both variables in the Workspace browser. You can **Shift**+click or **Ctrl**+click to select multiple variables to plot as x , y , or z components of a graph.



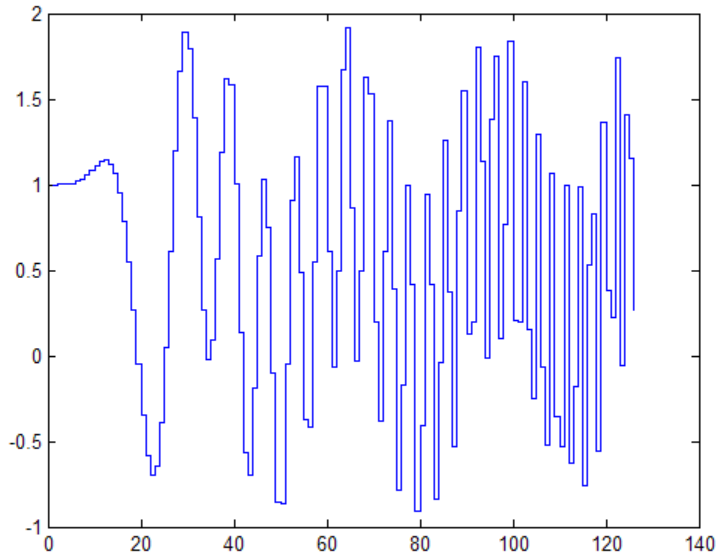
- 3 Click the down arrow icon in the Plot Selector on the toolbar.



- 4 In the Plot Selector menu, scroll to the graph type you want to display and double-click the icon on its left side.



A figure window opens with a graph of the selected variable or variables, using the plotting function you just chose.



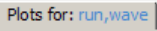





The Plot Selector GUI remains open until you click a desktop component or another window. This enables you to experiment with plot types without reopening the GUI each time.


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. The following table also describes what you can accomplish with the Plot Selector and how to do it.

To:	Do This:
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.

To:	Do This:
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.	<p>With one or more selected variables, select the first tab, Plots for:<variables>  and do either of the following:</p> <ul style="list-style-type: none"> • 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.	<p>With zero or more variables selected, select the second tab, All plots  and do either of the following:</p> <ul style="list-style-type: none"> • 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

To:	Do This:
<p>Add a graph type to Favorites.</p>	<p>In either tab:</p> <ul style="list-style-type: none"> • 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. <p>All of these actions create a copy the item in Favorites without removing it from its category.</p>
<p>Remove a graph type from Favorites.</p>	<p>In the Favorites category of either tab do either of the following:</p> <ul style="list-style-type: none"> • Click the star  at the right side of a menu item • Right-click a menu item and select Remove Favorite <p>You cannot drag a menu item out of Favorites.</p>
<p>Change the order of menu items within a category.</p>	<p>Click a menu item outside the graph icon and drag it to a new position within its category.</p>
<p>Move a menu item to a different category.</p>	<p>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.</p>

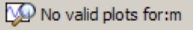
To:	Do This:
Get help for a graphing function.	On either tab, hover mouse pointer over a menu item. A syntax description pops up next to the item. To see the complete function reference page, click the <i>More Help</i> link in the description header.
Enable a dimmed menu item.	Select one or more variables 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.

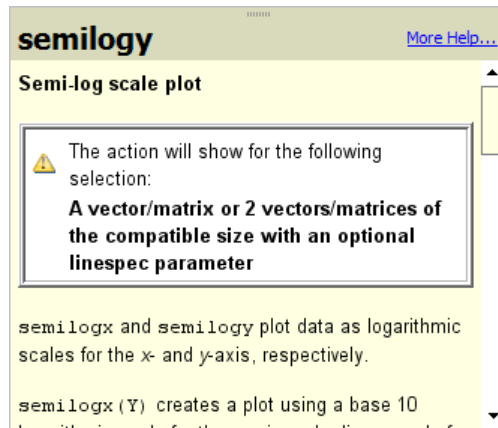
In the **All plots** tab, you can do several different things:

- 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

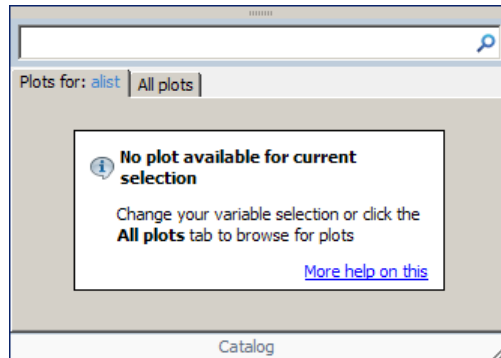
Even when the Plot Selector tool cannot successfully make a graph, it can still give you a starting point for doing so. You can drag any Plot Selector menu item into the Command Window to see the code it generates. Once the plotting command displays in the Command Window, it does not execute until you 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 `colourspec` 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 then edit the plotting command before executing it.

1 In the Command Window, enter

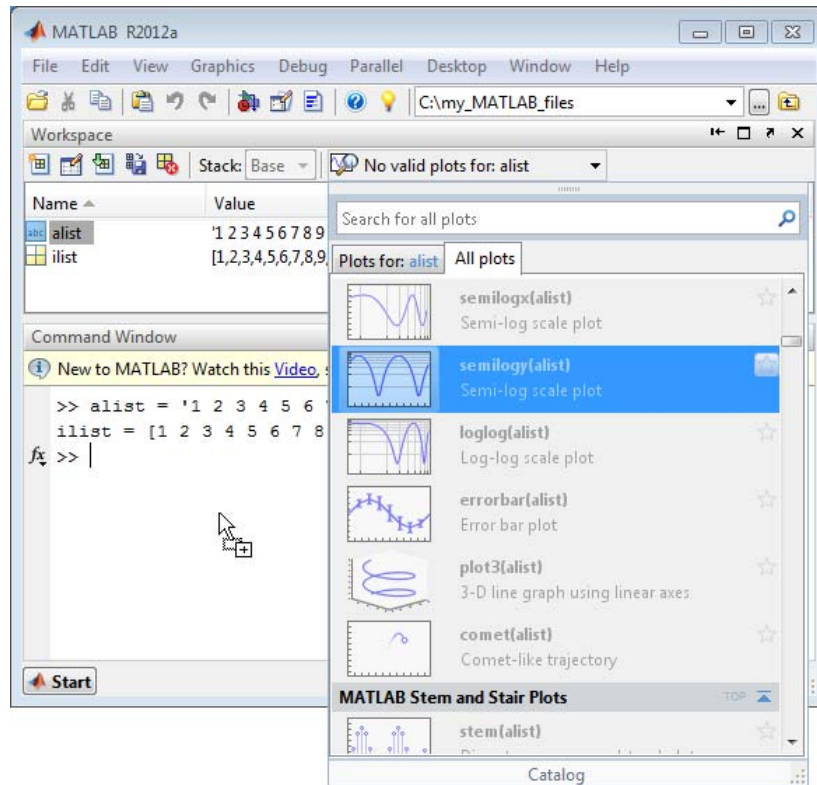
```
alist = '1 2 3 4 5 6 7 8 9 10';
ilist = [1 2 3 4 5 6 7 8 9 10];
```

- 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 Click `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, 'alist'); figure(gcf)
```

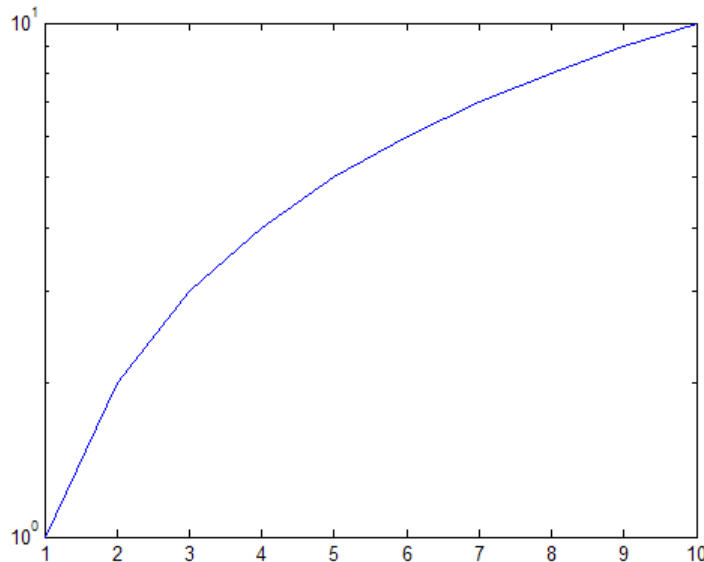
If you press **Enter**, an error displays because `alist` is not a proper calling argument.

```
Error using semilogy
Invalid first data argument
```

- 6 Instead, change the code to the following, substituting `ilist` for `alist`, and then press **Enter**:

```
semilogy(ilist, 'ilist'); figure(gcf)
```

This call works with the changed arguments and generates the following graph.



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.

Opening Variables and Objects for Viewing and Editing

In the Workspace browser, double-click a variable to open it in the Variable Editor. There, you can view and edit the contents of the variable. See

“Viewing and Editing Workspace Variables with the Variable Editor” on page 5-22 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.

Viewing and Editing Workspace Variables with the Variable Editor

In this section...

“About the Variable Editor” on page 5-22

“Opening the Variable Editor” on page 5-22

“Working with Different Data Types in the Variable Editor” on page 5-25

“Navigating the Variable Editor Using Shortcut Keys” on page 5-32

“Changing Size, Content, and Format of Variables in the Variable Editor” on page 5-33

“Selecting and Modifying Values in the Variable Editor” on page 5-34

“Exchanging Variable Editor Data with Other Tools and Applications” on page 5-39

“Creating Graphs and Variables, and Data Brushing in the Variable Editor” on page 5-40

About the Variable Editor

The Variable Editor enables you to:

- Display variables in the current workspace.
- View and edit values of one or two-dimensional arrays, character strings, cell arrays, structures, and objects and their properties.
- View the contents of multidimensional arrays.
- Copy and paste data values.


Edits you make in the Variable Editor immediately update the variable in the workspace.

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:

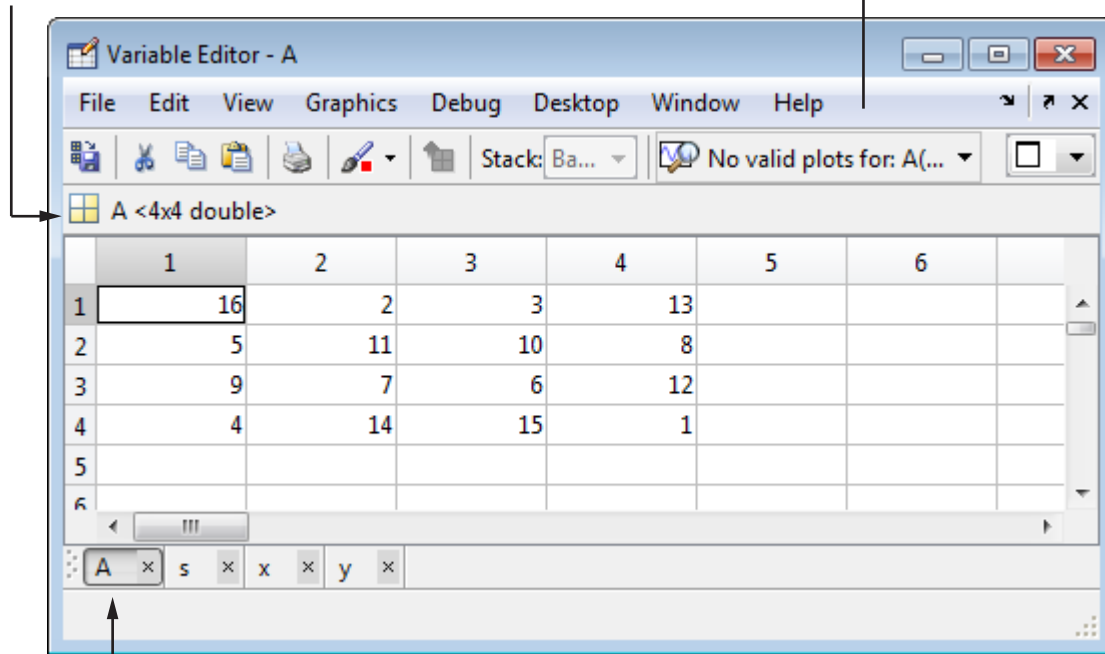

```
A = magic(4);  
x = 0:.1:4*pi;  
y = sin(x);  
s = sprintf('This is yext\nwith two lines');
```

- 2** If the Workspace browser is not open, select **Desktop > Workspace**.
- 3** 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.
- 4** 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.

Variable size
and class

Plot Selector



Tabbed document

- 5 Click the Plot Selector to create a graph of the selected variables.
- 6 Click a tabbed document to display a different variable that is open in the Variable Editor.

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.

Function 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')
```

To see the contents of a variable in the workspace, type the variable name at the Command Window prompt. For example, type:

```
A
```


MATLAB returns:

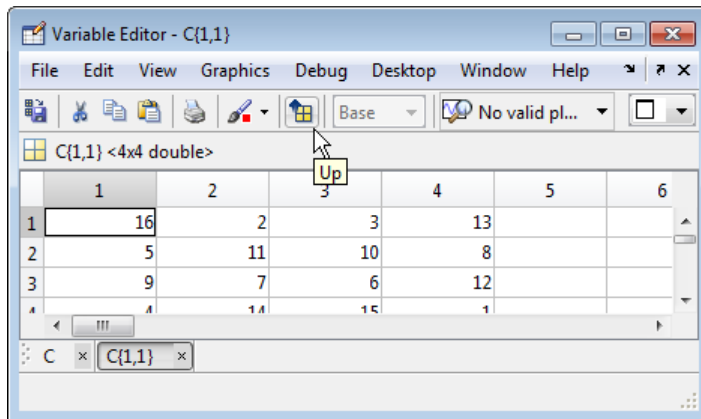
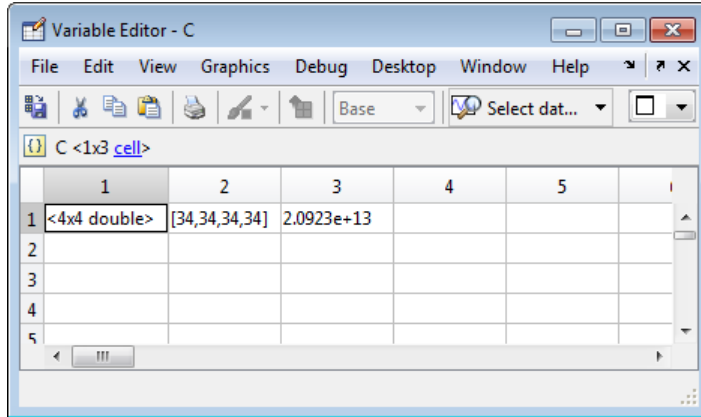
```
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Working with Different Data Types in the Variable Editor

- “Cell Arrays — Viewing and Editing in the Variable Editor” on page 5-25
- “Structures — Viewing and Editing in the Variable Editor” on page 5-26
- “Objects and Their Properties — Viewing and Editing in the Variable Editor” on page 5-28
- “Multidimensional Arrays — Viewing in the Variable Editor” on page 5-31

Cell Arrays — Viewing and Editing in the Variable Editor

To view and edit the content of cell arrays in the Variable Editor, double-click a cell array element. It opens it a new 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

To view and edit the content of a structure currently displaying in the Variable Editor, following the procedure exemplified here:

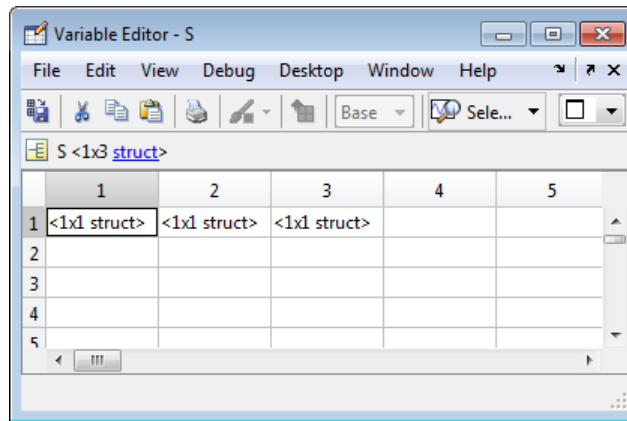
- 1 Create a structure by running the following code in the Command Window:

```
S.name = 'Ed Plum'  
S.score = 83;  
S.grade = 'B+'  
S(2).name = 'Toni Miller';  
S(2).score = 91;
```

```
S(2).grade = 'A-';  
S(3).name = 'Bill Cox';  
S(2).score = 92;  
S(3).grade = 'A-';  
S(3).score = 92;  
S(2).score = 91;
```

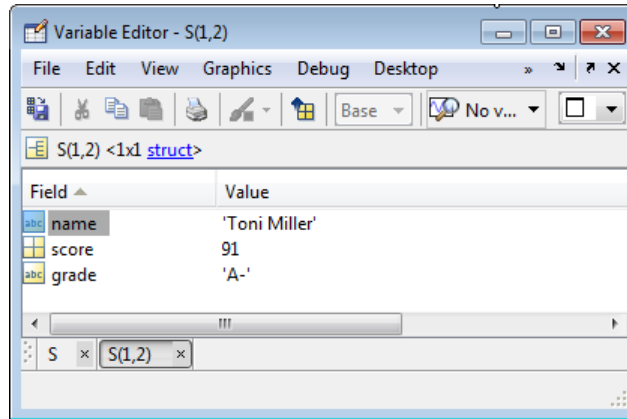
- 2 Open S in the Variable Editor by double-clicking S in the Workspace Browser.


The Variable Editor opens and displays S.



- 3 Display S(1,2), by double-clicking the element in row 1, column 2.

A new Variable Editor document opens and displays S(1,2).



- 4 Edit the value for the `grade` field by right-clicking the value, and then select **Edit Value** from the context menu.
- 5 Adjust the columns in the structure element document:
 - Sort columns by clicking a column header. Click the header again to reverse the sort order.
 - Show or hide columns by right-clicking any column header, and then, selecting or deselecting column header names, respectively. (This feature is available on Microsoft Windows platforms only.)
- 6 Return to the structure document by clicking the **Up** button .

Objects and Their Properties – Viewing and Editing in the Variable Editor

- “Viewing Object Properties in the Variable Editor” on page 5-28
- “Editing Property Values in the Variable Editor” on page 5-30
- “Getting Help for Objects and Properties from the Variable Editor” on page 5-31

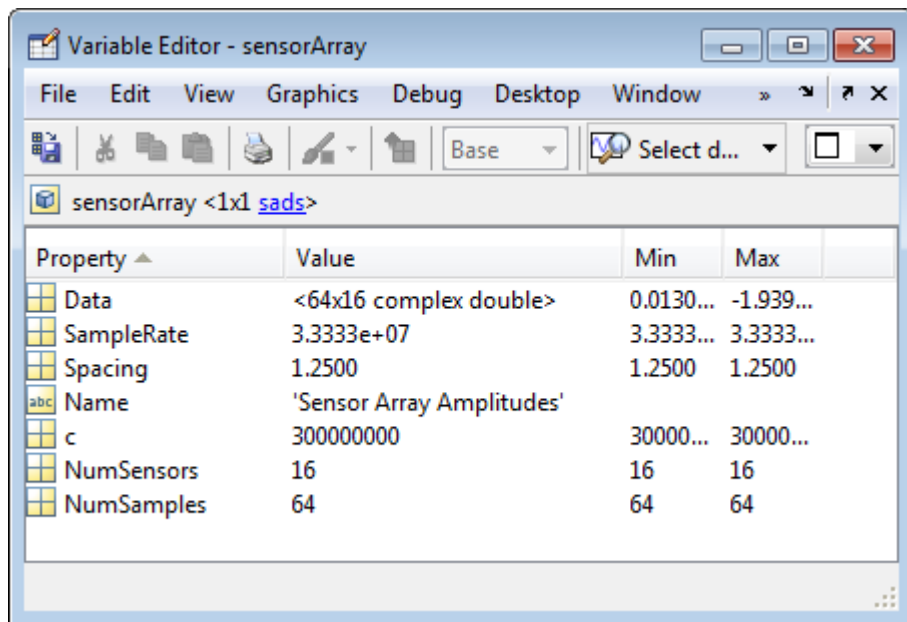
Viewing Object Properties in the Variable Editor. In the Variable Editor, you can view and edit properties of many MATLAB objects you create.



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.
- Sort by a column, click that column header; to reverse the sort order, click the column header again.
- View help for the class, click the class name link.

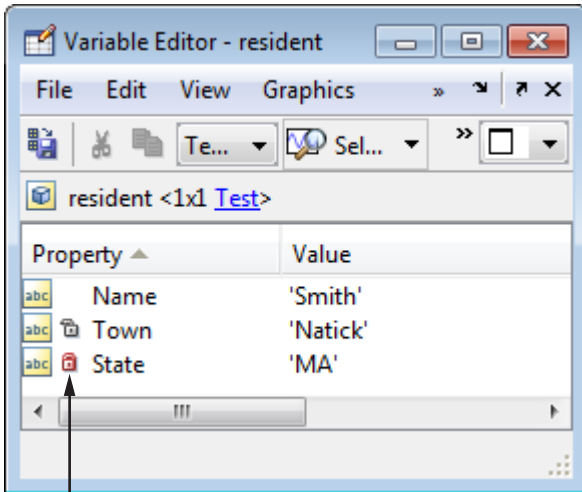
The Variable Editor has special attributes for `timeseries` objects; for more information, see “Viewing Time Series Objects”.

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 Externally Supplied Class”.




Lock icons, which can appear during debugging, denote protected  and private  properties of an object, indicating you do not have get access to those values outside class methods. The following illustration shows an object,

resident. The Town property is protected and the State property is private. An attempt to get the State property programmatically returns the error, Getting the 'State' property of the 'Test' class is not allowed.



Lock icon

Editing Property Values in the Variable Editor. To edit a property value while viewing the object, click the value field and then edit its contents. To more easily view a value before editing, double-click the value to display it in its own document, and then edit it.

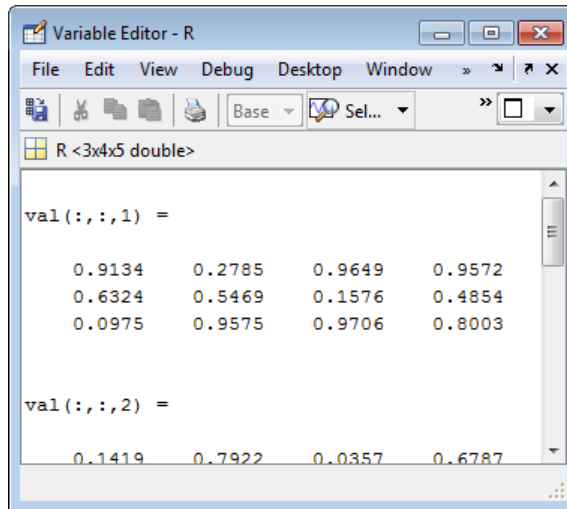
When viewing a property, use the **Up** button  to view the object. This button can help you navigate in the Variable Editor when there are many variables open.

If the Variable Editor window is small, the **Up** button might not be visible. To access it, click the >> button on the right side of the toolbar, and then select **Go Up One Level** from the menu.

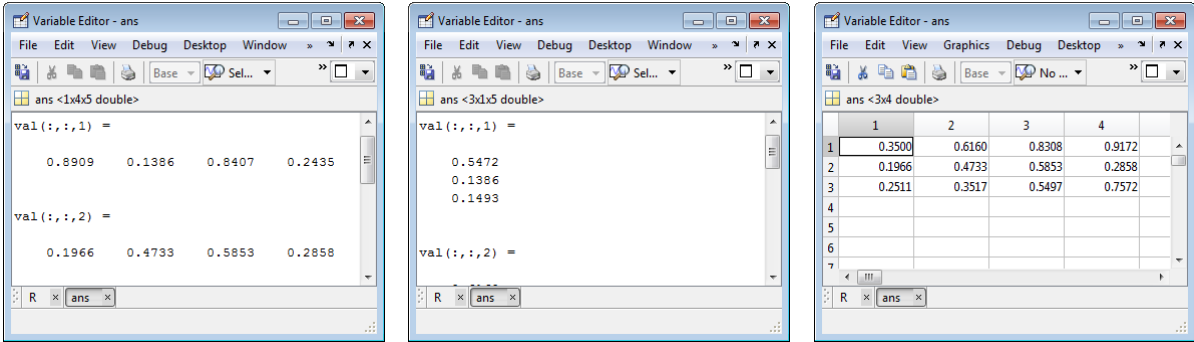
Getting Help for Objects and Properties from the Variable Editor. For most classes supplied by The MathWorks®, when you click the link to the class name, for example, `char`, the reference page displays in the Help browser. For user-created classes, help comments supplied in the class definition file display in HTML format in the Help browser. For more information, see “Help for Classes You Create”.

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 the 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)` open in the Variable Editor.



You can view subsets of multidimensional arrays as long as the indexing expression evaluates 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 the Variable Editor Using Shortcut Keys

To move among elements in the Variable Editor, use the following shortcut keys (sometimes called hot keys). Navigating in the Variable Editor is much like navigating in the Microsoft Excel application. You cannot modify these keyboard shortcuts. For information on additional Variable Editor keyboard shortcuts (which you can modify), see “Keyboard Shortcuts” on page 2-36.

To:	Press this Key:
Commit changes to an element and move to next element. “Variable Editor Preferences” on page 5-46 enable you to specify what the next element is (the default is down).	Enter
Move right. Within a selection, also moves from the last column to the first column in the next row.	Tab

To:	Press this Key:
Move in opposite direction of Enter or Tab .	Shift+Enter or Shift+Tab
Move up m rows, where m is the number of visible rows.	Page Up
Move down m rows, where m is the number of visible rows.	Page Down
Move to column 1.	Home
Move to row 1, column 1.	Ctrl+Home
Move to last column in current row.	End
Edit current element, positioning cursor at the end of the element.	F2 (Ctrl+U on Apple Macintosh platforms)




Changing Size, Content, and Format of Variables in the Variable Editor

To:	Do This:
Increase the size of an array.	<p>Scroll to the desired element in the variable and enter a value.</p> <p>Empty elements fill with zeros if numeric, or empty arrays if a cell array.</p>
Decrease the size of an array.	<ol style="list-style-type: none"> 1 Click in the row or column header of the rows or columns that you want to remove. 2 Right-click, and then select Delete from the context menu.

To:	Do This:
Change the value of an element.	<ol style="list-style-type: none"> 1 Click the element, and then type a new value. 2 Press Enter, or click another element.
Specify where the cursor moves to after you press Enter .	<p>Select File > Preferences > Variable Editor, and then select Editing options:</p> <ul style="list-style-type: none"> • To keep the cursor in the element where you typed, clear the Move selection after Enter check box. • To move the cursor to another element, select the Move selection after Enter check box. Then, in the Direction field, specify how you want the cursor to move.
Change the display format.	From the View menu, select a format.
Change the default format.	Select File > Preferences > Variable Editor , and then select a Format option.
Make MAT-file changes permanent.	See “Saving the Current Workspace” on page 5-6.

Selecting and Modifying Values in the Variable Editor

You can cut, copy, paste, insert, or delete selected elements, rows, and columns in an array, as described in the following table.

To:	Do This:
Select a column or row.	Click the row or column header (the element that shows the row or column number).
Select contiguous elements, rows, or columns in an array.	Click the header of the first row, and then Shift +click additional rows.
Select all elements.	Select Edit > Select All .
Cut selected elements.	<p>Select Edit > Cut or click the Cut button .</p> <p>The value of each element you cut becomes 0 if numeric or [] if a cell array.</p> <p>The cut values move to the clipboard.</p>
Copy selected elements.	Select Edit > Copy or click the Copy button  .
Paste cut or copied elements into selected elements.	<p>Select Edit > Paste or click the Paste button .</p> <p>If the shape of the elements you cut or copy differs from the shape of the elements into which you are pasting, the Variable Editor pastes all the elements. Either the Variable Editor expands the size of the selection you made, or it expands the array size to allow all the elements you are pasting.</p>

To:	Do This:
Insert elements, rows, and columns in arrays.	<ol style="list-style-type: none"><li data-bbox="877 335 1302 423">1 Click or select an area in the Variable Editor where you want to make an insertion.<li data-bbox="877 458 1176 484">2 Select Edit > Insert.<li data-bbox="877 519 1317 607">3 Complete the Insert dialog box to specify what you want to insert, and then click OK. <p data-bbox="872 644 1326 800">If you insert an <i>entire</i> row or column, the Insert dialog box does not open. MATLAB inserts a row above the select row or a column to the right of the selected column.</p>
Delete elements, rows, and columns in arrays.	<ol style="list-style-type: none"><li data-bbox="877 852 1273 909">1 Click or select the element or elements you want to delete.<li data-bbox="877 944 1181 970">2 Select Edit > Delete.<li data-bbox="877 1005 1322 1093">3 Complete the Delete dialog box to specify what you want to delete, and then click OK. <p data-bbox="872 1130 1326 1251">If you delete an <i>entire</i> row or column, the Delete dialog box does not open. MATLAB shifts columns or rows to replace the deleted column or row.</p>
Undo the last operation.	Select Edit > Undo , or click the desktop toolbar Undo button.

To:	Do This:
Redo the last operation.	Select Edit > Redo , or click the desktop toolbar Redo button.
Set the value of elements to 0.	Select elements, rows, or columns for which you want to set the value to 0, and then select Edit > Clear Contents .

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 In the Workspace Browser, double-click A.

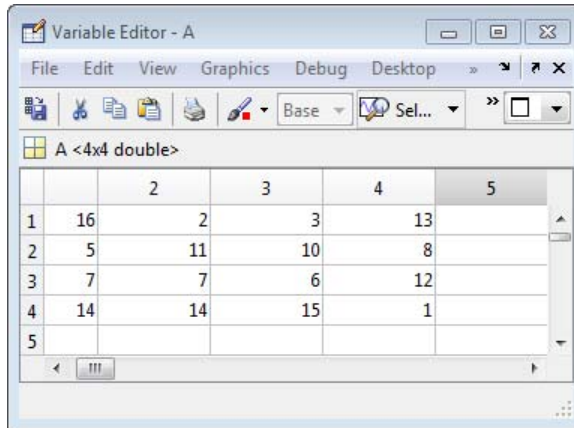
- 3 In the Variable Editor, select and copy these two elements from rows 3 and 4:

- a Click 7 (A(3,2)) and then **Shift**+click 14 (A(4,2)).

- b Right-click the selection, and then select **Copy** from the context menu.

- 4 Select 9 (A(3,1)), right-click, and then select **Paste** from the context menu.

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

In this example, you cut two rows and paste them into a single row. The Variable Editor expands the array size, adding a row to accommodate all pasted elements. The values of the elements you cut become 0.

- 1 Create a matrix variable.

```
A = magic(4);
```

- 2 In the Workspace Browser, double-click A.

- 3 In the Variable Editor, select rows 2 and 3:

- a Click row number 2, and then **Shift**+click row number 3.
- b Right-click the selection and select **Cut** from the context menu.

The values in the cut rows all become 0 as a result of the cut operation.

- 4 Select row number 4, and then right-click and select **Paste** from the context menu. The contents of the cut rows replace row 4 and extend the matrix to one additional row.

The screenshot shows the Variable Editor window titled "Variable Editor - A". The menu bar includes File, Edit, View, Graphics, Debug, and Desktop. The toolbar contains icons for file operations and a dropdown menu set to "Base". Below the toolbar, the variable name "A" is shown with its dimensions "<5x4 double>". The main area displays a table with 6 rows and 5 columns. The first row is highlighted in grey. The data in the table is as follows:

	2	3	4	5
1	16	2	3	13
2	0	0	0	0
3	0	0	0	0
4	5	11	10	8
5	9	7	6	12
6				

Exchanging Variable Editor Data with Other Tools and Applications

You can exchange Variable Editor data with other desktop tools and external applications if the data types are compatible. For example, you cannot paste text from the Command Window or an Microsoft Excel spreadsheet into a numeric array in the Variable Editor.

Tip If you cut and paste Variable Editor values into text files or other applications, you can change the character that delimits decimals in the data that is exported. You might do this, for instance, if you provide data to a locale that uses a character other than the period (.). Select **File > Preferences > Variable Editor**, and then change the character for **Decimal separator for exporting numeric data via system clipboard**.

- To exchange data with the Command Window, do either of the following:
 - Copy data from an array in the Variable Editor and paste it into the Command Window.
 - Copy a value from the Command Window and paste it into an element in the Variable Editor.
- To create new workspace variables from the Variable Editor:

- 1 Select an element, data range, row, or column in an array in the Variable Editor.
 - 2 Right-click, and then from the context menu, select **Create Variable from Selection**.
- To exchange data with a Microsoft Excel spreadsheet, do either of the following:
 - In the Microsoft Excel application, cut or copy cells, and then in the Variable Editor, select **Edit > Paste from Excel**
 - Cut or copy elements from an array in the Variable Editor and paste them into the Microsoft Excel application

Creating Graphs and Variables, and Data Brushing in the Variable Editor

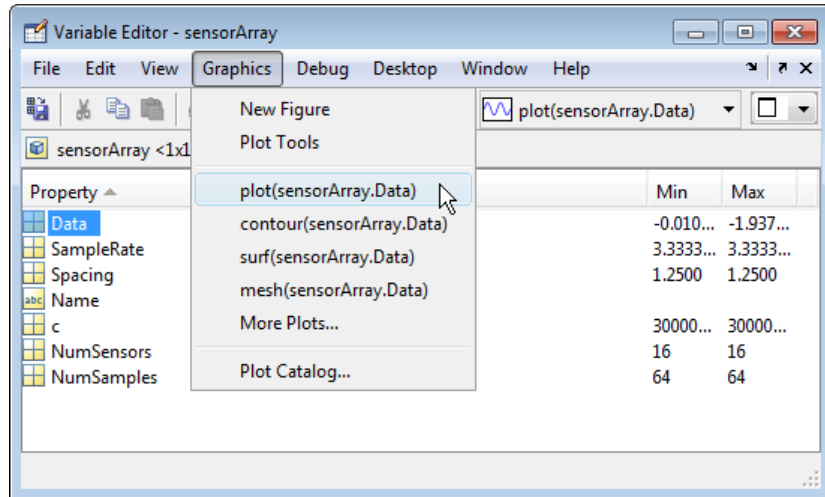
The Variable Editor, like the Workspace Browser, provides several methods for creating graphs without typing plotting commands. Once a graph displays, you can “brush” either the graph or array elements in the Variable Editor to see which observations correspond in the other.

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.

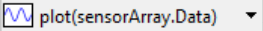
You can graph selections of numeric data and selected objects from the Variable Editor in three ways, illustrated here:

- Select data, and then choose from a list of graph types from the **Graphics** menu.

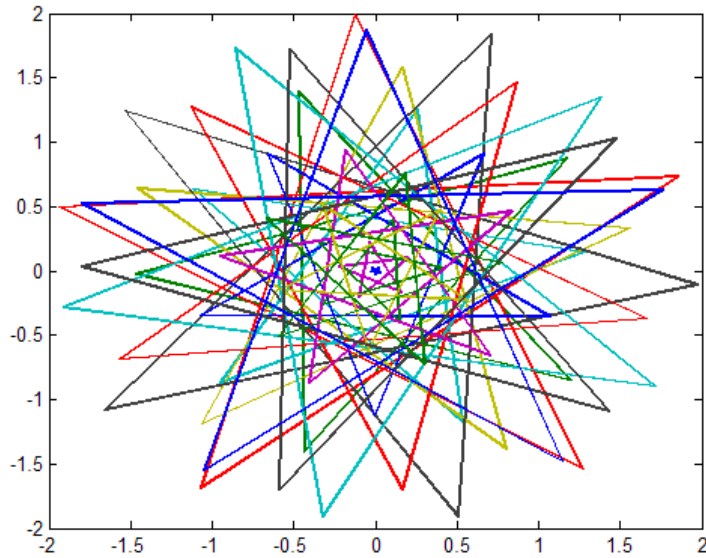


- Select data, and then right-click, and choose from a list of graph types from the context menu.

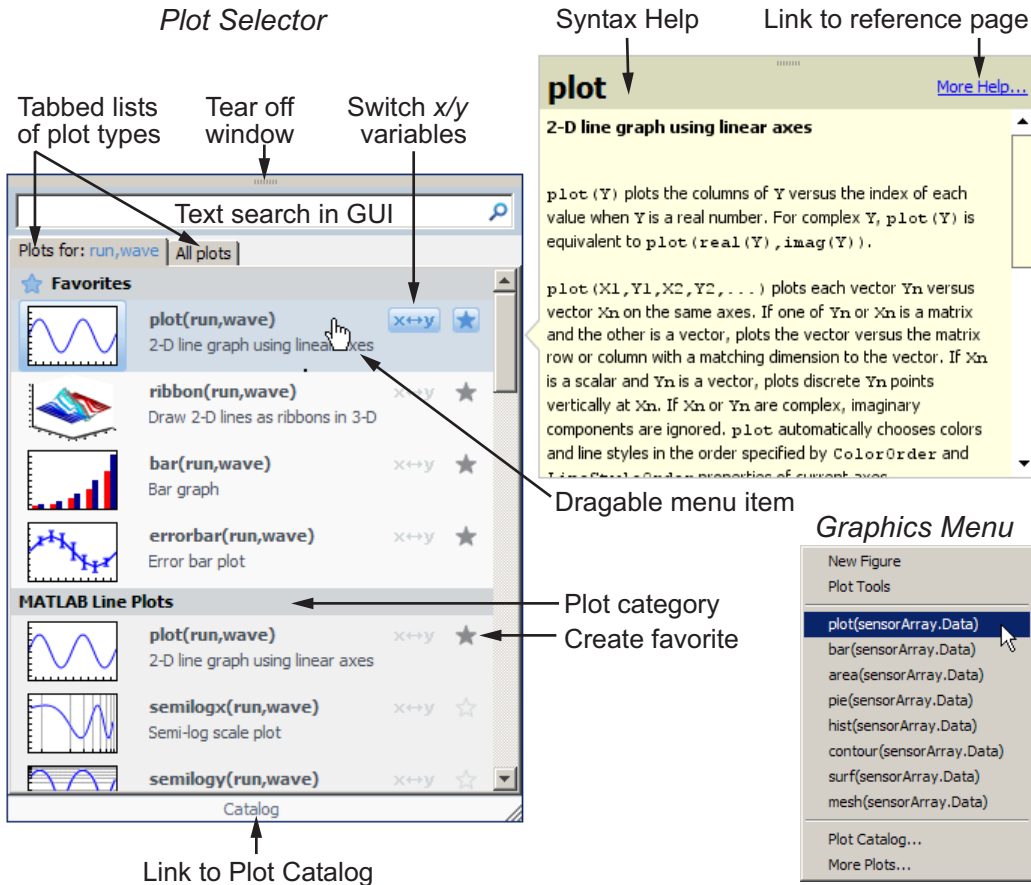
The types of graphs available on the context menu and the **Graphics** menu are the same.

- Select data, and then click the Plot Selector toolbar button  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 that 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 5-9.

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” and the reference pages for brush and linkdata.

Workspace Browser and Variable Editor Preferences

In this section...

“Workspace Browser Preferences” on page 5-45

“Variable Editor Preferences” on page 5-46

Workspace Browser Preferences

Workspace browser preferences enable you to restrict the size of arrays on which you perform calculations and to specify if you want those calculations to include or ignore NaNs.

To open Workspace browser preferences, select **File > Preferences > Workspace**.

Preference	Usage
<p><i>n</i> element and smaller arrays show statistics</p>	<p>Limit the size of arrays for which the Workspace browser displays statistics to improve performance when MATLAB updates the statistical results in the Workspace browser.</p> <p>For more information, see “Improving Workspace Browser Performance” on page 5-6.</p>
<p>Handling NaN values in calculations</p> <ul style="list-style-type: none"> • Use NaNs when calculating statistics • Ignore NaNs when calculating statistics 	<p>Specify whether NaN values be included or excluded from calculations for the statistics displayed in the Workspace browser.</p>

Variable Editor Preferences

Variable Editor preferences enable you to specify the array formatting, cursor movement, and the decimal separator for exporting data using the system clipboard.

To open Variable Editor preferences, select **File > Preferences > Variable Editor**.

Preference	Usage
Format	Select an option from the Default array format to 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 information on formatting options, see the reference page for the <code>format</code> function.
Editing	Specify where the cursor moves to after you type an element, and then press Enter : <ul style="list-style-type: none"> • To keep the cursor in the element where you typed, clear the Move selection after Enter check box. • To move the cursor to another element, select the Move selection after Enter check box. In the Direction field, specify how you want the cursor to move.
International number handling	In the Decimal separator for exporting numeric data via system clipboard field, specify the decimal separator for numbers you cut or copy from the Variable Editor when you paste them into text files or other applications.

Preference	Usage
	This preference has no effect on numeric data copied from and pasted into MATLAB. Within MATLAB, decimal separators are always periods.

Managing Files in MATLAB

- “Introduction to Managing Files in MATLAB” on page 6-2
- “Understanding File Locations in MATLAB” on page 6-4
- “Working with Files and Folders” on page 6-14
- “Finding Files and Folders” on page 6-28
- “Creating, Opening, Changing, and Deleting Files and Folders” on page 6-35
- “Comparing Files and Folders” on page 6-47
- “Making Files and Folders Accessible to MATLAB” on page 6-64
- “Using the MATLAB Search Path” on page 6-70
- “Related Topics for Managing Files” on page 6-82

Introduction to Managing Files in MATLAB

In this section...

“Ways to Manage MATLAB Files” on page 6-2

“Tools for Managing Files” on page 6-2

Ways to Manage MATLAB Files

MATLAB functions and desktop tools help you:

- Find a file you want to open, edit, load, or run in MATLAB
- Organize your files
- Obtain information about the contents and status of a file
- Ensure that MATLAB can access a file so that you can run or load it.
Typically, any file you access must be in the current folder or in a folder that is on the MATLAB search path.

Tools for Managing Files

MATLAB provides many interactive tools for working with files. Click the name of a tool in the following table to learn more about it.

Desktop Tool	Description
Current Folder Browser	View files, perform file operations such as opening, finding files and viewing file content, and managing and tuning your files.
Find Files Tool	Locate files by folder, file type, size, text within them, and other criteria.
Comparison Tool	View line-by-line differences between two files.

Desktop Tool	Description
Editor	Create, edit, debug, and analyze files containing MATLAB language statements (functions and scripts), and view and edit other text files.
File Exchange	Access a repository of files, created by users for sharing with other users, at the MathWorks Web site.

Understanding File Locations in MATLAB

In this section...
“Important MATLAB Folders” on page 6-4
“Path Names in MATLAB” on page 6-7

Important MATLAB Folders


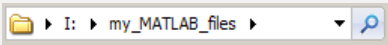
When you work with files and folders, be aware of key locations that MATLAB uses.

The Current Folder

The *current folder* is a reference location that MATLAB uses to find files. This folder is sometimes referred to as the *current directory*, *current working folder*, or *present working directory*. It is *not* the same location as the operating system current folder.

You can always load files and execute scripts and functions that are in the current folder, even if that folder is not currently on the MATLAB search path. Functions in the current folder take precedence over functions with the same file name that reside anywhere on the search path.

Viewing and Changing the Current Folder. You can view and change the current folder using various desktop tools and functions, as described in the following table. To specify the current folder programmatically when MATLAB starts, see “Startup Folder for the MATLAB Program” on page 1-12.

To:	Do this:
Identify the Current Folder	<p>Use one of the following:</p> <ul style="list-style-type: none"> • The Current Folder field on the Desktop toolbar.  <ul style="list-style-type: none"> • The Current Folder browser address bar — if the full path is not visible, hover over the folder icon.  <ul style="list-style-type: none"> • The pwd or cd function.
Change the current folder to one you specify	<p>Do one of the following:</p> <ul style="list-style-type: none"> • In the Current Folder field on the Desktop toolbar, type or browse to a different folder. • On the Current Folder browser address bar, click an arrow that appears between portions of the path, and then choose a drive or subfolder from the drop-down list. • Use the cd function.
Change the current folder to a recently used folder	<p>From the Current Folder field on the Desktop toolbar, click the down arrow, and then select a folder from the history.</p>
Change the current folder to an active document's folder	<p>Right-click the document tab in the Editor, and then select Change Current Folder to folder.</p> <p>Document tabs appear only when more than one document is open in the Editor.</p>
Make a subfolder the current folder	<p>In the Current Folder browser, right-click the subfolder, and then select Open from the context menu.</p>

To:	Do this:
Copy the Current Folder as a string	<ul style="list-style-type: none"> • Select the address in the Desktop toolbar, right-click, and then select Copy. • Click an empty area on the right edge of the Current Folder address bar, right-click, and then select Copy.
Get help using the Current Folder address bar	Right-click an empty area on the address bar, and then from the context menu, select Help Using Address Bar

matlabroot

matlabroot is the folder where you installed MATLAB. The location differs for each installation of MATLAB. Determine its location by running the *matlabroot* function. When you start MATLAB, your current folder can be *matlabroot*, but in practice it is usually a different folder.

The Startup Folder

Each time you start MATLAB, your current folder is always the same. This location is called the *startup folder*. The operating system commands that runs MATLAB specifies the location of the startup folder. You can configure MATLAB to make your initial current folder a different location. For more information, see “Startup Folder for the MATLAB Program” on page 1-12.

Locations of MathWorks Products

Files and folders for products provided by MathWorks are in *matlabroot/toolbox*. The files and folders under *matlabroot* are important to your installation. In particular:

- Do *not* store your personal files and folders in *matlabroot/toolbox*.
- Do *not* change files, folders, and subfolders in *matlabroot/toolbox*. The exception is the *pathdef.m* file, which you can update 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, running functions can produce unexpected results or generate warnings, that are related to the toolbox cache. See “Toolbox Path Caching in the MATLAB Program” on page 1-23.

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 folder called MATLAB 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 *userpath* MATLAB folder 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, you instruct MATLAB differently to set the current folder to *userpath* at startup. For more information, see “Startup Folder for the MATLAB Program” on page 1-12.

If you create subfolders within the MATLAB folder, make the new subfolders accessible to MATLAB.

If you store files in locations other than the MATLAB folder:

- Make the files accessible to MATLAB by adding their folders to the search path.
- Do not store the files in the folders provided for MathWorks products.

Path Names in MATLAB

A path name specifies file locations, for example, C:\work\my_data (on Microsoft Windows platforms) or /usr/work/my_data (on Linux or Apple Mac platforms). Path name specifications differ, depending on the platform on which you are running MATLAB. When you work with files and folders,

be aware of how MATLAB uses path names and the restrictions it places on them.

Specifying Path Names on Apple Mac Platforms

When you specify path names on Mac platforms, do not use accent characters. If path names include such characters, for instance umlauts or circumflexes, the Current Folder browser and MATLAB cannot recognize the path. In addition, attempts to save a file to such a path results in unpredictable behavior.

Specifying 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 Microsoft Windows platforms.

In the full path to a folder, the final slash is optional.

Type `filesep` in the Command Window to determine the correct file separator character to use when working with files programmatically.

Specifying Absolute and Relative Path Names

MATLAB always accepts *absolute* path names (also called *full* path names), such as `I:/Documents/My_Files`. An absolute path name can start with any of the following:

- UNC path '\\\ ' string
- Drive letter, on Microsoft Windows platforms, such as `C:\`.
- `'/'` character on Linux platforms

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 the `myfolder` folder in the current folder and `myfile.m` refers to the `myfile.m` file in the current folder.

- `../myfolder/myfile.m` refers to the `myfile.m` file in the `myfolder` folder, where `myfolder` is at same level as the current folder. Each repetition of `../` at the beginning moves up an additional folder level.

Tip If multiple documents are open and docked in the Editor, you can copy the absolute path of any of these documents to the clipboard. This is useful if you need to specify the absolute path in another MATLAB tool or an external application. Right-click the document tab, and then select **Copy Full Path to Clipboard**

Case Sensitivity of File Names

How MATLAB handles file names with respect to case depends on a number of factors. If you are unsure of how MATLAB will handle your specific case, it is best to specify path and case precisely when specifying a file name. The sections that follow describe how case affects common MATLAB operations.

Case Sensitivity of File Names When Calling a Function. You call function files by specifying the file name without the file extension. MATLAB assumes you want a case-insensitive match if it cannot find a case-sensitive match on the MATLAB search path (regardless of the operating system on which MATLAB is running).

Furthermore, if multiple files with the same name, but different extensions exist in the same folder, then MATLAB searches among the files in the folder in this precedence order:

- MEX-files
- MDL (Simulink model) files
- P-code files
- MATLAB code files (those with a `.m` extension)

For details on precedence, see .

For example, suppose `myfile.m` is on the search path, but `MYFILE.M` is not. If you type `MYFILE` at the MATLAB command prompt, then MATLAB runs

`myfile.m`, but warns you that there is a case mismatch and advises you that this warning will become an error in a future release.

If `myfile` (a MEX-file) and `MYFILE.m` are on the search path, and you type `MYFILE` at the command prompt, MATLAB runs `MYFILE.M`, even if `myfile` is higher on the search path.

To see which file MATLAB will use without running that file, use `which` with the `all` option. For example, `which myfile.m all`.

Case Sensitivity of File Names When You Load a MAT-File. When you call `load` and specify a file without an extension, MATLAB searches for a MAT-file. Case-sensitivity depends on the operating system where MATLAB is running, as follows:

- Linux

If you attempt to load `MYFILE`, and `MYFILE.MAT` is anywhere on the MATLAB search path, then MATLAB loads `MYFILE.MAT`. This is true, even if `myfile.mat`, and `myfile.m` are higher than `MYFILE.MAT` on the MATLAB search path.

If you attempt to load `MYFILE`, and `MYFILE.MAT` is not on the MATLAB search path, then MATLAB returns the message, `Unable to read file MYFILE.MAT: No such file or directory`. This is true even if `myfile.mat` is on the MATLAB search path.

- Windows

If you attempt to load `MYFILE` and `myfile.mat` is higher on the search path than `MYFILE.MAT`, then MATLAB loads `myfile.mat` without warning you that there is a case mismatch.

Case Sensitivity of File Names When You Save a MAT-File. When you call `save` and specify a MAT-file without an extension, MATLAB saves the file to the current folder. Case-sensitivity depends on the operating system where MATLAB is running, as follows:

- Linux

MATLAB saves the file using the case you specify. Two files with the same name, but different cases can exist in the same folder.

- Windows

Because the Windows operating system considers two files with the same name to be the same file (regardless of case), you cannot have two files with the same name in the same folder. If you save `MYFILE`, and `myfile.mat` already exists in the current folder, then `MYFILE.MAT` replaces `myfile.mat` without warning. If you save `myfile`, and `MYFILE.mat` already exists in the current folder, the contents of `myfile.mat` replace the contents of `MYFILE.mat`, but the name remains `MYFILE.mat`.

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 260 characters.
- For a relative path name, you might need to use fewer than 260 characters. When the Windows operating system processes a relative path name, it can produce a longer absolute path name, possibly exceeding the maximum length.

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.

Constructing Path Names on Different Platforms

Use `fullfile` to construct path names in statements that work on any platform. This function is particularly useful when you provide code to someone using it on a platform different from your own. The `ismac`, `ispc`, and `isunix` functions identify the platform you are currently using.

Including 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('temp file.m') % Function syntax works for a file name containing a space
```

The command syntax does not work. For example:

```
delete temp file.m % Command syntax does NOT work for a file name containing a space
```

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 multiple toolboxes contain functions with that name. For example, to get help for the `set` function in the Database Toolbox™ product, type:

```
help database/set
```

- Locate method files. For example, to get help for the time series object `plot` method type:

```
help timeseries/plot
```

Specifying the at sign character (@) in method folder names is optional.

- Locate private and method files, which sometimes are hidden.

Be sure to specify enough of the path name to make the partial path name unique.

See Also

- “Slash and Backslash — / \”

- “Naming Functions”
- `ismac`, `ispc`, and `isunix` functions, for MATLAB statements that require different path names for different platforms
- “Private Functions”.

Working with Files and Folders

In this section...
“Viewing Folder Contents” on page 6-14
“Using the Current Folder Browser” on page 6-20

Viewing Folder Contents

- “Opening the Current Folder Browser” on page 6-15
- “Preferences for the Current Folder Browser” on page 6-16
- “Refreshing the List of Files” on page 6-17
- “Viewing Hidden Files and Folders” on page 6-17
- “Controlling the Appearance of Files Inaccessible to MATLAB” on page 6-18
- “Using Functions to Get Details About Files and Folders” on page 6-20

You can view information about folders from the MATLAB Desktop like you can from operating system windows. You can also type commands to describe files as you can from a command shell. For a summary of MATLAB functions you can use, see “Using Functions to Get Details About Files and Folders” on page 6-20 .

The principal Desktop tool for working with files and folders is the *Current Folder browser*. Like other Desktop components, you can dock the Current Folder browser or open it as a separate window. The Current Folder browser displays details about files in your current folder and within the hierarchy of the folders it contains. You can modify the kinds of information it displays to suit your needs, for example by reordering or deleting specific columns of information.

The Current Folder browser:

- Always displays your current folder, as well as its subfolders.
- Lets you access operating system file management features from within MATLAB.

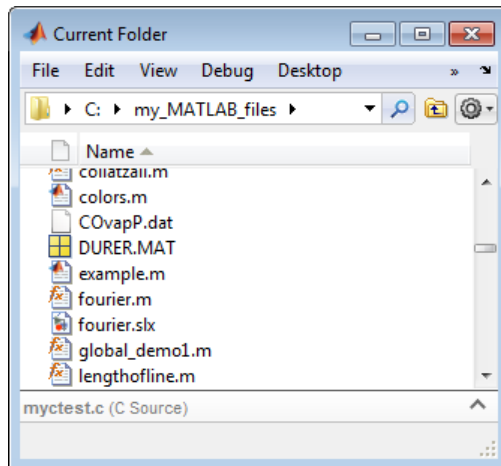
- Is similar to file browsers provided with operating systems, but also includes features unique to MATLAB. For example, you can add folders to the search path from the Current Folder browser.

The following sections explain what you can do with the Current Folder browser and how to use it.

Opening the Current Folder Browser

To open the Current Folder browser, select **Desktop > Current Folder** from the MATLAB desktop.

The Current Folder browser shows the full path to the current folder in the address bar, and shows the contents of the current folder in a pane underneath the bar.




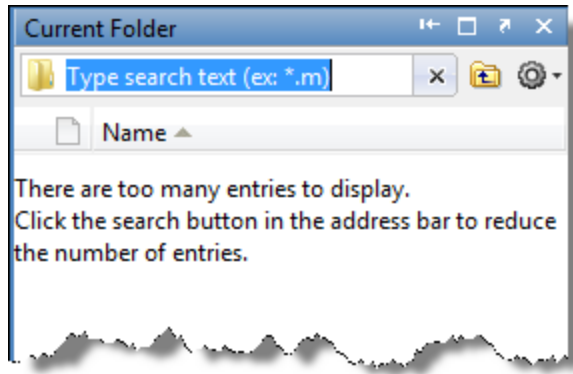
Double-clicking a subfolder displays its contents, and makes that folder the current folder.

If there are an exceptionally large number of entries to display, this message might appear in the Current Folder browser:

There are too many entries to display.

Click the search button in the address bar to reduce the number of entries.

When you click the search button , the address bar becomes a search field.



Preferences for the Current Folder Browser

You can set preferences for aspects of the Current Folder browser. Access these preferences by selecting **File > Preferences > Current Folder**. The preferences are:

- History — The number of recently used folders maintained in the Current Folder browser drop-down list.
- Refresh — How frequently the Current Folder browser updates to reflect changes to files made outside of MATLAB.
- Path indication— Controls the appearance of folders and files that are inaccessible to MATLAB, and whether to display tooltips describing their status.
- Toolbar — Provides a link to the Toolbars preferences. Those preferences enable you to adjust the toolbar layout and controls for Desktop tools, including the Current Folder browser.
- Hidden files — Controls whether the Current Folder browser displays hidden files and folders.

This preference not on available Microsoft Windows platforms.

Tip For information on changing the date format in the Current Folder browser, see “Customizing the Column Display” on page 6-21

Refreshing the List of Files

When files and folders are created, deleted, or changed 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. If this seems to be a problem, 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 improve responsiveness 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 manually 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 Hidden Files and Folders

The operating system, by default, hides certain files and folders from system file browsers and file-listing commands. The Current Folder browser can display hidden files and folders. You control this in different ways on different operating systems.

On Microsoft Windows platforms, the Current Folder browser follows the Windows preference for showing hidden files. To set or change the Windows preference:

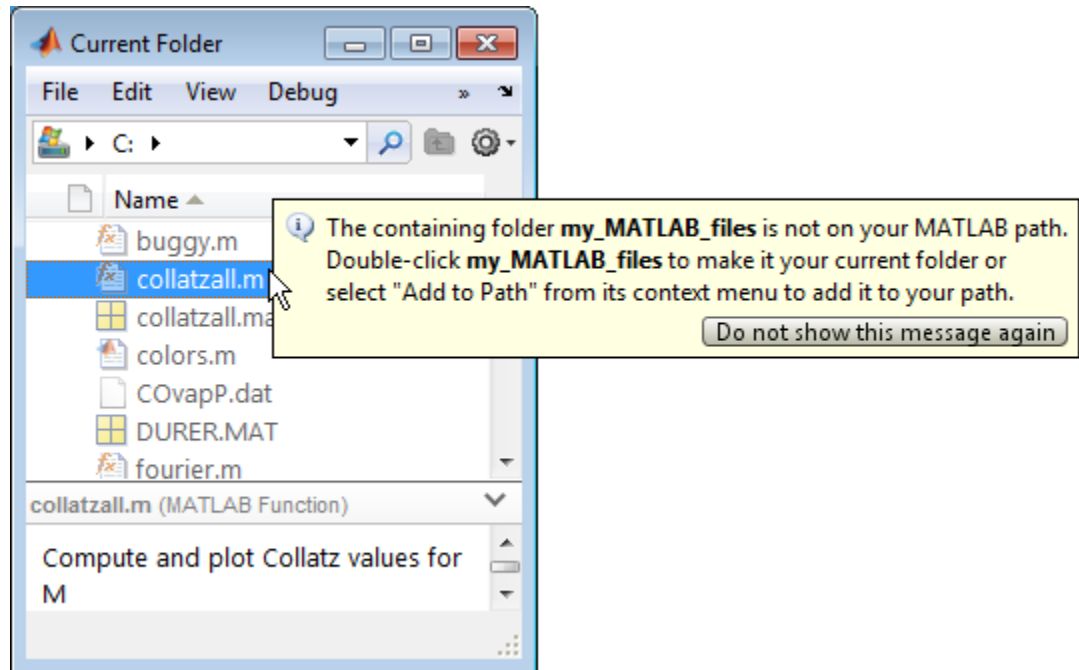
- 1 Open Windows Explorer.
- 2 Select **Tools > Folder Options**.
- 3 Click the **View** tab.
- 4 Under **Advanced** settings, select **Show hidden files and folders**.

On other platforms, specify the behavior using Current Folder preferences:

- 1 Select **File > Preferences > Current Folder**.
- 2 Specify the setting for **Hidden files and folders**.

Controlling the Appearance of Files Inaccessible to MATLAB

MATLAB cannot access files if they are not on the search path or, in some cases, if they are in a private folder. By default, the Current Folder Browser dims the display of files and folders inaccessible to MATLAB. Furthermore, if you hover over a dimmed file, a tooltip provides information on why that file is inaccessible. If you disable this feature, the Current Folder browser displays all files and folders as undimmed and provides no tooltips regarding their availability to MATLAB.



To customize this feature:

- 1 Select **File > Preferences > Current Folder**.
- 2 Select the **Indicate inaccessible files** check box to enable this feature; deselect it and skip to step 5 to disable this feature.
- 3 Move the **Text and icon transparency** slider to adjust the level of dimming.

View the region below the slider to preview how your choice will affect the appearance of files in the Current Folder browser.
- 4 Select **Show tooltip explaining why files are inaccessible** to enable tooltips; deselect it to disable them.
- 5 Click **OK**.

For more information, see “Private Folders” and “What Is the Search Path?” on page 6-70.

Using Functions to Get Details About Files and Folders

In the Command Window, you can list files, move to another folder, create folders, and delete files and folders. Most of these commands work like they do in an operating system command shell, but have different options. For example, some of the commands can return an argument in the form of a structure containing file information. The following table lists some of the actions you can perform with commands.

To...	Use This Function
Change your current folder	<code>cd</code>
Get the name, date, and size for a file or folder	<code>dir</code> or <code>ls</code>
Get and set the read-write status and other attributes for a file or folder	<code>fileattrib</code>
Separate a path name into folder and file name	<code>fileparts</code>
Build a file or folder name from parts	<code>fullfile</code>
Determine if a string corresponds to a folder	<code>isdir</code>
Move a file to a different folder	<code>movefile</code>
Create a folder	<code>mkdir</code>
Delete a file	<code>delete</code>
Delete a folder	<code>rmdir</code>
List files specific to MATLAB	<code>what</code>

Using the Current Folder Browser

- “Customizing the Column Display” on page 6-21

- “Viewing File Descriptions” on page 6-22
- “Viewing File Details Without Opening Files” on page 6-23
- “Viewing Help for a MATLAB Program File” on page 6-26
- “Sorting and Grouping Files and Folders” on page 6-26

The Current Folder browser lists details about files and folders in columns, beneath file and folder names, and in the details panel. Any file that is modified in the Editor, but not yet saved has an asterisk (*) next to it in the Current Folder browser. The browser displays columns for **Size**, **Date Modified**, **Type**, and **Description**. You can modify the information it displays. You also use this tool to perform operations on files and folders, such as moving, compressing, renaming, creating, and deleting them.

Note Do not use accented characters, such as à, é, ñ, or ü, in folder names. The Current Folder browser cannot locate folders containing such characters or save files to them.

Customizing the Column Display

You can show and hide columns, change their order, and adjust the date format in the Current Folder browser.

To Specify the Columns to Display.

- 1 Select **View > Show** or right-click on any column header.
- 2 Select the columns to show. Clear the columns to hide.

In addition, consider:

- Hiding the **Type** column if the icon column provides enough information about the type.
- Sorting or grouping by a column without showing the column.

Select **View > Group By** or **View > Sort By**. Then, choose the method by which you want to group or sort columns.

To Modify Columns.

- To change the order, drag a column header to a new position.
- To change the width, drag the edge of the column header.

To Change the Date Format. MATLAB uses your operating system's short date format to display dates in the Current Folder browser and the Command History window. To change the date format, for instance from MM/DD/YYYY to DD/MM/YYYY, (where MM is the numerical value for the month, DD is the numerical value for the day, and YYYY is the numerical value for the year):

- 1 Change the short date format for your operating system. For instructions, see your operating system documentation.
- 2 Refresh the date display by either restarting MATLAB or doing the following:
 - In the Current Folder browser, right-click, and then choose **Refresh** from the context menu.

Dates refresh and use the new format.
 - In the Command History window, right click, and then choose **Clear Command History** from the context menu.

The window clears. MATLAB specifies new dates in the window using the new format.

Viewing File 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. When the Current Folder browser window is wide enough, descriptions display on the same line as file names. The Current Folder browser shows descriptions only for files and folders that are relevant to products from MathWorks. How the Current Folder browser gets the description depends on the type of item:

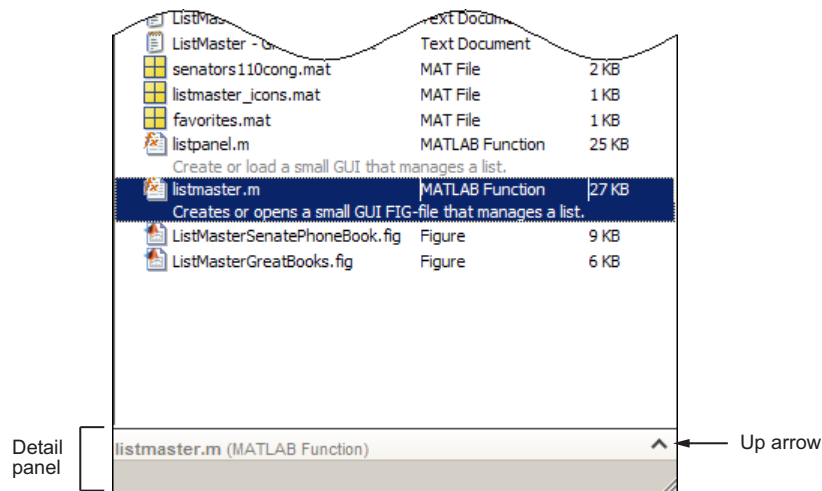
- **MATLAB program files** — The description is the first line of the help comments, known as the H1 line.

- **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 “Add Help for Your Program Files”.

Viewing File Details Without Opening Files

Display file details without opening a file by selecting the file, and then clicking the up arrow button on the lower right corner of the Current Folder browser. The details panel expands.



File and Folder Details. When you select a file or folder, the details panel displays more information about that file or folder, if possible. For example, if you select a MATLAB code file, the details panel shows the functions or subfunctions which that the file contains.

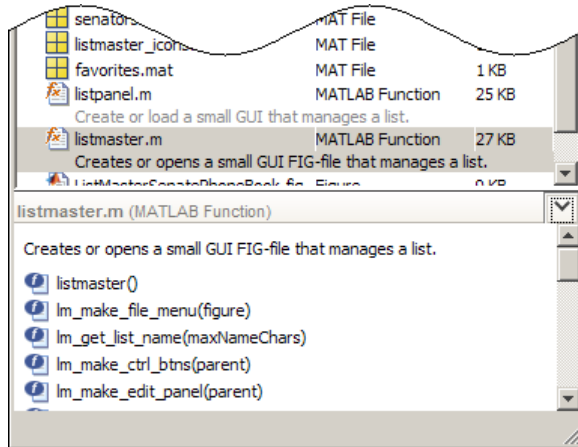
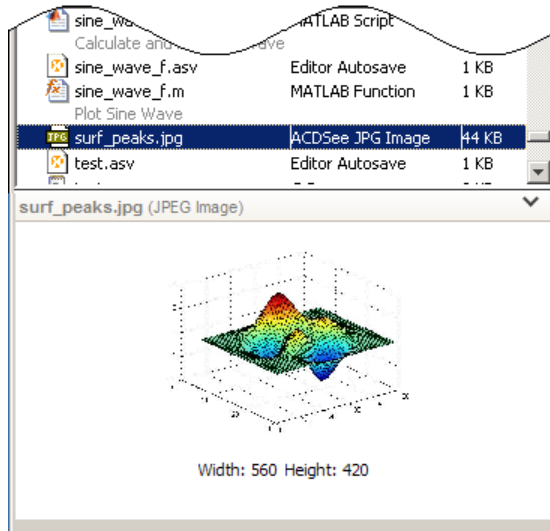
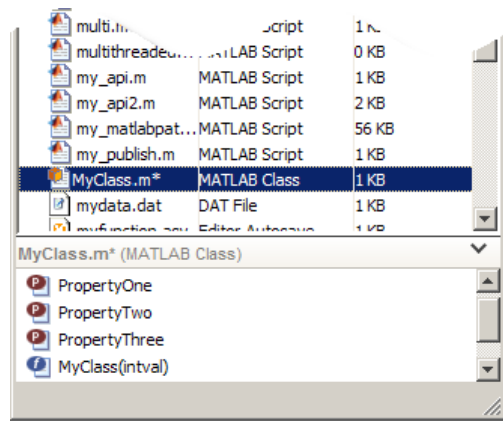


Image File Details. When you select a JPEG, JPG, BMP, WBMP, PNG, or GIF image, the details panel displays a thumbnail of the image and lists its width and height in pixels. To open the Import Wizard, double-click the thumbnail.







Viewing Unsaved File Changes. When you select a file that is currently open in the Editor and that contains unsaved changes, an asterisk (*) appears after that file name. The Current Folder browser columns reflect the content of the unsaved file. For instance, if you open a file, change it from a script to a function, and modify the H1 line, then the icon, type name, and description update in the Current Folder browser.

The preview in the details panel also reflects the unsaved file content, not the content on disk. For instance, in the following example, `PropertyTwo` exists in the modified `MyClass.m` file, but not the `MyClass.m` file on disk.



Viewing and Going to Elements within a MATLAB Program File. The details panel lists these elements when you selected a file with a `.m` extension:

- Subfunctions 
- Cells 
- Properties 
- Methods 

To open the file in the Editor, scroll to the start of the element in the details panel and double-click the element.

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. The folder containing the MAT-file does not need to be on the search path for you to load it in this way.

Viewing Help for a MATLAB Program File

From the Current Folder browser, you can view help for a file that has a .m extension and is in the current folder or in a folder on the search path:

- 1 Right-click the file.
- 2 Select **View Help** from the context menu.

The reference page, if it exists, opens in the Help browser. Otherwise, help comments from the beginning of the file, if any exist, display in the Help browser.

Sorting and Grouping Files and Folders

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.

You can sort and then group, or group and then sort.

Regardless of the sorting and grouping options selected, the Current Folder browser lists folders and files separately.

Tip To view *only* files of a certain type (for example, files having a .m extension) use a simple search. See “Simple Search for File and Folder Names in the Current Folder Browser” on page 6-28.

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.

Alternatively, click the column header by which you want to sort. Click it again to reverse the direction of sorting.

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**.

Finding Files and Folders

In this section...

- “Finding Files and Folders by Name in the Current Folder” on page 6-28
- “Simple Search for File and Folder Names in the Current Folder Browser” on page 6-28
- “Advanced Search for Files — Find Files Tool” on page 6-29
- “Locating a File or Folder in the Operating System Browser” on page 6-33
- “Finding Files and Folders Using Functions” on page 6-34
- “Additional Ways to Find Files” on page 6-34

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.

Typeahead and *find as you type* are other names for this feature.

Simple Search for File and Folder Names in the Current Folder Browser

To search for names that contain a specified series of characters in the current folder and subfolders:

- 1** In the Current Folder field, change the current folder to be the one you want to search.

- 2 Click the search button  in the address bar.
- 3 Type the absolute path name or begin typing a file name. The asterisk character (*) is a wildcard.

If you type a partial path name, such as `matlab\toolbox`, MATLAB regards it as a file name.

- 4 Press **Enter**:
 - If you typed a path name, that path becomes the current folder.
 - If you typed a file name, MATLAB displays all files within the current folder (including its subfolders) that match that file name.
- 5 Continue filtering the list by doing either of the following:
 - Typing additional characters
 - Removing characters you already typed

For example, to show only file names that begin with `coll` and have a `.m` extension, type `coll*.m`

- 6 Clear the results and show all items in the current folder by pressing the **Esc** key.

Instant search and *filtering* are other names for this feature.

Advanced Search for Files – Find Files Tool

To look for a specified string in file names and within files located in multiple folders, select **Edit > Find Files**, which opens the Find Files tool. The following sections provide details on using the tool.

- “Steps for Using the Find Files Tool” on page 6-30
- “Opening Files from the Results List” on page 6-31
- “Accessing Previous Results” on page 6-32
- “Skipping File Types” on page 6-32

Steps for Using the Find Files Tool

- 1 Open the Find Files tool by selecting **Edit > Find Files**.
- 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`.

- 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, select **Contains text** under the **More options Search type**.
- 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	<p>For Include only file type(s), select the file type you are looking for.</p> <p>For example, select <code>*.m</code> to limit the search to MATLAB program files.</p>
All types	<p>a For Include only file type(s), select All files (*).</p> <p>b Clear the Skip file type(s) check box, under More options.</p>
Other variations	<p>a For Include only file type(s), select All files (*).</p> <p>b Select the Skip file type(s) check box, under More options.</p> <p>c Select Edit to specify the file types.</p>

See “Skipping File Types” on page 6-32.

- 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 file to open. To select multiple files:
 - Click to the left of an icon and drag up or down to select contiguous items
 - **Shift**+click to select contiguous items
 - **Ctrl**+click to select non-contiguous items
- 2** Right-click and select one of the **Open** options from the context menu.

For details about the **Open** options, see “Opening and Running Files” on page 6-44.

Accessing Previous Results

View the results of a previous search by selecting its tab at the bottom of the results pane. The Find Files tool 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:

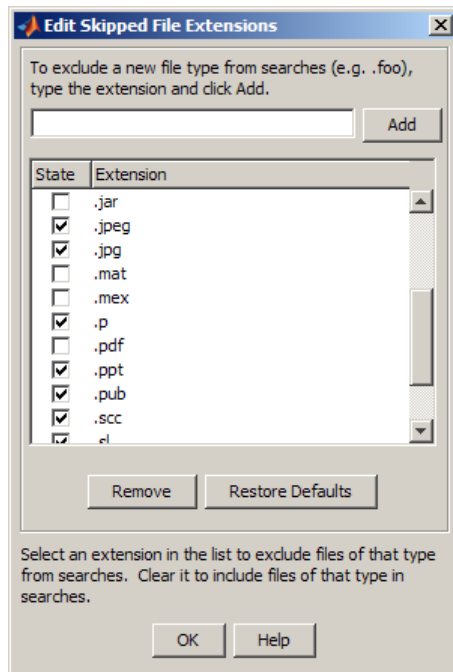
- 1** For **Include only file type(s)**, select **All files (*)**.
- 2** Specify the file types you want the search to ignore:
 - a** Select the **Skip file type(s)** check box.
 - b** Click **Edit**.
- 3** In the resulting Edit Skipped File Extension dialog box, specify which file types to look in and 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 that you want to skip or look for:
 - 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 irrelevant 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.

When you use the Find Files tool, search ignores the selected file types after making the changes.



Locating a File or Folder in the Operating System Browser

To go to a file or folder location in Windows Explorer or Apple Mac Finder, do one of the following:

- In the Current Folder browser, right-click the file or folder, and then select **Show in Explorer** or **Show in Finder**.
- In the Current Folder browser, right-click in white space, and then select **Open Current Folder in Explorer** or **Open Current Folder in Finder**.

- In the Editor, right-click a document tab, and then select **Show in Explorer** or **Show in Finder**.

Document tabs appear in the Editor only when multiple documents are open and docked in the Editor.

The Windows Explorer or Mac Finder opens to the folder containing the selected item.

Finding Files and Folders Using Functions

To...	Use This Function
List files and folders in the current folder or in subfolders on the search path	<code>dir</code>
Determine if a variable, function, or folder exists.	<code>exist</code>
Search for the specified string in the first line of help in a MATLAB program file	<code>lookfor</code>
See files and folders that are relevant to MATLAB	<code>what</code>
See the full path to a file	<code>which</code>

Additional Ways to Find Files

- “Find Functions Using the Function Browser” on page 3-7
- “Search Syntax and Tips” on page 4-5
- “Find Files in File Exchange — Searching and Using Tags” on page 7-12

Creating, Opening, Changing, and Deleting Files and Folders

In this section...

“Creating New Files and Folders” on page 6-35

“Copying, Renaming, and Deleting Files and Folders” on page 6-40

“Opening and Running Files” on page 6-44

Creating New Files and Folders

You can add files and subfolders to your current folder with the Current Folder browser or by typing commands.

- “Creating Files and Folders with the Current Folder Browser” on page 6-35
- “Creating and Updating MAT-Files with the Current Folder Browser” on page 6-36
- “Creating and Managing Zip File Archives” on page 6-37
- “Creating Files and Folders Using Functions” on page 6-39

Creating Files and Folders with the Current Folder Browser

- 1 In the Current Folder browser, navigate to the folder where you want to create a file or folder.

For guidance on where to create files, see “Locations for Storing Your Files” on page 6-7.

- 2 Right-click in white space, and then select one of the following from the context menu:

- **New Folder.**

MATLAB creates and selects a folder named `New Folder`.

- **New File > *file-type*,**

For *file-type* you can choose: **Script**, **Function**, **Class**, **Enumeration**, **Model** (if Simulink is installed), or **Zip File**. Function,

class, and enumeration files that you create this way contain template information representing the fundamental elements for the file (such as function arguments).

MATLAB creates and selects a new file named untitled with the appropriate extension.

3 Replace the selected name by typing a new name.

For file naming conventions, see “Function Name” and “Naming Functions”.

4 Press **Enter**.

Creating and Updating MAT-Files with the Current Folder Browser

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 6-7.
- 2** In the Workspace browser, select a variable to save. Hold down the **Ctrl** key and click any other variable names you want to include in the MAT-file.
- 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 6-44.

Creating and Managing Zip File Archives

To back up files, conserve file storage space, or to forward collections of files to other people, create archives using zip files. You can create, view, and adjust the contents of a zip file from within the Current Folder browser, as described in the sections that follow.

Viewing the Contents of Zip Files. To view the contents of a zip file without extracting any files it contains, click the associated + (expand) button in the Current Folder browser. This feature is helpful when you want to:

- Confirm the contents of a newly created zip file
- View the contents of a zip file before extracting files
- Selectively open certain items from a zip file

By default, files within a zip file appear dimmed to indicate that they are not on the MATLAB path.

Note Archives created outside of MATLAB can be encrypted or password-protected. You cannot add files to, or extract files from, protected archives from within MATLAB.

Creating Zip Archives. You can either create an empty archive, or select files, folders, or both to create an initial archive. In either case, you can add more files later.

1 Do one of the following in the Current Folder browser:

- Create an empty zip file:

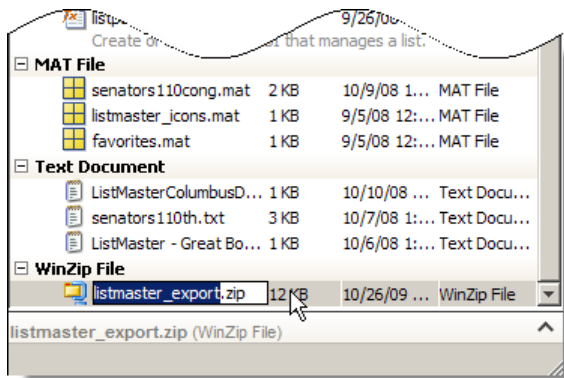
Right-click white space, and then select **New File > Zip File**.

- Create a populated zip file from selected files, folders, or both:

Select the folders and files you want to archive, right-click, and then select **Create Zip File**.

In either case, MATLAB creates an archive with a default name of `Untitledn.zip`, where n is an integer.

- 2 Type over the default file name to specify a descriptive name, for example `listmaster_export.zip`, as shown here.



See also “Adding Files to a Zip Archive” on page 6-39.

Extracting Files from Zip Files. To extract a single file from within a zip file in the Current Folder browser, do one of the following:

- Copy a file name and paste it into a folder in the Current Folder browser.
- Drag the file into a folder in the Current Folder browser.

MATLAB extracts the file and saves it to the folder where you dragged or pasted it.

To extract all the files from a zip file, do one of the following:

- Double-click the zip file in the Current Folder browser.
- Right-click the zip file, and then select **Extract**.

MATLAB extracts the entire contents of the zip file into a folder having the same name as the zip file.

Because MATLAB creates a folder when extracting files, none of the extracted files can overwrite files that have the same name. If you attempt to overwrite a folder with the same name when extracting, MATLAB prompts you to determine what you want to do.

Adding Files to a Zip Archive. To add files and folders to a zip file archive in the Current Folder browser, do one of the following:

- Select, and then drag the file that you want to add onto the archive.
- Copy the file that you want to add to the archive. Then, select the archive to which you want to add the file and paste the file into the archive.

If the archive contains a file or folder with the same name as the one you are adding, a MATLAB dialog box opens. The dialog box asks if you want to replace the existing file in the archive.

Comparing the Contents of a Zip Archive to Unzipped Files and Folders. To determine differences between archived and unarchived files, use the Comparison Tool from within the Current Folder browser as you would for any other files and folders.

For instance:

- Right-click a zip archive, and then from the context menu select **Compare Against** and specify the folder to which you want to compare the contents of the zip archive.
- Expand a zip archive, right-click a file within it, and then from the context menu select **Compare Against**. Specify the file to which you want to compare the archived file.

For details, see “Comparing Files and Folders” on page 6-47.

Creating Files and Folders Using Functions

As an alternative to using the Current Folder browser to create files and folders, you can run functions in the Command Window or from a script.

To...	Use This Function
Create a folder	mkdir
Create a text file, such as a MATLAB program file	edit
Create a MAT-file	save
Create archive of files	zip, gzip, tar
Extract files from archive	unzip, gunzip, untar

See also “Locations for Storing Your Files” on page 6-7.

Copying, Renaming, and Deleting Files and Folders

- “Renaming Files Using the Current Folder Browser” on page 6-40
- “Renaming Files and Folders Using Functions” on page 6-41
- “Deleting Files and Folders Using the Current Folder Browser” on page 6-41
- “Deleting Files and Folders Using Functions” on page 6-42
- “Copying and Moving Files and Folders” on page 6-43
- “Changing Properties of Files and Folders” on page 6-43

Renaming Files 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 Functions”.
 - The folder is on the search path. See “Handling Errors and Unexpected Behavior When Updating Folders” on page 6-81.
- 4** Press **Enter**.

Renaming Files and Folders Using Functions

Use the `movefile` function.

Deleting Files and Folders Using the Current Folder Browser

To remove items:

- 1** Select the item to delete. To select multiple items:
 - Click to the left of an icon and drag up or down to select contiguous items
 - **Shift**+click to select contiguous items
 - **Ctrl**+click to select non-contiguous items
- 2** Right-click and select **Delete** from the context menu.

Note You cannot delete a folder while it is on the search path. See “Handling Errors and Unexpected Behavior When Updating Folders” on page 6-81.

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
Microsoft Windows platforms	<p>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.</p> <p>To delete a selection permanently when the system preference is set to recycle, press Shift+Delete.</p>
Linux platforms	<p>Specify the behavior:</p> <ol style="list-style-type: none"> 1 Select File > Preferences > General. 2 Set the Deleting files option you want.

Platform	Behavior Deleting Files and Folders Using the Current Folder Browser
	<p>To move files to a temporary folder, determine the location by running <code>tempdir</code>.</p> <p>To delete a selection permanently when the preference is set to recycle, press Shift+Delete.</p>
Apple 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	<code>delete</code>
Delete a folder	<code>rmdir</code>

You cannot 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.

Setting the preference to delete files permanently makes `delete` run faster.

To override the preference when using the `delete` function, use the `recycle` function.

The location for deleted files varies by platform, as the following table indicates.

Platform	Location for Files Not Permanently Deleted Using the delete Function
Microsoft 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.
Apple 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, click and drag a file from one folder to another or to another application, such as Windows Explorer.

Note You cannot move a folder that is on the search path using the Current Folder browser. See “Handling Errors and Unexpected Behavior When Updating Folders” on page 6-81

To copy and move files and folders using functions, use `copyfile` and `movefile`.

Changing Properties of Files and Folders

To change some properties of files and folders, such as read/write permissions, use the `fileattrib` function.

Opening and Running Files

- “Opening Files and Importing Data Using the Current Folder Browser” on page 6-44
- “Opening Files and Importing Data Using Functions” on page 6-45
- “Running MATLAB Program Files from the Current Folder Browser” on page 6-45

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** — Opens the file using the appropriate MATLAB tool for the file type. For example, this option loads a MAT-file into the Workspace browser.
 - **Open in GUIDE** — Opens a FIG-file in GUIDE instead of a figure window. For more information, see “Opening GUIDE”.
 - **Open as Text** — Opens the file in the Editor as a text file, even if the file type is associated with another application or tool.

This is useful, for example, if you have imported a tab-delimited data file (.dat) into the workspace and you find you want to add a data point. Open the file as text in the Editor, make your addition, and then save the file.

- **Open Outside MATLAB** — Opens the file using 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 “Managing File Associations for MATLAB on Windows Systems” on page 1-5.

For information on how to view information about a file without opening it, see “Viewing File Details Without Opening Files” on page 6-23.

Opening Files and Importing Data Using Functions

To...	Use This Function
Open a file or open a variable in the Variable Editor	<code>open</code>
Add variables from a MAT-file to the workspace	<code>load</code>
Import data files	<code>importdata</code>
Import data file using the Import Wizard	<code>uiimport</code>
Access the system clipboard	<code>clipboard</code>


See Also.

- “Supported File Formats”

Running MATLAB Program Files from the Current Folder Browser

For convenience, you can run MATLAB scripts and functions from the Current Folder browser. Script files do not accept input arguments or return values and can be run directly. If the program is a function which requires input arguments or returns output arguments, you can define a *run configuration* for it that defines arguments. See “Using Run Configurations for Functions” on page 6-46. Run any program file in the following way:

- 1 In the Current Folder browser, change the current folder to the folder containing the file to run.
- 2 Right-click the file name to open the context menu.
- 3 (Optional) If you have defined a run configuration for the file you want to use, select it from the **Run Configurations** on the context menu. Select **Edit Configurations** to edit or create one.
- 4 From the context menu, select **Run**.

If you have customized the Current Folder Browser toolbar with a **Run** button , you can select the file and then click the **Run** button. That button has a dropdown list of function run configurations you have defined. For details, see “Toolbar Customization” on page 2-11.

Using Run Configurations for Functions. If you run a function that requires input arguments, executing it from the **Run** context menu or toolbar button item might not work properly. Specify default input arguments for functions that require them by defining run configurations for them. To create a run configuration:

- 1 Right-click the name of a function in the Current Folder browser and select **Run Configurations > Edit Configurations**.
- 2 Give your run configuration a name.
- 3 Type in the expressions for running the function in the **MATLAB expression** panel.
- 4 Click **Run** if you want to test the configuration.
- 5 Click **Close** to save the configuration and exit the dialog box.

Executing a function with a run configuration sets up function arguments as the configuration specifies. You can create multiple configurations for a function. Your configurations are saved with your preferences. To use a run configuration:

- 1 Right-click on the function name and select **Run Configurations > *configuration name***.
- 2 The function executes according to the configuration you select and that configuration is the selected one the next time you use this context menu.

For more information about using run configurations, see “Run MATLAB Files in the Editor” on page 8-46.

Comparing Files and Folders

In this section...

“Comparing Files and Folders” on page 6-47

“Comparing Folders and Zip Files” on page 6-49

“Comparing Text Files” on page 6-53

“Comparing Files with Autosave Version or Version on Disk” on page 6-58

“Comparing MAT-Files” on page 6-58

“Comparing Binary Files” on page 6-61

“Using Comparison Tool Features” on page 6-61

“Function Alternative for Comparing Files and Folders” on page 6-63

Comparing Files and Folders

You can use the Comparison Tool to determine and display the differences between selected pairs of files or folders. The comparison process involves three steps:

- 1 “Select the Files or Folders to Compare” on page 6-47
- 2 “Choose a Comparison Type” on page 6-48
- 3 “Explore the Comparison Tool Report” on page 6-49

Select the Files or Folders to Compare

You can compare files and folders using any of these methods:

- From the Current Folder browser:
 - Select a file or folder, right-click and select **Compare Against**.
 - For two files or subfolders in the same folder, select the files or folders, right-click and select **Compare Selected Files/Folders**.
- If you have a file open in the Editor, select **Tools > Compare Against**. Then, select the option you want: **Choose**, **Autosave Version**, or

Compare Against Version on Disk. Use **Choose** to browse to a file, or see “Comparing Files with Autosave Version or Version on Disk” on page 6-58.

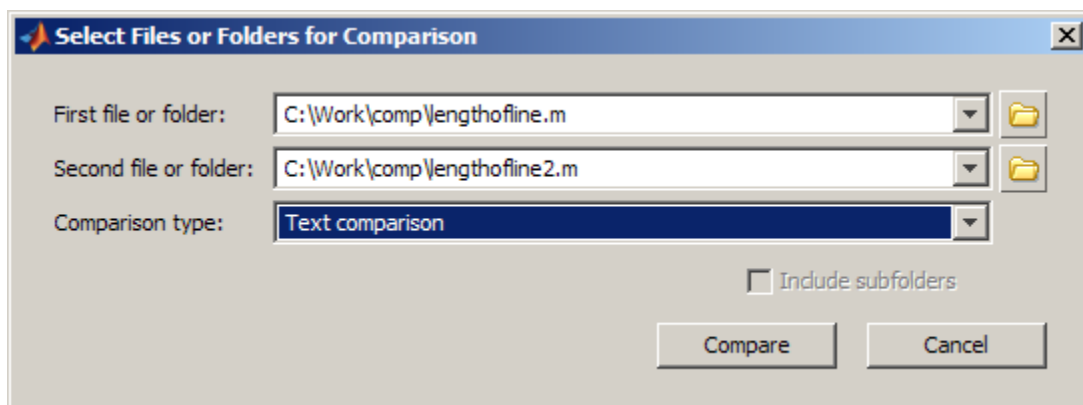
- From the MATLAB desktop, select **Desktop > Comparison Tool**. Then, select the files or folders to compare.
- From the command line, use the `visdiff` function.

If you use **Compare Against**, select the second item to compare in the Select Files or Folders for Comparison dialog box. Optionally, change the comparison type if multiple types are available for your selections.

Choose a Comparison Type

If you specify two files or folders to compare using either the Current Folder browser or the `visdiff` function, then the Comparison Tool automatically performs the default comparison type.

If there are multiple comparison types available for your selections, you can change what type of comparison to run. For example, text, binary, file list, or XML comparison. To change the comparison type, create a new comparison using the Comparison Tool. You can change comparison type in the Select Files or Folders for Comparison dialog box.



For example, from the Current Folder browser, if you select two MAT-files to compare, you get the default comparison type showing information about the

variables. To change the comparison type to binary, create a new comparison using the Comparison Tool. See “Selecting Files or Folders to Compare from the Comparison Tool” on page 6-61.

Explore the Comparison Tool Report

Comparison Tool report features depend on your comparison type. You can use the tool to:

- Compare lines in two text files (some other applications refer to this as a *file diff* operation). See “Comparing Text Files” on page 6-53.
- Compare and merge variables in two MAT-files. See “Comparing MAT-Files” on page 6-58.
- Determine whether the contents of two binary files match. See “Comparing Binary Files” on page 6-61.
- Compare any combination of folders, zip files, or Simulink manifests to determine:
 - Which file and folder names are unique to each list
 - If files and folders with the same name in each list have the same content See “Comparing Folders and Zip Files” on page 6-49.
- Compare XML files:
 - If you select XML files to compare and you have MATLAB Report Generator™ software, the Comparison Tool runs a hierarchical matching algorithm. You then see a report showing a hierarchical view of the portions of the two XML files that differ.
 - If you have Simulink Report Generator software, you can select a pair of Simulink models (.mdl files) to compare XML files generated from them.

Comparing Folders and Zip Files

- “Folder Comparison Report” on page 6-50
- “Highlighting of Differences” on page 6-51
- “Next Steps Using the Report” on page 6-52


Folder Comparison Report

To select items to compare, see “Select the Files or Folders to Compare” on page 6-47. You can perform *file list comparisons* for any combinations of folders, zip files, and Simulink Manifests.

When you use the Comparison Tool to compare two folders (sometimes referred to as *directories*) or any file list comparison (for example, folder versus zip file), a window opens and presents the contents side by side. The tool enables you to:

- Determine the files that the comparison lists have in common.
- Determine if files with identical names that are common to both comparison lists also have identical content.
- Open a new comparison of two files or folders that are common to both comparison lists, but have different content.
- Open a file for viewing in the Editor.
- Specify filters to ignore certain files or folders

For list comparisons, if you want to expand the list to see all files in subfolders in one report, select the **Include subfolders** check box when selecting items to compare. If you do not include subfolders, you can click **compare** links in the report to open a new comparison of two folders with changed content.

To define filters to exclude unimportant differences, click the Filters toolbar button, , or select **Comparison > Edit File and Folder Filters**.

The File and Folder Filters dialog box opens. Specify filters to ignore certain files and folders, such as backup files or files created by a revision control system. Filters can save time when reviewing differences, especially when comparing many subfolders. Double-click to edit existing filters.

For example, to ignore all files and folders in a folder named CVS, open the File and Folder Filter dialog box and enter:

```
CVS/
```

To ignore all files in a folder named CVS, but not ignore subfolders, enter:

CVS/*

Highlighting of Differences

The Comparison Tool displays the contents of the lists side by side and highlights files and subfolders that do not match. The following table describes how the tool highlights each type of change. The status message (such as **identical** or **contents changed**) appears in the **Change Summary** column.

Change Summary	Highlighting for Folders	Highlighting for Files	Notes
Contents changed	Dark pink	Pink	The contents of the files or folders differ. Click the compare link to investigate.
Added	Dark green	Green	File or folder only exists in the right list.
Removed	Dark purple	Purple	File or folder only exists in the left list.
Identical	None	None	

The following image shows an example of the Comparison Tool when two folders are compared. The results are sorted by **Type**.

Comparing folder curvefitting vs. folder curvefitting2

Left file list	Contents of folder C:\Work\curvefitting
Right file list	Contents of folder C:\Work\curvefitting2

Click on a column header to sort the table

Type	File Name	In left list (folder curvefitting)		In right list (folder curvefitting2)		Change Summary
		Size (bytes)	Last Modified Date	Size (bytes)	Last Modified Date	
folder	cftoolgui/	-	2011-08-17 12:05:58	-	2011-08-17 12:05:53	contents changed (compare)
MATLAB File	csapidem.m (open)	<i>(not in this list)</i>		15038	2009-01-08 13:56:28	added
folder	curvefit/	-	2011-08-17 12:06:00	-	2011-08-17 12:05:57	contents changed (compare)
folder	demosearch/	-	2011-08-17 12:06:00	-	2011-08-17 12:05:57	identical
folder	html/	-	-	<i>(not in this list)</i>		removed
Text Document	kdm.txt (open)	<i>(not in this list)</i>		1367	2008-10-03 11:13:37	added
Editor Autosave	lengthofline.asv (open)	<i>(not in this list)</i>		2403	2009-11-16 15:24:45	added
MATLAB File	lengthofline.m (open: left right)	2405	2009-11-12 16:56:52	2408	2009-11-16 15:25:14	contents changed (compare)
Firefox Document	manifest_report.html (open)	2410	2008-11-27 14:24:50	<i>(not in this list)</i>		removed
XML Document	report.xml (open)	<i>(not in this list)</i>		4868	2008-09-11 17:53:32	added
folder	sftoolgui/	<i>(not in this list)</i>		-	-	added
MAT-file	splinetool.mat (open)	1720	2001-08-20 18:14:12	<i>(not in this list)</i>		removed

Next Steps Using the Report

To explore the report you can use the following tools:

- You can sort the results by name, type, size, or last modified timestamp by clicking the column headers. For example, click the **Type** column header to sort by folder and file type, as shown in the preceding figure.
- To open a new comparison of two files or folders with changed contents, click the **compare** link next to file or folder names highlighted in pink.
- To open a file in the Editor, click the **open** link next to a file name.

If the file is present in both folders, you can click links to open the **left** or **right** file.

- If subfolders are very large and contain many files, analysis continues in the background. The tool displays the number of items still to be compared at the top of the report, as shown in the next figure. You can click the links to **Skip Current** item or **Cancel All** to stop further analysis.



Items still to be compared: 12 [Skip Current](#) [Cancel All](#)

- For details on other comparison tool features, see “Using Comparison Tool Features” on page 6-61.

Comparing Text Files

- “Selecting Text Files to Compare” on page 6-53
- “Highlighting of Differences” on page 6-53
- “Stepping Through Differences” on page 6-56
- “Viewing a Summary of Differences” on page 6-56
- “Ignoring Whitespace Differences in Text Comparisons” on page 6-57
- “Showing Differences Only” on page 6-57
- “Changing the Display Width of a Text Comparison” on page 6-57
- “Saving HTML Report” on page 6-57

Selecting Text Files to Compare

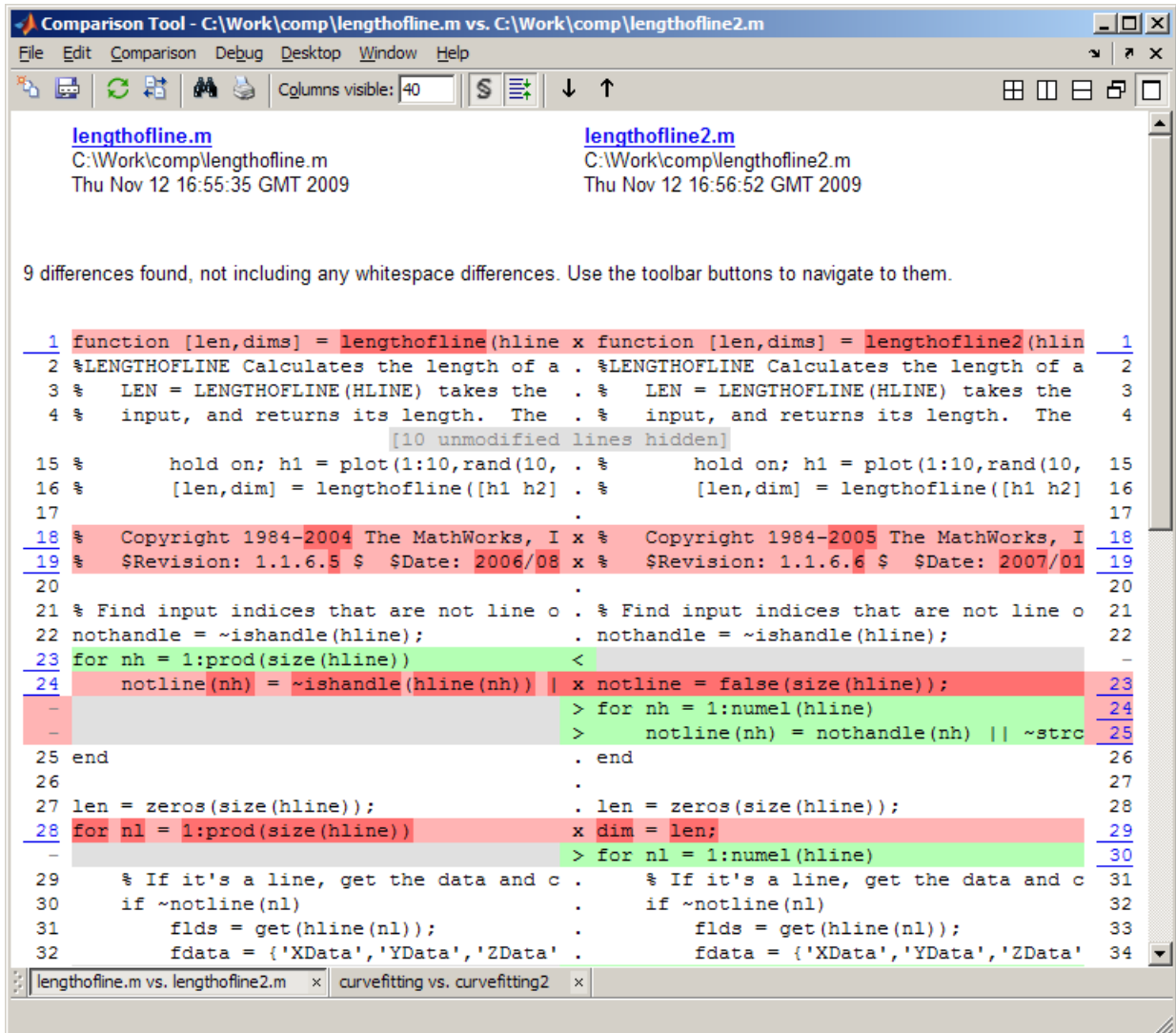
To select files to compare, see “Select the Files or Folders to Compare” on page 6-47.

Highlighting of Differences

When you use the Comparison Tool to compare two text files, a window opens and presents the two files side by side. Symbols indicate how you can adjust the files to make them match. This feature can be useful when you want to compare the latest version of a text file to an autosave version.

The Comparison Tool report displays the files side by side and highlights lines that do not match, as follows:

- Dark pink highlighting indicates changed characters within lines.
- Pink highlighting and an **x** between the two files indicate that the content of the lines differs between the two files.
- Green highlighting and a right (>) or left (<) angle bracket between the two files indicate a line that exists on one side only.



The Comparison Tool attempts to match lines and detects local text that is added, deleted, or changed. It does not do a simple line-by-line comparison. In the previous image, for example, the tool determines that `lengthofline.m` has a line of code that does not exist in `lengthofline2.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.

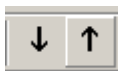
If the files you are comparing are extremely long, the tool could run out of memory while attempting to perform the file comparison. In which case, the message,

```
Maximum file length exceeded.  
Defaulting to line-by-line comparison.
```

appears. In a line-by-line comparison, the tool highlights the lines containing the end statement because in performing this operation, it finds that the last line in one file does not match the last line in the other file.

Stepping Through Differences

Because text files can be lengthy, the Comparison Tool provides toolbar buttons to help you step through the results from one difference to the next.



To navigate through comparison results:

- Click the down arrow toolbar button to go to the next set of lines that differ.
If no additional sets of lines differ, the down arrow takes you to the end of the file.
- Click the up arrow toolbar button to go to a previous set of lines that differ.
If no previous set of lines differ, the up arrow takes you to the beginning of the file.

Alternatively, use the menu items **Comparison > Next** or **Previous**.

Viewing a Summary of Differences


To see a summary of differences between two text files, scroll to the bottom of the Comparison Tool and view the list, which contains information such as:

- Number of matching lines: 52
- Number of unmatched lines in left-hand file: 12
- Number of unmatched lines in right-hand file: 15

Ignoring Whitespace Differences in Text Comparisons

You may want to hide whitespace differences to help you distinguish between functional changes and changes to indentation.




Use the toolbar button, , to toggle the display of differences only involving whitespace characters, or select **Comparison > Ignore Whitespace**.

Showing Differences Only

You can specify whether to show only differences or entire files. It can be useful to hide unmodified lines in large text comparison reports. When you are showing differences only and sections are hidden, the report displays messages like the following: 10 unmodified lines hidden.



Use the toolbar button, , to toggle the display of sections of the report that do not contain any differences, or select **Comparison > Show Differences Only**.

Changing the Display Width of a Text Comparison

To increase or decrease the line lengths of the text files in the comparison display, edit the number in the **Columns visible** field. Resize the window, if necessary.

For details on other comparison tool features, see “Using Comparison Tool Features” on page 6-61.

Saving HTML Report

Click Save As on the toolbar to save a copy of the comparison report as an HTML file. The tool creates a corresponding folder containing the style sheet and JavaScript files that control the report highlighting. To preserve the highlighting, keep the folder with the HTML file.

Note Report links for opening files in MATLAB only work in the MATLAB Web browser.

Comparing Files with Autosave Version or Version on Disk

From the Editor you can compare one open text file with another, or you can choose to compare the latest version of a file in the Editor to an autosave version or a saved version. For an example, 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 > Choose** (or **Save and Compare Against** if your file is modified).

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:

- To compare the open file to the Editor's automatic copy (*filename.asv*) select **Tools > Compare Against > Autosave Version**. For more information, see “Autosaving Files” on page 8-7.
- To compare an open file that has been changed, but not saved, to the saved version, select **Tools > Compare Against Version on Disk**


Comparing MAT-Files

Note To select files to compare, see “Select the Files or Folders to Compare” on page 6-47.

You can use the Comparison Tool to compare two MAT-files. The tool presents the variables in the two files side by side, which enables you to:

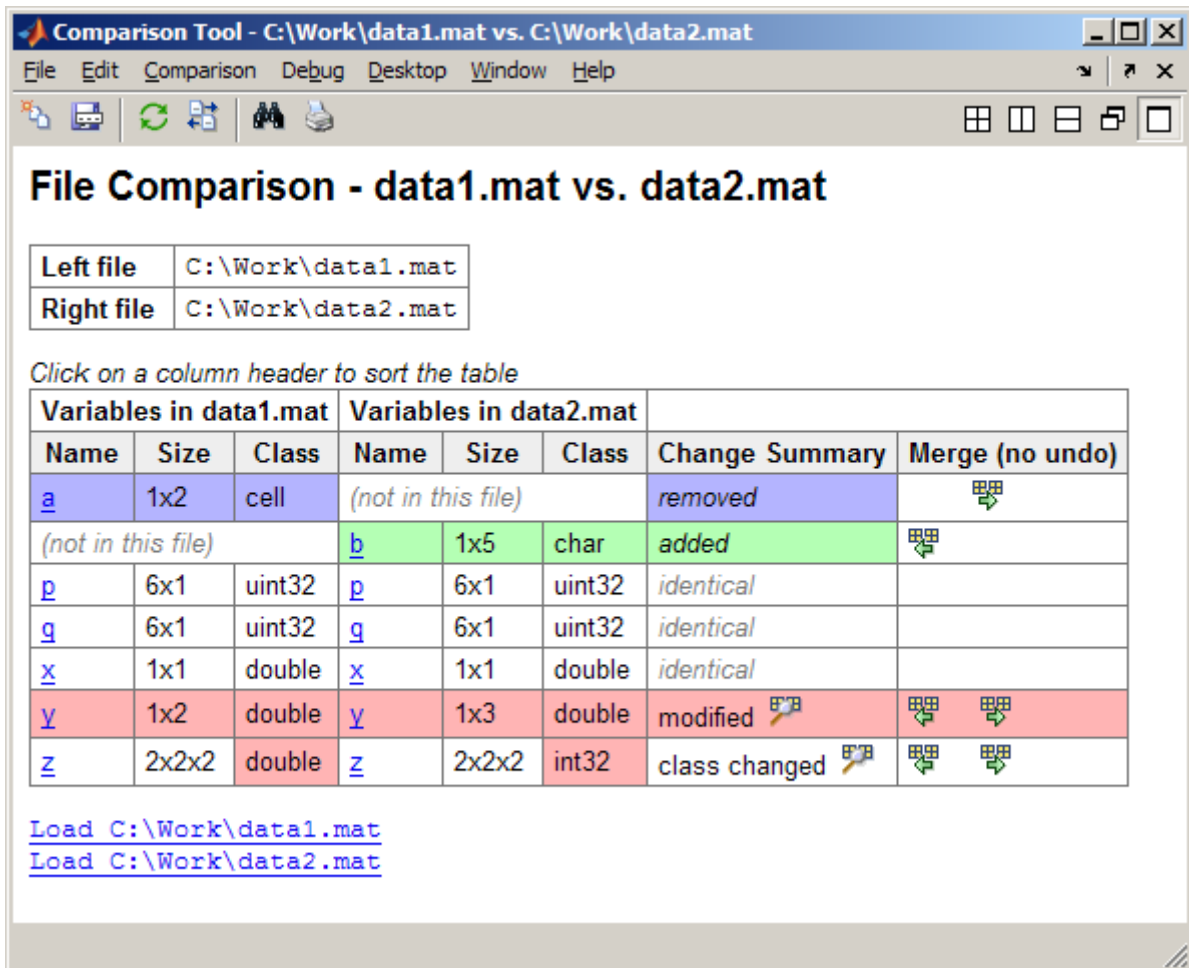
- View and sort by the name, size, class, and change summary of all variables.
- View details of differences between variables, to see which fields of a structure are different, and view differences in individual elements of an array.
- Merge changes between files by copying modified variables from one file to the other (*Caution*: No undo).
- See which variables are common to each file and which are unique.
- Load the contents of the variables into the Variable Editor by clicking the name of that variable.
- Load the MAT-files into the workspace by clicking a **Load** link.
- Save a copy of the report as an HTML file. Click Save As on the toolbar.

The Comparison Tool report highlights changes in variables as follows.

Change Summary	Highlighting	Notes
Modified	Pink	Values of the variable differ between the two files. Click the View differences button  to investigate. A new variable comparison report opens to display differences in individual array elements or differing fields of a structure. Double-click pink rows or cells to investigate further layers of differences.
Added	Green	Variable only exists in right file.
Removed	Purple	Variable only exists in right file.
Identical	None	Variable identical in both files.
Class changed	Pink (only in Class columns)	Variable data class changed. Click the View differences button to investigate.

Click the Merge button  in the Merge column to copy modified variables from one file to the other.

The following image shows the results when you compare two files, data1.mat and data2.mat.



Comparing Binary Files

Note To select files to compare, see “Select the Files or Folders to Compare” on page 6-47.

You can use the Comparison Tool to compare two binary files such as DLL files or MEX-files. Also, you can select the **Binary** comparison type for any pair of files with a choice of comparison types.

- 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**.

Using Comparison Tool Features

You can use the Comparison Tool for the following tasks:


- “Selecting Files or Folders to Compare from the Comparison Tool” on page 6-61
- “Exchanging the Left and Right Sides of the Report” on page 6-62
- “Refreshing the Report to Show Updated Files” on page 6-62
- “Finding Text” on page 6-62
- “Viewing New Comparisons” on page 6-62
- “Viewing Previous Comparisons” on page 6-63

Selecting Files or Folders to Compare from the Comparison Tool


To compare two files or folders from the Comparison Tool, follow these steps:

- 1** Select **Desktop > Comparison Tool** to open the tool.

The Comparison Tool opens a window that is empty, except for the title bar, a menu bar, and a toolbar.

- 2** Compare files or folders by clicking the New comparison button  or by selecting **File > New Comparison**.


The dialog box Select Files or Folders for Comparison appears.

- 3 In the dialog box, select two files or folders to compare. Use the drop-down lists or the **Browse** buttons  to locate and select the items that you want to compare.


You also can drag and drop a file or folder from Windows Explorer to the left and right file and folder fields.

- 4 Optionally, choose the comparison type you want to use. Either use the default **Comparison type** value, or if multiple comparison types are available, select a different one from the list. For example, for text files you could select text or binary comparison types.
- 5 Click **Compare**.


Exchanging the Left and Right Sides of the Report

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 .


Refreshing the Report to Show Updated Files

After making changes to and saving the files in the Editor, update the results in the Comparison 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 “Find Text Currently Displayed in the Command Window” on page 3-53.

Viewing New Comparisons

You can perform another file comparison by selecting **File > New Comparison**, or by clicking the toolbar button .

New comparisons open additional tabs in the Comparison Tool.

Viewing Previous Comparisons

You can see the results of previous comparisons in the current session by selecting that comparison's tab entry on the document bar at the bottom of the window. If you close the Comparison Tool, the current and previous comparisons are lost.

Function Alternative for Comparing Files and Folders

Use the `visdiff` function to open the Comparison Tool from the Command Window.

```
visdiff(fileorfoldername1, fileorfoldername2)
```

For example, type:

```
visdiff('lengthofline.m', 'lengthofline2.m')
```

Making Files and Folders Accessible to MATLAB

In this section...

“Files and Folders That MATLAB Can Access” on page 6-64

“How to Make Files Accessible” on page 6-64

“Determining if MATLAB Can Access a File” on page 6-66

“Ensuring MATLAB Uses the File You Want” on page 6-67

Files and Folders That MATLAB Can Access

For performance reasons, MATLAB limits where it looks for files. To run or get help for a MATLAB program, or to load a MAT-file, the file must be in one of these locations:

- The current folder in MATLAB
- A folder that is on the search path. See “What Is on the Search Path?” on page 6-70

Make the following accessible to MATLAB:

- Folders containing files that you and others create.
- Folders containing files that are *called by* files you run.
- Subfolders containing files that you run. Making a folder accessible does not make its subfolders accessible.

For files in @ (class) and + (package) folders, make the parent folder accessible. For details, see “Organizing Classes in Folders”.

How to Make Files Accessible

For files that you and others create, see “Basic Options for Making Files Accessible” on page 6-65.

To understand the differences in the basic options, and for other approaches, see “All Options for Making Files Accessible” on page 6-65.

Basic Options for Making Files Accessible

- Store the files that you and others create in the MATLAB folder, which is on the search path. See “Locations for Storing Your Files” on page 6-7.
- 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 <i>seldom</i> run the file.	Change the current folder to the folder that contains the file. See “Viewing and Changing the Current Folder” on page 6-4.
The file is a <i>script</i> (takes no input or output arguments).	Use the run function.
The files are in <i>one</i> folder.	Put the files in the <i>userpath</i> folder. See “Locations for Storing Your Files” on page 6-7.
The files are in <i>multiple</i> folders.	Add the folders to the search path. See “Adding Folders to the Search Path” on page 6-73. If you regularly use the files, save the changes. See “Saving Changes to the Search Path” on page 6-77.
The files call other files that are in multiple folders.	1 Determine the location of all the called files. See “Displaying Dependencies Among MATLAB Code Files” on page 9-14. 2 Add the folders to the search path. See “Adding Folders to the Search Path” on page 6-73.

Usage	Recommendation
Some files in multiple folders have the same name.	See “Detecting and Addressing Name Conflicts” on page 6-68.
You use files in different versions of MATLAB or on different platforms.	Modify the search path in a <code>startup.m</code> file. See “Using the Search Path with Different MATLAB Installations” on page 6-78.

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: <ul style="list-style-type: none"> • Dialog box • Message: Undefined function or method 'fileName' • Message: Cannot find function 'fileName'
View the file in the Current Folder browser.	Either or both of the following are true: <ul style="list-style-type: none"> • File is in the current folder. • File does not appear dimmed in Current Folder browser, assuming the Indicate inaccessible files option is selected in “Preferences for the Current Folder Browser” on page 6-16. 	Either or both of the following are true: <ul style="list-style-type: none"> • File is in a subfolder of the current folder, and the subfolder is not on the search path. • File appears dimmed in the Current Folder browser, assuming the Indicate inaccessible

Option	When MATLAB Can Access the File	When MATLAB Cannot Access the File
		files option is selected in “Preferences for the Current Folder Browser” on page 6-16.
Select File > Set Path .	Set Path dialog box list includes the file location.	The list in the Set Path dialog box does not include the file location.
Run <code>dir</code> with no arguments.	The result includes the file, indicating the file is in current folder.	The result does not include the file.
Run <code>path</code> .	The result includes the file location, indicating the file is in a folder on search path.	The result does not include the file location.
Run <code>which filename</code> .	The result is the full path to the file.	The 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 in a folder on the search path.
- MATLAB uses the file whose folder is closest to the top of the search path instead of a file further down.

The file that MATLAB does *not* use is called a *shadowed* file. In some cases, MATLAB warns you that a shadowed file exists.

Other name conflicts include the following:

- 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.

When there are name conflicts, MATLAB follows these precedence rules:

- “Precedence Rules” and “Function Precedence Order”
- “Class Precedence and MATLAB Path”

Detecting and Addressing Name Conflicts

MATLAB might not be accessing the file that 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 that you want.
- Maintain a single version of a file instead of multiple versions.

Name conflicts can arise from using files that you create. Conflicts also can arise from using:

- Files that others 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 1-23

Using the MATLAB Search Path

In this section...

“What Is the Search Path?” on page 6-70

“Viewing Files and Folders on the Search Path” on page 6-72

“Changing the Search Path” on page 6-73

“Using the Search Path with Different MATLAB Installations” on page 6-78

“Recovering from Problems with the Search Path” on page 6-79

“Handling Errors and Unexpected Behavior When Updating Folders” on page 6-81

What Is the Search Path?

The search path, or *path* is a subset of all the folders in the file system. MATLAB software uses the search path to locate files used with MathWorks products efficiently. 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. These folders are under *matlabroot/toolbox*, where *matlabroot* is the folder displayed when you type `matlabroot` in the Command Window.
- By default, the MATLAB *userpath*. See “Locations for Storing Your Files” on page 6-7.
- Folders you explicitly add to the search path for the files you and others 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 That MATLAB Can Access” on page 6-64.

Order of Folders on the Search Path

The *order* of folders on the search path is important when two files with the same name are in folders on the search path. MATLAB uses the file nearest to the top of the search path. To customize the order of files on the search path, see “Ensuring MATLAB Uses the File You Want” on page 6-67.

Relationship Between the Search Path and the System Path

The search path is *not* the same as the system path. Furthermore, there is no explicit relationship between the MATLAB search path and the system path. However, both paths help in locating files, as follows:

- MATLAB uses the search path to locate MATLAB files efficiently.
- The operating system uses a system path to locate operating system files efficiently.

Therefore, you can issue MATLAB commands that result in the use of both the MATLAB search path and the system path. For example, if you type `dos('tasklist &')` in the MATLAB Command Window, then:

- 1** MATLAB uses the search path to locate and run `dos.m`.
- 2** The `dos` function passes `'tasklist &'` to the Microsoft Windows operating system.
- 3** Microsoft Windows uses the system path to locate and run `tasklist.exe`.

Similar behavior results when you use the MATLAB `unix` and `system` functions or the shell escape (`!`). For details on using the shell escape with MATLAB, see “Run External Commands, Scripts, and Programs” on page 3-15.

How MATLAB Stores the Search Path

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`.

When you change the search path, MATLAB uses it in the current session. To use it in future sessions, save the changes as described in “Saving Changes to the Search Path” on page 6-77.

Viewing Files and Folders on the Search Path

MATLAB provides various ways for you to view the search path, as described in these topics:

- “Using the Current Folder Browser” on page 6-72
- “Using the Set Path Dialog Box” on page 6-72
- Using MATLAB `path` and `pathtool` functions

Using the Current Folder Browser

To determine if files or folders in the Current Folder browser are on the search path:

- 1 In the Current Folder browser, right-click any file or folder, and ensure there is a check mark next to **Indicate Files Not on Path**.

If there is no check mark, select **Indicate Files Not on Path**. A check mark appears.

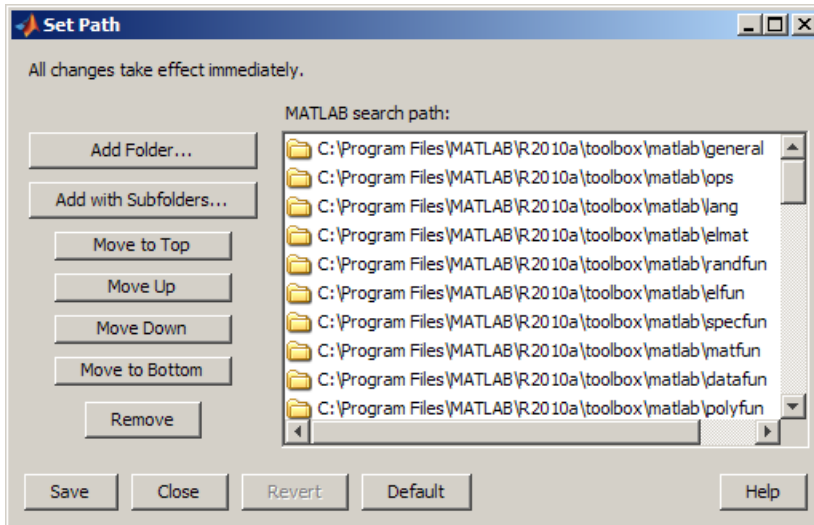
- 2 Hover the pointer over any dimmed file or folder in the Current Folder browser to find out why it is dimmed.

A tooltip opens with an explanation. Frequently, the tooltip indicates that the file or folder is not on the MATLAB path.

Using the Set Path Dialog Box

To view the entire MATLAB search path, select **File > Set Path**.

The Set Path dialog box opens, listing all folders on the search path.



Changing the Search Path

MATLAB provides various ways for you to change the search path, as described in the following sections:

- “Adding Folders to the Search Path” on page 6-73
- “Removing Folders from the Search Path” on page 6-75
- “Changing the Order of Folders on the Search Path” on page 6-76
- “Saving Changes to the Search Path” on page 6-77
- “Specifying Startup Options in the MATLAB Startup File” on page 1-19
- Using MATLAB Search Path functions

Adding Folders to the Search Path

You can add folders to the search path for just the current session, or for both the current and future sessions.

Current Session Only. To add folders to the top of the search path for the duration of the current session use the following method:

- 1 From the Current Folder browser, select, and then right-click the folder or folders to add.
- 2 From the context menu, select **Add to Path**, and then select an option:
 - **Selected Folders**
 - **Selected Folders and Subfolders**

To add the folder that contains an Editor document to the top of the search path:

- 1 In the Editor, right-click the document tab.

Document tabs appear in the Editor only when multiple documents are open and docked in the Editor.

- 2 Select **Add *folder-name* to Search Path**.

To change the ordering in the search path, follow the instructions in “Changing the Order of Folders on the Search Path” on page 6-76.

Current and Future Sessions. To add folders to the search path for the current session and future sessions, use the `addpath` function, or follow these steps:

- 1 From the Current Folder browser, select **File > Set Path**.

The Set Path dialog box appear.

- 2 Click one of these buttons:

- **Add Folder**
- **Add with Subfolders**

- 3 In the Browse For Folder dialog box, select the folder to add to the search path, and then click **OK**.

MATLAB adds the specified folder to the top of the search path.

If you do not want the folder at the top of the search path, see “Changing the Order of Folders on the Search Path” on page 6-76.

4 Keep or cancel the search path 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**, and then click **Close**.

For details on where to save the file, see “Saving Changes to the Search Path” on page 6-77.

- To undo your changes, click **Revert**, and then click **Close**.
- To restore the default search path, click **Default**, and then click **Close**. See “Restoring the Default Search Path” on page 6-78.

Removing Folders from the Search Path

You can remove folders from the search path for just the current session, or for both the current and future sessions.

For the Current Session Only. To remove one or more folders from the search path:

- 1** Select and right-click the folder or folders to remove.
- 2** From the context menu, select **Remove from Path**, and then select an option:
 - **Selected Folders**
 - **Selected Folders and Subfolders**

To remove the folder that contains an Editor document from the search path:

- 1** In the Editor, right-click the document tab.

Document tabs appear in the Editor only when multiple documents are open and docked in the Editor.

- 2** Select **Remove *folder-name* from Search Path**.

For Current and Future Sessions. To remove folders from the search path for the current and future sessions, use the `rmpath` function, or follow these steps:

- 1 Open the Set Path dialog box by selecting **File > Set Path**.
- 2 Select the folders 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**.
For details on where to save the file, see “Saving Changes to the Search Path” on page 6-77.
 - To restore the default search path, click **Default**. See “Restoring the Default Search Path” on page 6-78.
- 5 Click **Close**.

Changing the Order of Folders on the Search Path

Change the order of folders in the search path when files with the same name appear in multiple folders on the search path. When you specify such a file, MATLAB uses the one found in the folder nearest to the top of the search path.

Moving Folders to Various Positions on the Search Path. To change the order of folders on the search path:

- 1 Open the Set Path dialog box by selecting **File > Set Path**.
- 2 Select the folders to move on the search path.
- 3 Click one of the Move buttons, such as **Move to Top**. The order of the folders changes.
- 4 To use the 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 MATLAB session.

5 Click Close.

Note The MATLAB (*userpath*) folder automatically moves to the top of the search path the next time you start MATLAB. See “Locations for Storing Your Files” on page 6-7.

Moving a Folder to the Top or Bottom of the Search Path. To move a folder to the top or bottom of the search path, use the path function.

Saving Changes to the Search Path

Changes you make to the search path always remain in effect during the current MATLAB session. For MATLAB to use the changed search path in future sessions, save the search path, which updates the `pathdef.m` file.

Note The MATLAB (*userpath*) folder automatically moves to the top of the search path the next time you start MATLAB. See “Locations for Storing Your Files” on page 6-7.

Ways to Save Changes. To save changes to the search path, do one of the following:

- Click **Save** in the Set Path dialog box. See “Using the Set Path Dialog Box” on page 6-72.
- Use the `savepath` function.

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 MathWorks.

To restore the default search path, do one of the following:

- Click **Default** in the Set Path dialog box. See “Using the Set Path Dialog Box” on page 6-72. This method also adds the *userpath* folder. See “Locations for Storing Your Files” on page 6-7.
- Use the `restoredefaultpath` function.

See also “Recovering from Problems with the Search Path” on page 6-79.

Using the Search Path with Different MATLAB Installations

Using the Search Path with Different Versions

The default search path changes for each MATLAB version because the default folders that come with the products change. Different MATLAB versions cannot use the same `pathdef.m` file.

To use your files with a new MATLAB version 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. For more information, see “Specifying Startup Options in the MATLAB Startup File” 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. Proceed from one step to the next 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 another location, make corrections. For example, delete the incorrect `pathdef.m` file and ensure the correct `pathdef.m` file is in a location that MATLAB can access. See “Where to Save the Search Path File” on page 6-77.
- 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 Restart MATLAB 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 6-72. Depending on the problem, you might not be able to open the dialog box.
 - b** Restart MATLAB 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** Restart MATLAB 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` file in the startup folder, it caused the problem. Either remove the bad `pathdef.m` file 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 1-12.

- c** Restart MATLAB 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.

Handling Errors and Unexpected Behavior When Updating Folders

You can 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, then:

- 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 6-47
- Chapter 9, “Tuning and Managing MATLAB Code Files”
- Chapter 12, “Source Control Interface”
- Chapter 7, “File Exchange — Finding and Getting Files Created by Other Users”

File Exchange — Finding and Getting Files Created by Other Users

- “Download Files from File Exchange” on page 7-2
- “Find Files in File Exchange — Searching and Using Tags” on page 7-12
- “View and Sort the List of Files in File Exchange” on page 7-27
- “View Details About a File” on page 7-29
- “Get Files from the File Exchange Repository” on page 7-31
- “Best Practices for Using Files Provided by Other Users” on page 7-36
- “Contribute to the File Exchange Repository” on page 7-38
- “Frequently Asked Questions About File Exchange” on page 7-41

Download Files from File Exchange

In this section...
“About File Exchange” on page 7-2
“Steps for Using File Exchange” on page 7-4
“Example — Finding and Downloading a File in File Exchange” on page 7-5

About File Exchange

- “What Is File Exchange?” on page 7-2
- “What You Need to Use File Exchange” on page 7-2
- “Ways to Access the File Exchange Repository” on page 7-3

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:

- MATLAB program 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 “Specify Proxy Server Settings for Connecting to the Internet” on page 2-73.

- 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 “Download Files from File Exchange” on page 7-2.
- 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.

Steps for Using File Exchange

- 1** Open the tool by selecting **Desktop > File Exchange**.
- 2** In the login window, provide the email 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 “Find Files in File Exchange — Searching and Using Tags” on page 7-12.

- 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 7-13.

- 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 “View and Sort the List of Files in File Exchange” on page 7-27.

- 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 “View Details About a File” on page 7-29.

- 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 “Get Files from the File Exchange Repository” on page 7-31


- 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 7-36.

- 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 “Contribute to the File Exchange Repository” on page 7-38.

- 10 When you finish using the tool, close it or log out.

If you have questions while you work, see “Frequently Asked Questions About File Exchange” on page 7-41 by clicking the Help button .

Example — Finding and Downloading a File in File Exchange

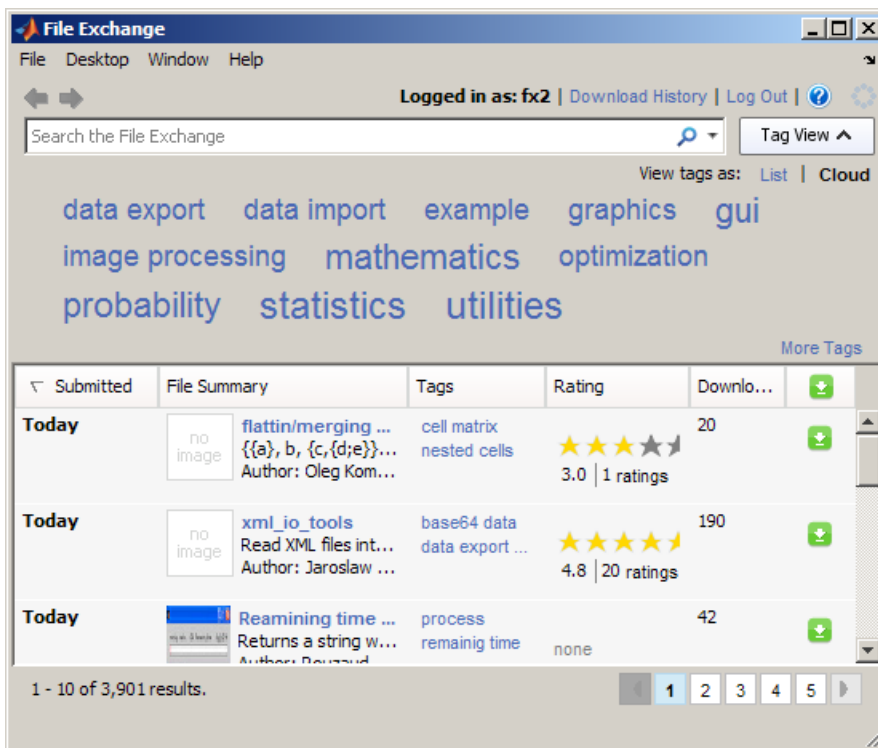
This example looks for files to help you analyze voice signals.

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 7-12.

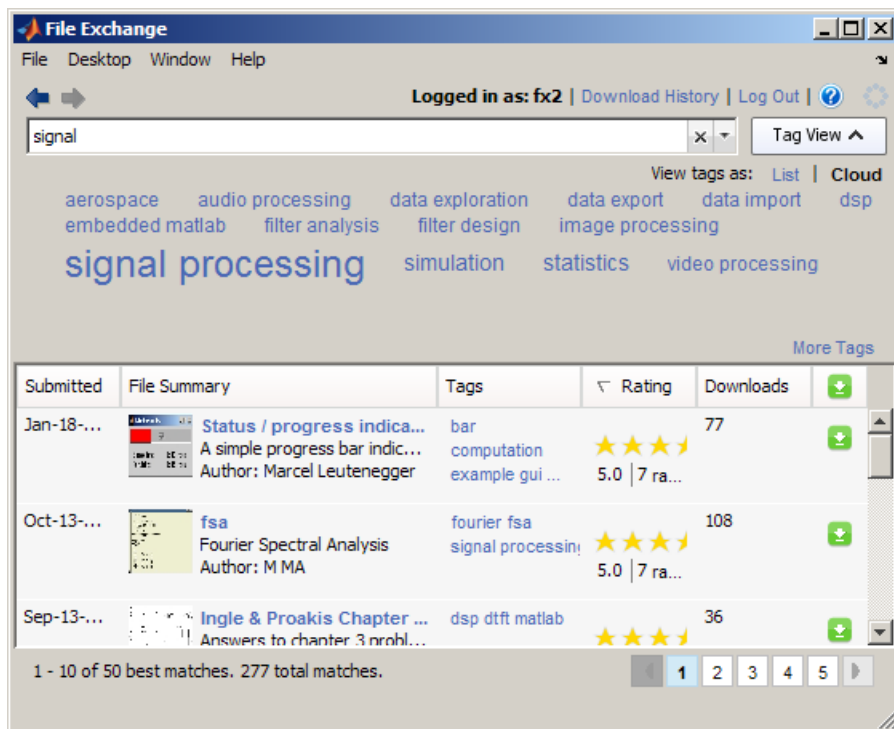


- 3 Because there are no obvious results or tags of interest, change the criteria. Change the sort order to show results that have the highest ratings first by clicking the **Rating** column header.

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 7-28.

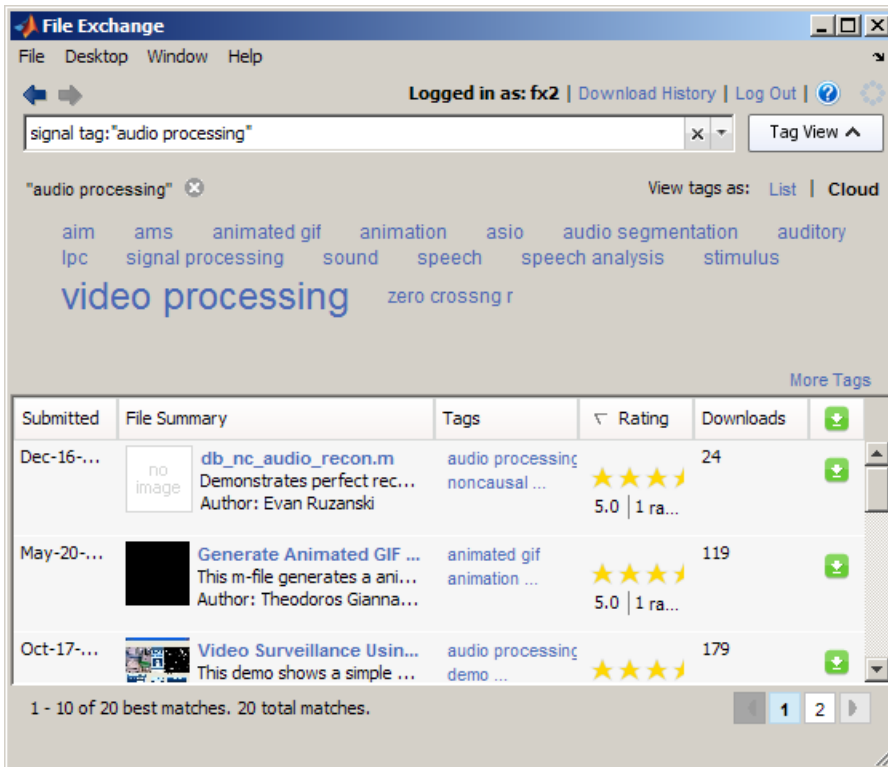


- 4 Narrow the results by clicking the audio processing tag.

File Exchange:

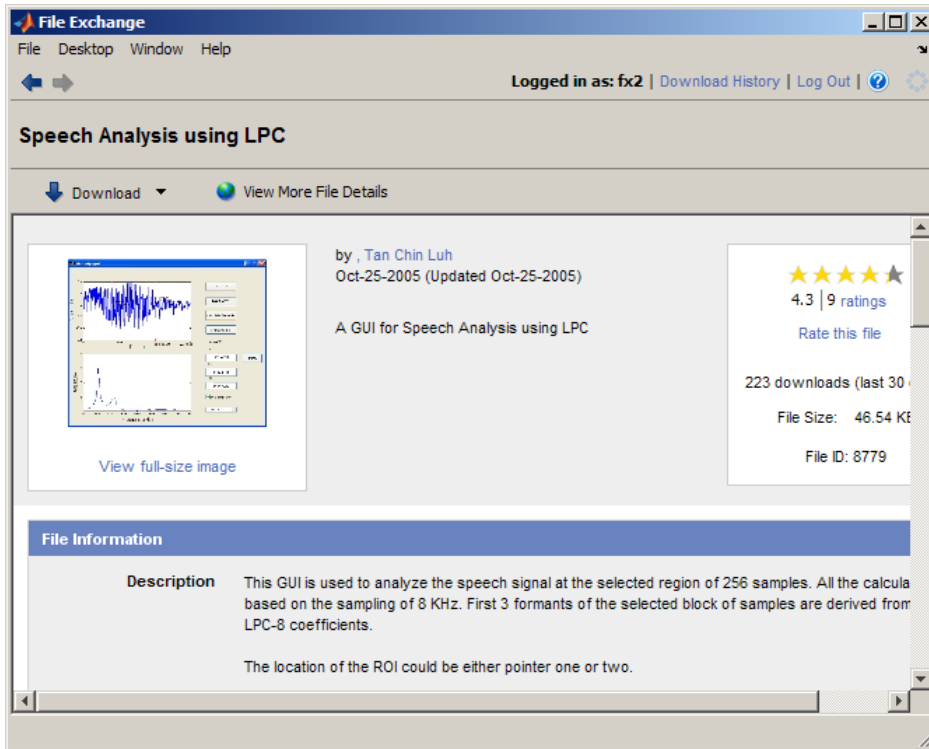
- Reports that 20 files are associated with the audio processing tag or variations of it, and have signal in the title, tag, or description.
- Shows the popular tags associated with the 20 files listed.

For more information, see “Finding Files Using Tags” on page 7-15.



- 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 “View Details About a File” on page 7-29.



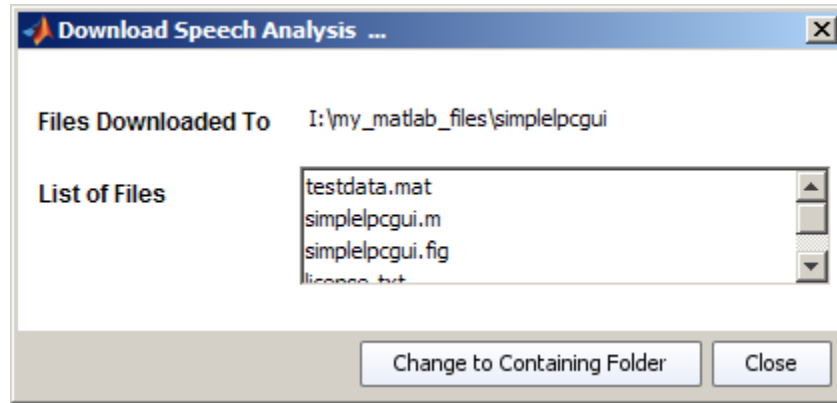
6 Download the file to the current folder:

- 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 “Get Files from the File Exchange Repository” on page 7-31.

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 Current Folder to Download Location** 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 7-36.

For more options to find files, see “Example — Using Tags to Find Files in File Exchange” on page 7-17.

Find Files in File Exchange — Searching and Using Tags

In this section...
“About Finding Files in File Exchange” on page 7-12
“Using Search to Find Files in File Exchange” on page 7-12
“Finding Files by Product, Author, and Other Attributes in File Exchange” on page 7-13
“Using Tags to Find Files in File Exchange” on page 7-13
“Clearing Your Criteria” on page 7-25
“Getting Better Results Using Search and Tags” on page 7-25

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 7-5 .

Syntax for Search Words

To view guidelines for entering search words:

- 1 Click the arrow in the search field.
- 2 From the menu, select **Help Searching for 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 **Help Searching for Files**.

Using Tags to Find Files in File Exchange

- “What Are Tags?” on page 7-13
- “Ways to View Tags” on page 7-14
- “Finding Files Using Tags” on page 7-15
- “Example — Using Tags to Find Files in File Exchange” on page 7-17
- “Applying a Tag to a File” on page 7-24
- “Adding a New Tag to a File” on page 7-24

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” on page 7-14
- “Viewing More Tags for a List of Files” on page 7-14
- “To See Different Tags” on page 7-15
- “Viewing Tags for a File” on page 7-15

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

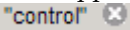
- “Selecting Tags to Find Files” on page 7-15
- “Removing Tags You Already Selected to Expand Results” on page 7-16
- “Directly Entering a Tag Name” on page 7-16

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 **Help Searching for Files**.

Example — Using Tags to Find Files in File Exchange

The example illustrates advanced aspects of using tags to find files.

You want to find files to help you compare your data, which has small sample sizes, to known distributions.

Note The File Exchange repository changes daily, so your results could differ from this example.

- 1 Open File Exchange and change the tag view by selecting **List** from **View tags as**. For more information, see “Ways to View Tags” on page 7-14.

Among the list of tags, the `statistics` tag looks relevant. But because it was applied 712 times, it could be too general. 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 7-16.

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.

The screenshot shows the File Exchange search results for the tag "distribution". The search bar contains "tag:'distribution'" and the results are displayed in a list view. The results are filtered by the "distribution" tag, and the "statistics" tag is selected. The results table shows the following entries:

Sub...	File Summary	Tags	Rating	Downloads
Jun-25-...	Submit MATLAB functions to a cluster Run MATLAB functions on a gridengine/torq... Author: Volkmar Glauche	cluster distributed ...	none	41
Jun-25-...	Use handle classes to model depende... Trigger computations in one object once co... Author: Volkmar Glauche	cluster demo ...	none	9
Jun-19-...	Multicore - Parallel processing on mult... This package provides some functions realizi...	distributed co distributed pr	★ ★ ★	219

At the bottom of the results, it says "1 - 10 of 50 best matches. 64 total matches." and there are pagination controls for pages 1, 2, 3, 4, and 5.

- 3 Narrow your results by selecting the **statistics** tag. For more information, see “Selecting Tags to Find Files” on page 7-15.

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.

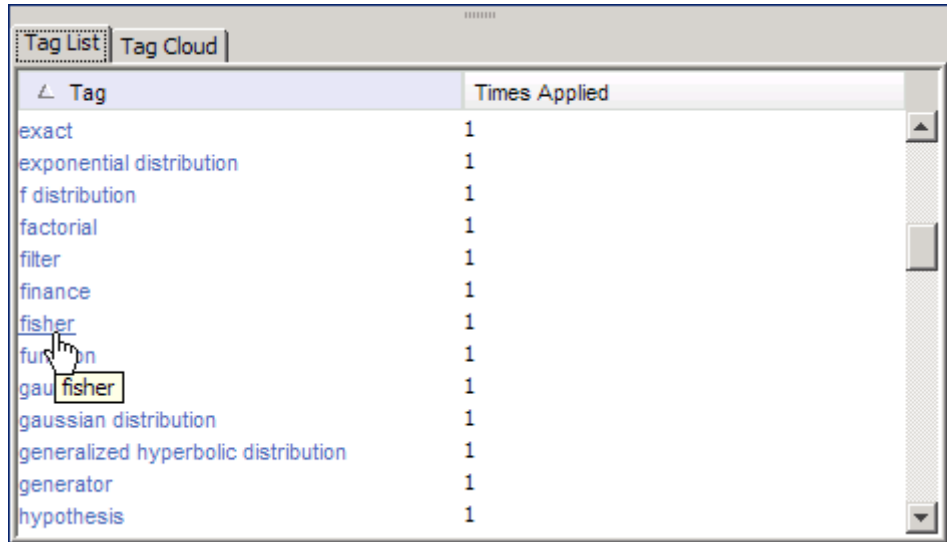


- 4 None of the popular tags are of interest. The FISHERTEST file is close to what you want. But the description shows it as being for a 2-by-2 contingency table and you have a 2-by-3 table.

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 7-14.



The screenshot shows a window titled 'Tag List' with a 'Tag Cloud' tab. It contains a table with two columns: 'Tag' and 'Times Applied'. The table lists various mathematical and statistical terms, each with a count of 1. A mouse cursor is hovering over the 'fisher' tag, and a small box labeled 'fisher' is visible next to it.

Tag	Times Applied
exact	1
exponential distribution	1
f distribution	1
factorial	1
filter	1
finance	1
fisher	1
function	1
gaussian distribution	1
generalized hyperbolic distribution	1
generator	1
hypothesis	1

- 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 7-16.



- 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.

File Exchange

File Desktop Window Help

Logged in as: fx2 | Download History | Log Out | ?

tag:statistics tag:fisher

"statistics" "fisher"

View tags as: List | Cloud

fisher matrix (1) gui (1) matrix power (1)
 fisher4cast (1) interactive plot (1) probability (10)
 fishers exact test (1) latex (1) sample sizes (1)
 fishers exact test 2x3 matrix (1) likelihood (1) statistical (1)
 fishers exact test 2x4 matrix (1) matrix (1) test (2)

More Tags

Submitted	File Summary	Tags	Rating	Down...
May-28-2009	Fisher Matrix Toolbox (Fish... General Fisher Matrix Toolbox a... Author: Yabebal Fantaye	bao cosmology dark energy error ellips error ellipse fisher matrix	★★★ 5.0 1	171
Feb-19-2009	FISHERTEST Fisher Exact test for 2-x-2 con... Author: Jos	chance distribution exact factorial fisher probability ...	none	161
Dec-12-2008	MyFisher24 A very compact routine to com... Author: Giuseppe Cardillo	fishers exact test 2x4 matr probability statistics	none	145
Nov-26-2008	MyFisher33 A very compact routine to com... Author: Giuseppe Cardillo	fisher exact test 3x3 matr probability statistics test	none	169

1 - 6 of 6 best matches. 6 total matches.

- 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 7-38.

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 7-27.

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 <i>all</i> 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: " <i>partial_tagname</i> " 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 " " 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 Help Searching for Files
Be sure that search attributes and values are valid.		For correct names and values, click the arrow in the search field and select Help Searching for Files .
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

View and Sort the List of Files in File Exchange

In this section...

“Viewing the List of Files in File Exchange” on page 7-27

“Sorting the List of Files in File Exchange” on page 7-28


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.

View Details About a File

In this section...
“Viewing the File Details Page” on page 7-29
“Viewing the Contents of a File” on page 7-29

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.

Get Files from the File Exchange Repository

In this section...

“About Downloading Files” on page 7-31

“Downloading from the List of Files” on page 7-31

“Downloading from the File Details Page to a Location You Choose” on page 7-32

“The Default Folder for Downloaded Files” on page 7-32

“Which Location Should You Choose When Downloading Files?” on page 7-32

“Downloading a Submission that Consists of Multiple Files” on page 7-33

“Viewing and Locating Files You Downloaded” on page 7-33

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

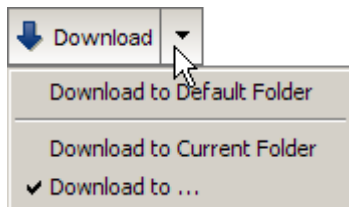
Downloading from the List of Files

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

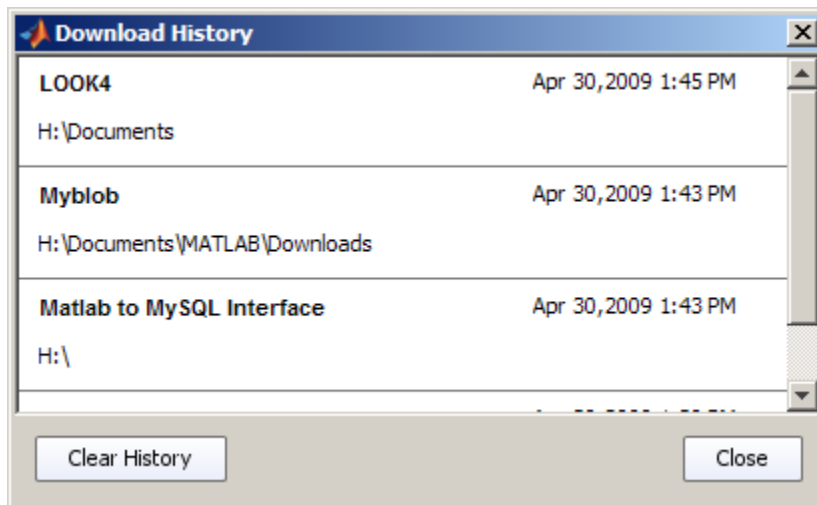
Use the download history to view and locate files you downloaded.

- “Viewing a Log of Files You Downloaded” on page 7-33
- “Locating a File You Downloaded” on page 7-34
- “Viewing Details for a File You Downloaded” on page 7-34
- “Clearing the Download History” on page 7-35

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 Current Folder to Download Location**.

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” on page 7-36
“Consult the File Details Page” on page 7-36
“Look for Updates to the File” on page 7-36
“Read the File” on page 7-37
“Ask Questions” on page 7-37

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 6-73.

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 6-67.
- 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 7-40.
- Contact the author. Select the author name on the file details page to display the author profile, which includes contact information.

Contribute to the File Exchange Repository

In this section...
“How You Can Contribute to the Repository” on page 7-38
“Adding Tags to a File” on page 7-38
“Removing Tags from a File” on page 7-39
“Rating a File” on page 7-39
“Providing Comments About a File” on page 7-40
“Submitting Your Files to the Repository” on page 7-40

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?” on page 7-41
- “How Do I Use File Exchange?” on page 7-41
- “How Does the File Exchange Desktop Tool Relate to File Exchange on the Web Site?” on page 7-42
- “Why Do I See Only 50 Files and How Can I See More?” on page 7-42
- “What Are Tags and How Do I Use Them?” on page 7-43
- “What Are the Tags Above the List of Files?” on page 7-43
- “How Can I See Other Tags?” on page 7-43
- “Why Are the Tags Changing?” on page 7-44
- “Is Search Looking Inside Files?” on page 7-44
- “How Can I Start Over When Looking for Files?” on page 7-44
- “How Can I Choose Where to Download a File To?” on page 7-45
- “How Do I Contribute My Files to the Repository?” on page 7-45

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 “About 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 “Download Files from File Exchange” on page 7-2.

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 “View and Sort the List of Files in File Exchange” on page 7-27.

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 7-13.

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 7-14.

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 7-14.

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 7-13.

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 **Help Searching for 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 7-27.

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 “Get Files from the File Exchange Repository” on page 7-31.

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 “Contribute to the File Exchange Repository” on page 7-38.

Editing and Debugging MATLAB Code

- “Create, Open, Save, and Close Files” on page 8-2
- “Adjust Editor Appearance” on page 8-9
- “Editing Code” on page 8-15
- “File Navigation” on page 8-34
- “Find and Replace Text in Files” on page 8-41
- “Run MATLAB Files in the Editor” on page 8-46
- “Evaluate Subsections of Files Using Code Cells” on page 8-58
- “Avoid Mistakes While Editing Code” on page 8-84
- “Debugging Process and Features” on page 8-116
- “Debugging Functions” on page 8-150
- “About MATLAB Code Files” on page 8-151
- “Editor/Debugger Preferences” on page 8-152
- “About Code Analyzer Preferences” on page 8-161

Create, Open, Save, and Close Files


In this section...

- “Create New Files” on page 8-2
- “Open Existing Files” on page 8-2
- “Save Files” on page 8-4
- “Close Files” on page 8-7
- “Close the Editor” on page 8-8

Create New Files

With the Editor as the active window, you can:

- Create an empty file for MATLAB code or for a text file (such as for an XML or HTML file).

Select **File > New > Script** or click the **New script** button  on the MATLAB desktop toolbar.

Alternatively, you can use the `edit` command.

- Create a file prepopulated with basic MATLAB function, class (`classdef`), enumeration class elements.

Select **File > New** and then select **Function**, **Class**, or **Enumeration**, respectively.


- Create a file from statements in the Command History Window.

- 1 In the Command History window, select the MATLAB statements you want to include in the file.

- 2 Right-click and select **Create Script**.

Open Existing Files

To open an existing file or files in the Editor, choose the option that achieves your goals, as described in this table.

Goal	Steps	Additional Information
<p>Open with associated tool</p> <p>Open a file using the appropriate MATLAB tool for the file type.</p>	<p>Select File > Open or click the Open file button  on the toolbar.</p>	<p>For example, this option opens a file with a .m extension in the Editor and loads a MAT-file into the Workspace browser.</p>
<p>Open as text file</p> <p>Open a file in the Editor as a text file, even if the file type is associated with another application or tool.</p>	<p>Select File > Open as Text.</p>	<p>This is useful, for example, if you have imported a tab-delimited data file (.dat) into the workspace and you find you want to add a data point. Open the file as text in the Editor, make your addition, and then save the file.</p>
<p>Open function from within file</p> <p>Open a subfunction or function file from within a file in the Editor.</p>	<p>Position the cursor on the name within the open file, and then right-click and select Open file-name from the context menu.</p>	<p>You also can use this method to open a variable or Simulink model.</p> <p>For details, see “Open a File or Variable from Within a File” on page 8-39.</p>
<p>Reopen file</p> <p>Reopen a recently used file.</p>	<p>At the bottom of the File menu, select a file.</p>	<p>To change the number of files on the list, select File > Preferences, and then Editor/Debugger. Under Most recently used file list, change the value for Number of entries.</p>

Goal	Steps	Additional Information
<p>Reopen files at startup</p> <p>At startup, automatically open the files that were open when the previous MATLAB session ended.</p>	<p>Select File > Preferences > Editor/Debugger, and then select On restart reopen files from previous MATLAB session.</p>	<p>None.</p>
<p>Open file displaying in another tool</p> <p>Open a file name displaying in another MATLAB desktop tool or Microsoft tool.</p>	<p>Drag the file from the other tool into the Editor.</p>	<p>For example, drag files from the Current Folder browser or from Windows Explorer.</p>
<p>Open file using a function</p>	<p>Use the <code>edit</code> or <code>open</code> function.</p>	<p>For example, type the following to open <code>collatz.m</code>:</p> <pre>edit collatz.m</pre> <p>If <code>collatz.m</code> is not on the search path or in the current folder, use the relative or absolute path for the file.</p>

For special considerations on the Macintosh platform, see “Navigating Within the MATLAB Root Folder on Macintosh Platforms” on page 2-79.

Save Files

After you modify a file, an asterisk (*) follows the file name in the title bar of the Editor. This asterisk indicates that there are unsaved changes to the file.

You can perform four different types of save operations, which have various effects, as described in this table.

Save Option	Steps
Save file to disk and keep file open in the Editor.	Select File > Save .
Rename file, save it to disk, and make it the active Editor document. Original file remains unchanged on disk.	<ol style="list-style-type: none"><li data-bbox="847 409 1159 439">1 Select File > Save As.<li data-bbox="847 470 1299 531">2 Specify a new name, type, or both for the file, and then click Save.
Save file to disk under new name. Original file remains open and unsaved.	<ol style="list-style-type: none"><li data-bbox="847 579 1230 609">1 Select File > Save Backup. MATLAB opens the Select File for Backup dialog box.<li data-bbox="847 730 1280 791">2 Specify a name and type for the backup file, and then click Save.

Save Option	Steps
<p>Save changes to all open files using current file names.</p> <p>All files remain open.</p>	<ol style="list-style-type: none"> 1 Select File > Save All. <p>MATLAB opens the Select File for Save As dialog box for the first unnamed file.</p> 2 Specify a name and type for the unnamed file, and then click Save. 3 Repeat step 2 until all unnamed files are saved.

Recommendations on Saving Files

MathWorks recommends that you save files you create and files from MathWorks that you edit to a folder that is not in the *matlabroot*/toolbox folder tree, where *matlabroot* is the folder returned when you type *matlabroot* in the Command Window. If you keep your files in *matlabroot*/toolbox folders, they can be overwritten when you install a new version of MATLAB software.

At the beginning of each MATLAB session, MATLAB loads and caches in memory the locations of files in the *matlabroot*/toolbox folder tree. Therefore, if you:

- Save files to *matlabroot*/toolbox folders using an external editor, run `rehash toolbox` before you use the files in the current session.
- Add or remove files from *matlabroot*/toolbox folders using file system operations, run `rehash toolbox` before you use the files in the current session.
- Modify existing files in *matlabroot*/toolbox folders using an external editor, run `clear function-name` before you use these files in the current session.

For more information, see `rehash` or “Toolbox Path Caching in the MATLAB Program” on page 1-23.

Autosaving Files

When you modify a file in the Editor, the Editor saves a copy of the file using the same file name but with an `.asv` extension every 5 minutes. The autosave version is useful if you have system problems and lose changes you 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`.

Select **File > Preferences > Autosave** to select preferences to:

- Turn the autosave feature off or on.
- Automatically delete autosave files when you close the corresponding source file.

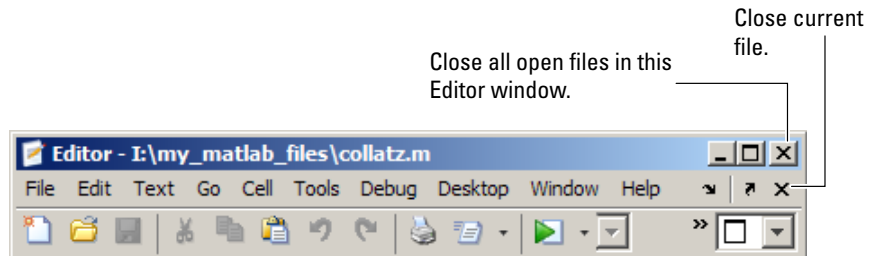
Select **Automatically delete autosave files** to have the Editor automatically delete the corresponding autosave file when you close the Editor. It is best to keep autosave-to-file relationships clear and current. Therefore, when you rename or remove a file, consider deleting or renaming the corresponding autosave file.

- Specify the number of minutes between automatic saves.
- Specify the file extension and location for autosave files.
- Specify a location for autosave files

If you edit a file in a read-only folder and the autosave **Location** preference is **Source file directories**, then the Editor does not create an autosave copy of the file.

Close Files

To close the current file, click the Close button in the Editor menu bar. This is different from the Close button in the title bar of the Editor, which closes all open files in that Editor window.



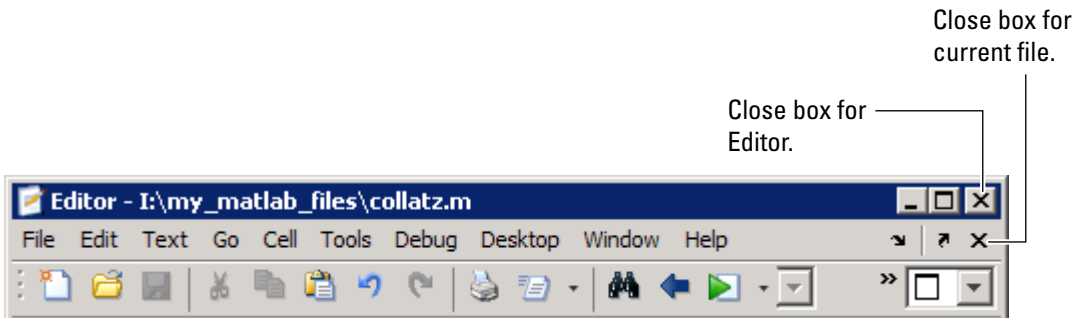
To close all files within the Editor, select **Window > Close Editor Documents**. This menu option does not close any files undocked from the Editor. The Editor remains open with no files in it.

If each file is open in a separate window, close all the files at once using **Close All Documents** in the **Window** menu. This 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 button. The prompt does not appear and the document closes without saving any unsaved changes.

Close the Editor

To close the Editor, click the Close button in the *title bar* of the Editor. This button is different from the Close button in the Editor *menu bar*. The latter Close button closes the active document when multiple documents are open in a single window.



Adjust Editor Appearance

In this section...
“Line and Column Number Indicators” on page 8-9
“Right-Side Text Limit Indicator” on page 8-10
“Cursor Location Indicator — Class, Function, or Subfunction” on page 8-11
“Display Two Parts of a File Simultaneously” on page 8-11

See also:

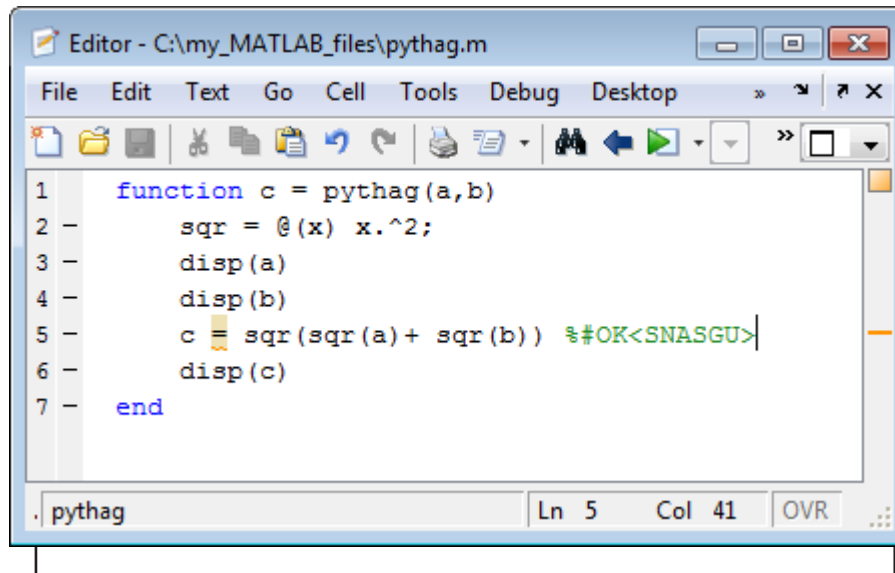
- “Open and Rearrange Desktop Tools and Documents” on page 2-13
- “Colors in the MATLAB Editor — What Do They Mean?” on page 8-30

Line and Column Number Indicators

Line numbers display along the left side of the Editor window. The line and column numbers for the current cursor position display on the right side of the Editor status bar.

To set preferences for displaying line numbers:

- 1** Select **File > Preferences > Editor/Debugger > Display**.
- 2** Select or clear **Show line numbers**, and then click **OK**.



Status bar

Right-Side Text Limit Indicator

By default, a light gray vertical line (rule) appears at column 75 in the Editor, indicating where a line exceeds 75 characters. You can set this text limit indicator to another value, which is useful, for example, if you want to view the code in another text editor that has a different line width limit.

To hide, or change the appearance of the vertical line, change **Display** preferences by selecting **File > Preferences > Editor/Debugger > Display**.

Note This limit is a visual cue only and does not prevent text from exceeding the limit. For information on setting a value to wrap comment text at a specified column number automatically, see “Wrap Comments” on page 8-29.

Cursor Location Indicator — Class, Function, or Subfunction

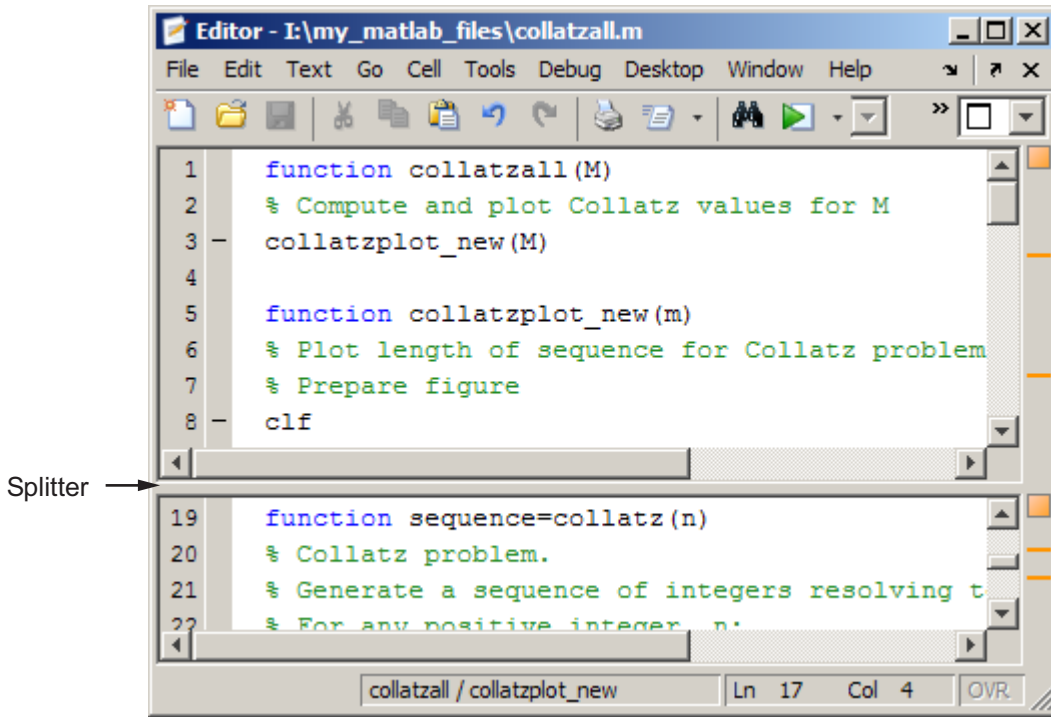
The right side of the Editor status bar shows the class, function, or subfunction where the cursor is currently placed, depending on the type of file you are viewing:

- Class file — The name of the class followed by the name of the current function that the cursor is within (if any).
- Function file — The name of the main function followed by the name of the current function the cursor is within (if any).

Display Two Parts of a File Simultaneously

You can simultaneously display two different parts of a file in the Editor by splitting the screen display, as shown in the image that follows. This feature makes it easy to compare different lines in a file or to copy and paste from one part of a file to another.

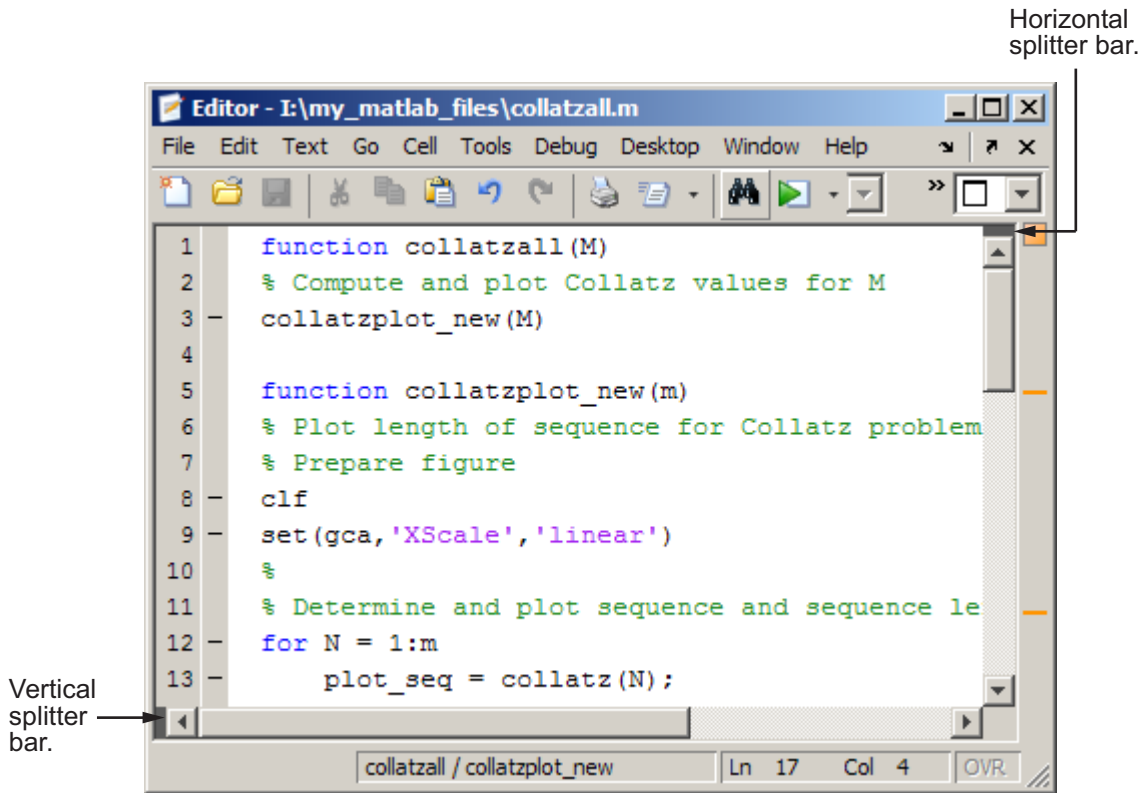
See also “Position Documents” on page 2-27 for instructions on displaying multiple documents simultaneously.



The following table describes the various ways you can split the Editor and manipulate the split-screen views. When you open a document, it opens unsplit, regardless of its split status it had when you closed it.

Operation	Instructions
Split the screen horizontally.	<p>Do either of the following</p> <ul style="list-style-type: none"> • Select Window > Split Screen > Top/Bottom. • If there is a vertical scroll bar, as shown in the illustration that follows, drag the splitter bar down.
Split the screen vertically.	<p>Do either of the following:</p> <ul style="list-style-type: none"> • Select Window > Split Screen > Left/Right.

Operation	Instructions
	<ul style="list-style-type: none">• If there is a horizontal scroll bar, as shown in the illustration that follows, drag the splitter bar from the left of the scroll bar.
Specify the active view.	Do either of the following: <ul style="list-style-type: none">• Select Window > Split Screen > Switch Focus.• Click in the view you want to make active. Updates you make to the document in the active view are also visible in the other view.
Remove the splitter	Do one of the following: <ul style="list-style-type: none">• Double-click the splitter.• Select Window > Split > Screen > Off.



Editing Code

In this section...

“Insert Text and Overwrite Existing Text” on page 8-15
“Change the Case of Selected Text” on page 8-16
“Undo and Redo Editor Actions” on page 8-16
“Highlight Current Line” on page 8-16
“Indent Code” on page 8-17
“Code Folding — Expand and Collapse Code Constructs” on page 8-20
“Add Comments” on page 8-26
“Wrap Comments” on page 8-29
“Colors in the MATLAB Editor — What Do They Mean?” on page 8-30
“Code Contains %*##ok* — What Does That Mean?” on page 8-33

See also:

- “Avoid Mistakes While Editing Code” on page 8-84
- “Fonts” on page 2-2
- “Color Settings” on page 2-8

Insert Text and Overwrite Existing Text

On Windows and UNIX platforms, you can enter text in the Editor that inserts within existing text, or that overwrites existing text, as follows:

Note The Macintosh platform does not support overwrite mode.

1 In the Editor, place the cursor where you want to enter text.

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.

- 2 Press the **Insert** key to toggle the typing mode from insert to overwrite mode, or the reverse.

The **Insert** key is the default keyboard shortcut for changing the typing mode. For details, on changing keyboard shortcuts, see “Customize Keyboard Shortcuts” on page 2-48.

Change the Case of Selected Text

To change the case of text in the Editor:

- 1 Select the text.
- 2 From the **Text** menu, select one of the following:
 - **Change to Upper Case** to change all text to capital letters (A ,B, C,...)
 - **Change to Lower Case** to change all text to small letters (a, b, c, ...)

Undo and Redo 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 operation.

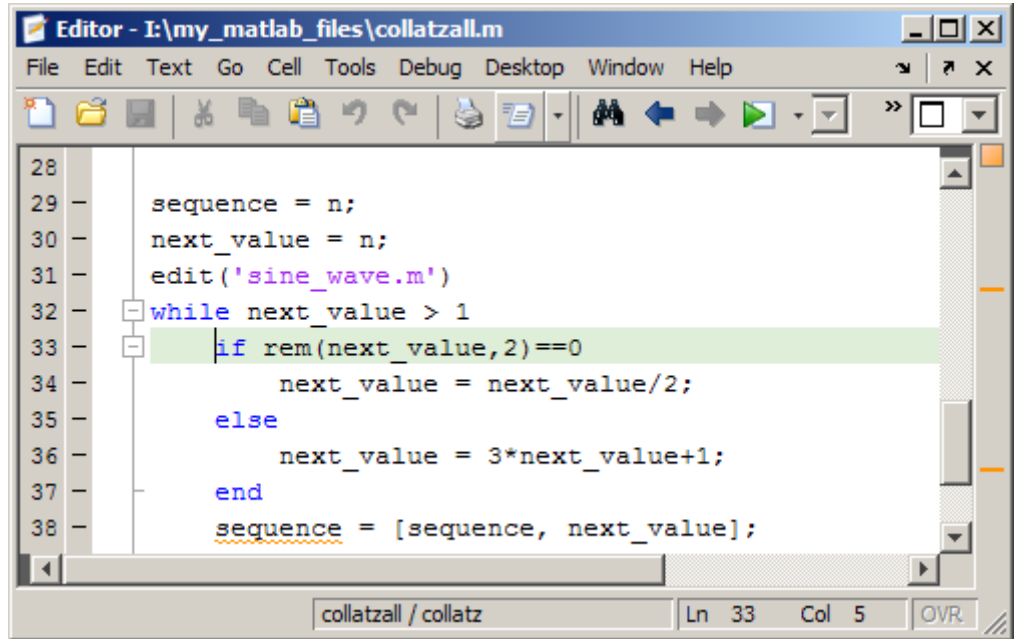
Highlight Current Line

You can set a preference to highlight the current line, that is, the line with the cursor. This helps you see where copied text will be inserted when you paste, for example.

To highlight the current line:

- 1 Select **File > Preferences > Editor/Debugger > Display**.
- 2 Under **General display options**, select **Highlight current line**.
- 3 If you want a highlight color other than green, click the color palette and choose a different color.

In the following image, the current line is highlighted in the default highlighting color—green.



Indent Code

Indenting code makes reading statements such as `while` loops easier. To set and apply indenting preferences to code in the Editor:

Note Indenting preferences are not available for TLC, VHDL, or Verilog.

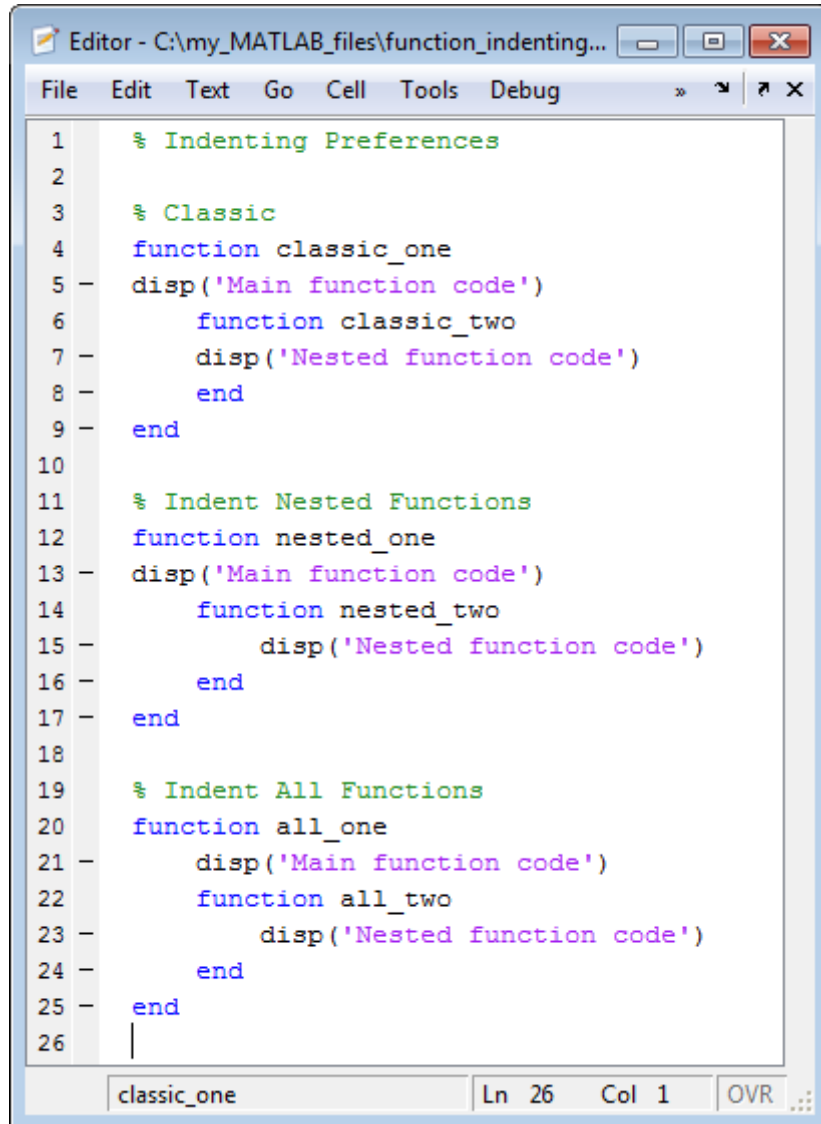
- 1 Select **File > Preferences > Editor/Debugger > Language**.
- 2 From the **Language** drop-down menu, select a language.
- 3 Select or clear **Apply smart indenting while typing**, depending on whether you want indenting applied automatically, as you type.

If you clear this option, you can manually apply indenting by selecting the lines in the Editor to indent, right-clicking, and then selecting **Smart Indent** from the context menu.

4 Do one of the following:

- If you chose any language other than MATLAB in step 2, click **OK**.
- If you chose MATLAB in step 2, select a **Function indenting format**, and then click **OK**. Function indent formats 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.

The following image illustrates the function indenting formats.



The screenshot shows a MATLAB editor window titled "Editor - C:\my_MATLAB_files\function_indenting...". The window contains the following code:

```
1  % Indenting Preferences
2
3  % Classic
4  function classic_one
5 - disp('Main function code')
6     function classic_two
7 -     disp('Nested function code')
8 -     end
9 - end
10
11 % Indent Nested Functions
12 function nested_one
13 - disp('Main function code')
14     function nested_two
15 -         disp('Nested function code')
16 -     end
17 - end
18
19 % Indent All Functions
20 function all_one
21 -     disp('Main function code')
22     function all_two
23 -         disp('Nested function code')
24 -     end
25 - end
26 |
```

The status bar at the bottom of the editor shows "classic_one", "Ln 26", "Col 1", and "OVR".

Regardless of whether you apply indenting automatically or manually, you can move selected lines further to the left or right, by doing one of the following:

- Selecting **Text > Decrease Indent** or **Text > Increase Indent**, respectively.
- Pressing the **Tab** key or the **Shift+Tab** key, respectively.

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.

Code Folding — Expand and Collapse Code Constructs

Code folding is the ability to expand and collapse certain MATLAB programming constructs. This improves readability when a 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. MATLAB programming constructs include:

- Cells used for running sections of code and publishing code
- Class code
- For and parfor blocks
- Function and class help
- Function code

To see the entire list of constructs, select **File > Preferences > Editor/Debugger > Code Folding**.

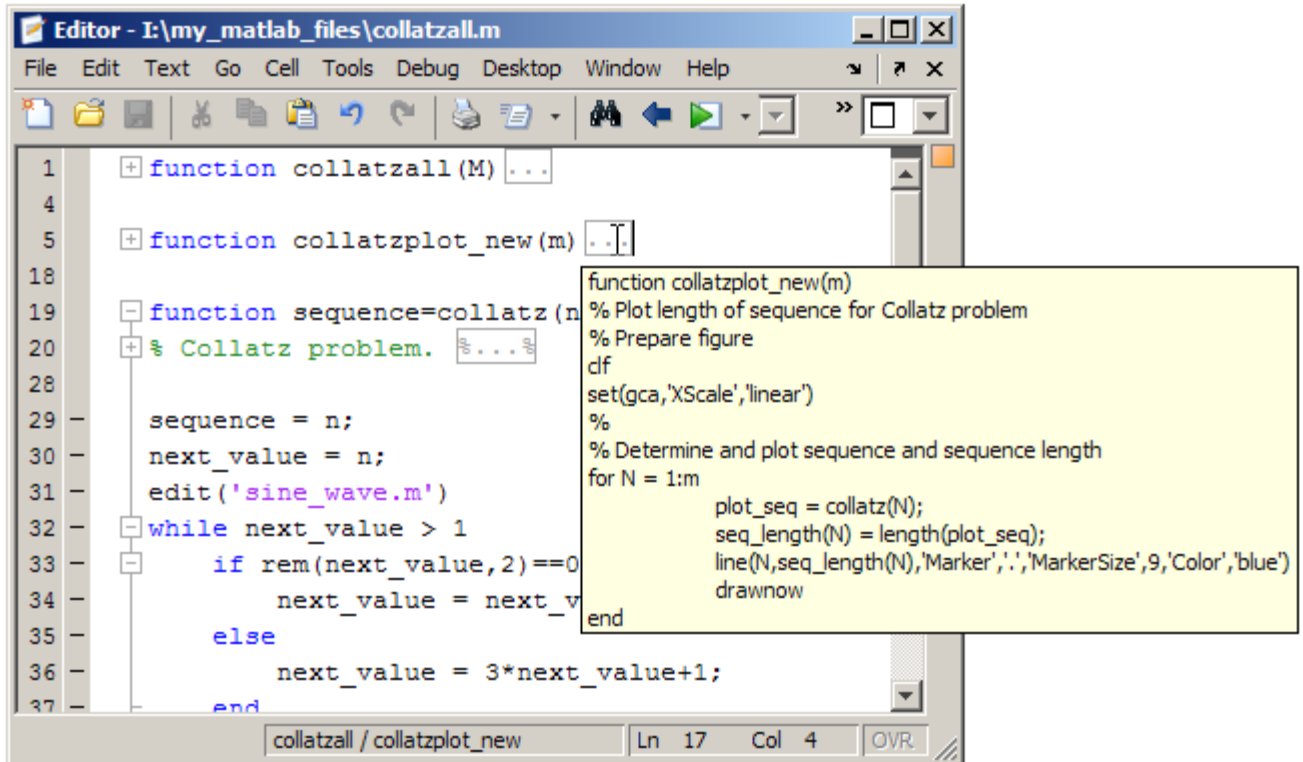
To expand or collapse code, click the plus **+** or minus sign **-** that appears to the left of the construct.

To expand or collapse all of the code in a file, place your cursor anywhere within the file, right-click, and then select **Code Folding > Expand All** or **Code Folding > Collapse All** from the context menu.

View 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.

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.



Print Files with Collapsed Code

If you print a file with one or more collapsed constructs, those constructs are expanded in the printed version of the file.

Code Folding Behavior for Functions that Have No Explicit End Statement

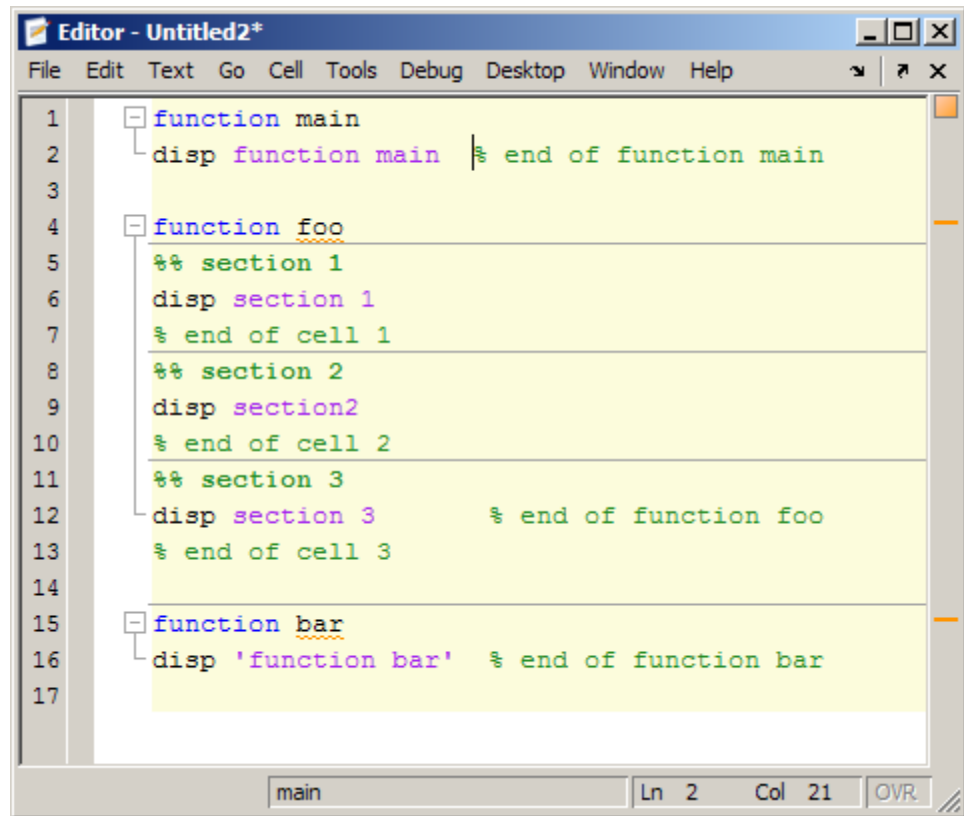
If you enable code folding for functions and a function in your code does not end with an explicit end statement, you see the following behavior:

- If a line containing only comments appears at the end of such a function, then the Editor does not include that line when folding the function. MATLAB does not include trailing white space and comments in a function definition that has no explicit end statement.

Code Folding Enabled for Function Code Only on page 8-23 illustrates this behavior. Line 13 is excluded from the fold for the `foo` function.

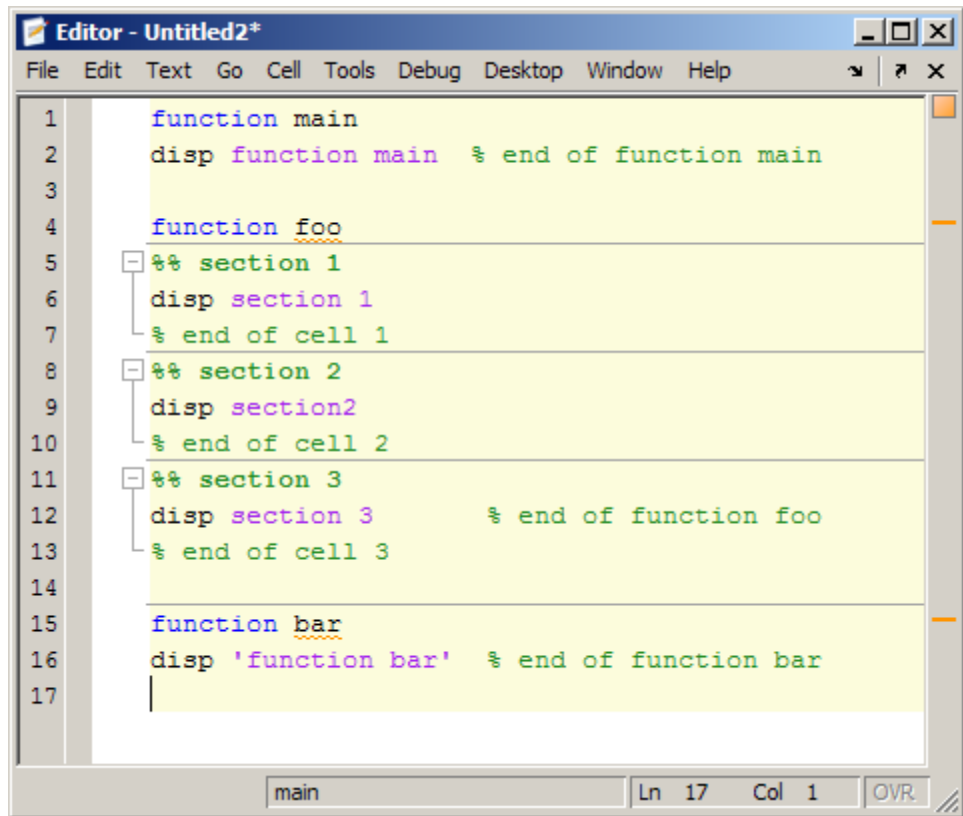
- If a fold for a cell overlaps the function code, then the Editor does not show the fold for the overlapping cell.

The three figures that follow illustrate this behavior. The first two figures, Code Folding Enabled for Function Code Only on page 8-23 and Code Folding Enabled for Cells Only on page 8-24 illustrate how the code folding appears when you enable it for function code only and then cell code only, respectively. The last figure, Code Folding Enabled for Both Functions and Cells on page 8-25, illustrates the effects when code folding is enabled for both. Because the fold for cell 3 (lines 11–13) overlaps the fold for function `foo` (lines 4–12), the Editor does not display the fold for cell 3.



```
1 function main
2   disp function main   % end of function main
3
4 function foo
5   %% section 1
6   disp section 1
7   % end of cell 1
8   %% section 2
9   disp section2
10  % end of cell 2
11  %% section 3
12  disp section 3      % end of function foo
13  % end of cell 3
14
15 function bar
16  disp 'function bar' % end of function bar
17
```

Code Folding Enabled for Function Code Only

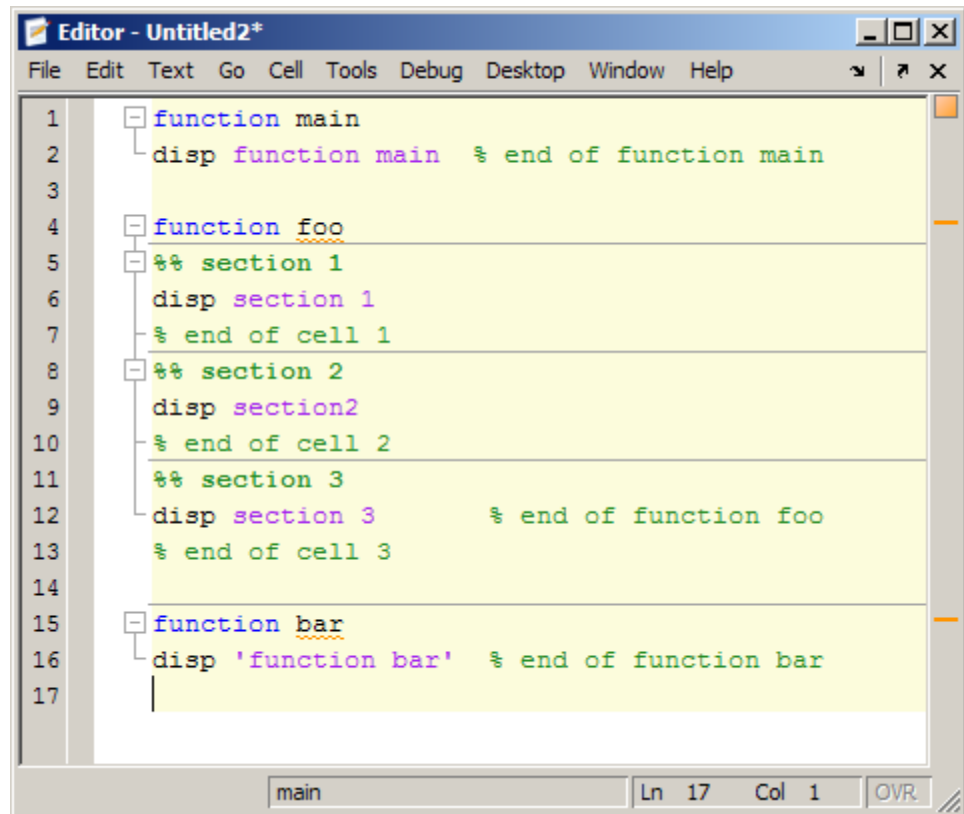


The screenshot shows the MATLAB Editor window titled "Editor - Untitled2*". The menu bar includes File, Edit, Text, Go, Cell, Tools, Debug, Desktop, Window, and Help. The code is as follows:

```
1 function main
2 disp function main % end of function main
3
4 function foo
5 %% section 1
6 disp section 1
7 % end of cell 1
8 %% section 2
9 disp section2
10 % end of cell 2
11 %% section 3
12 disp section 3 % end of function foo
13 % end of cell 3
14
15 function bar
16 disp 'function bar' % end of function bar
17
```

The code is displayed with syntax highlighting. The "function" keywords are blue, "disp" is purple, and comments are green. The code is organized into three cells, each with a collapse icon (a square with a minus sign) on the left margin. The first cell (lines 5-7) is collapsed, the second (lines 8-10) is expanded, and the third (lines 11-13) is collapsed. The status bar at the bottom shows "main", "Ln 17 Col 1", and "OVR".

Code Folding Enabled for Cells Only

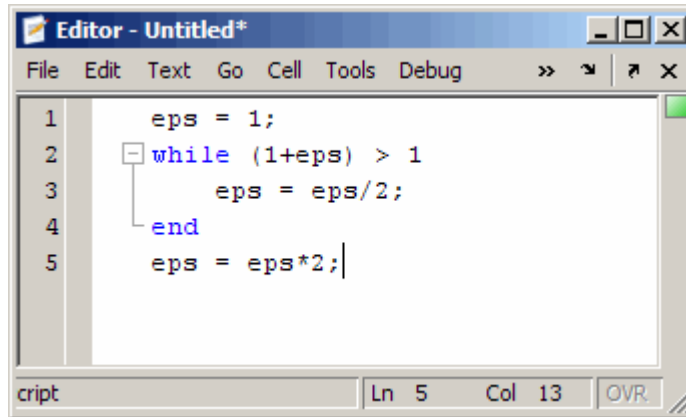


```
1  function main
2  disp function main % end of function main
3
4  function foo
5  %% section 1
6  disp section 1
7  % end of cell 1
8  %% section 2
9  disp section2
10 % end of cell 2
11 %% section 3
12 disp section 3      % end of function foo
13 % end of cell 3
14
15 function bar
16 disp 'function bar' % end of function bar
17
```

Code Folding Enabled for Both Functions and Cells

Effect of Syntax Errors on Code Folding

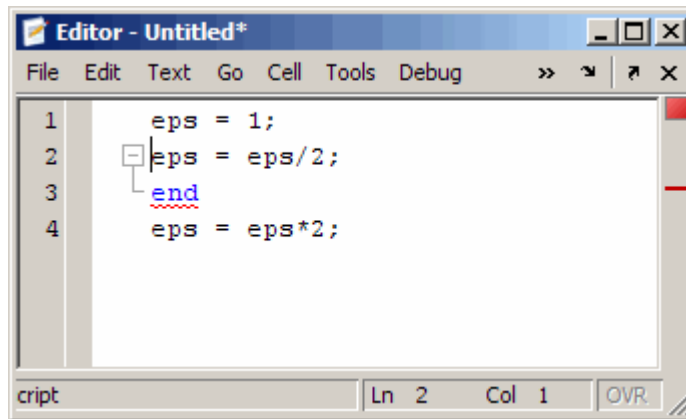
If your code contains syntax errors, the code folding indicators might appear to be in the wrong location. For example, suppose your code currently appears as shown in the first figure that follows. If you delete the `while` statement, it introduces a syntax error at line 3, as shown in the second figure 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.



The image shows a MATLAB Editor window titled "Editor - Untitled*" with a menu bar (File, Edit, Text, Go, Cell, Tools, Debug) and standard window controls. The code in the editor is as follows:

```
1     eps = 1;
2     while (1+eps) > 1
3         eps = eps/2;
4     end
5     eps = eps*2;
```

The status bar at the bottom indicates the file name "cript", the cursor position "Ln 5 Col 13", and the view mode "OVR". A green cursor is visible at the end of line 5.



The image shows a MATLAB Editor window titled "Editor - Untitled*" with a menu bar (File, Edit, Text, Go, Cell, Tools, Debug) and standard window controls. The code in the editor is as follows:

```
1     eps = 1;
2     while (1+eps) > 1
3         eps = eps/2;
4     end
5     eps = eps*2;
```

The status bar at the bottom indicates the file name "cript", the cursor position "Ln 2 Col 1", and the view mode "OVR". A red cursor is visible at the end of line 2.

Add Comments

Comments in MATLAB code are strings or statements that do not execute. Add comments in a 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 file runs without those lines. These topics provide details:

- “Commenting in MATLAB Code Using the MATLAB Editor” on page 8-27

- “Commenting in Java and C/C++ Files Using the MATLAB Editor” on page 8-27
- “Commenting in MATLAB Code Using Any Text Editor” on page 8-27
- “Commenting Out Part of a Statement” on page 8-28

Commenting in MATLAB Code Using the MATLAB Editor

You can comment the current line or a selection of lines in MATLAB code.

- 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 Right-click and select **Comment** from the context menu.

The Editor adds a comment symbol, `%` at the start of each commented line.

To uncomment the current line or a selected group of lines, right-click and select **Uncomment** from the context menu.

Commenting in Java and C/C++ Files Using the MATLAB Editor

To add comments to selected lines in Java and C/C++ files, select **Text > Comment**. The Editor adds the `//` symbols at the front of the selected lines.

To remove comments from selected lines in Java and C/C++ files, select **Text > Uncomment**. The Editor removes the `//` symbols from the front of the selected lines.

Commenting in MATLAB Code Using Any Text Editor

You can make any line in MATLAB code 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. For instance, if the following line appears in a file, then MATLAB ignores the line when you run the file:

```
% This is 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 no other text. Remove the block comment symbols, `%{` and `%}`, to uncomment the lines.

This example shows some lines of code commented out. When you run the file, the commented lines do not execute. This is useful when you want to identify the section of a file that is not working as expected.

Comment a block of code by adding `%{` before the first line and `%}` after the last line.

```

a = magic(3)
%{
sum(a)
diag(A)
sum(diag(a))
%}
sum(diag(fliplr(a)))

```

You can create a nested block comment, as shown in the following example.

Create a nested comment, that is, a block comment within a block comment.

Extended comment

```

%{
a = magic(3)
%{
sum(a)
diag(A)
sum(diag(A))
%}
sum(diag(fliplr(a)))
%}

```

Original comment

Commenting Out Part of a Statement

To comment out the end of a statement, put the comment character, `%`, before the comment. When you run the file, MATLAB software ignores any text on the line after the `%`.

```
a = zeros(10) % Initialize matrix
```

Any text following a % within a line
is considered to be a comment.

To comment out text within a multiline statement, you must use an ellipsis (...); the comment character (%) does not work in this context. MATLAB ignores any text appearing after the ellipsis 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 ellipsis. The following example comments out the Middle Initial line.

```
header = ['Last Name, '...  
'First Name, '...  
... 'Middle Initial, '...  
'Title']
```

Notice that Middle Initial is green, which is the syntax highlighting color for a comment.

MATLAB effectively runs

```
headers = ['Last Name, ' ...  
'First Name, ' ...  
'Title']
```

Wrap Comments

By default, as you type them in the Editor, comments wrap whenever they reach a column width of 75. In addition, you can manually wrap comment lines, such as those that you paste into a file.

Wrap Comments Manually

- 1 Place the cursor anywhere within a block of contiguous lines of comments.
- 2 From the context menu, select **Wrap Comments**.

MATLAB wraps all the comments in the block. To wrap only a subset of the comments, select the subset before you wrap it.

To change the column at which comments wrap, or to disable automatic comment wrapping, set **Comment Formatting** preferences using “Editor/Debugger Language Preferences” on page 8-155.

Exclusions from Comment Wrapping

When you wrap comments, the Editor does *not* wrap the following:

- A bulleted list item onto the line that precedes it
A bulleted list item begins with * or # .
- Code cell titles (%% followed by a space)
- Long strings, such as URLs

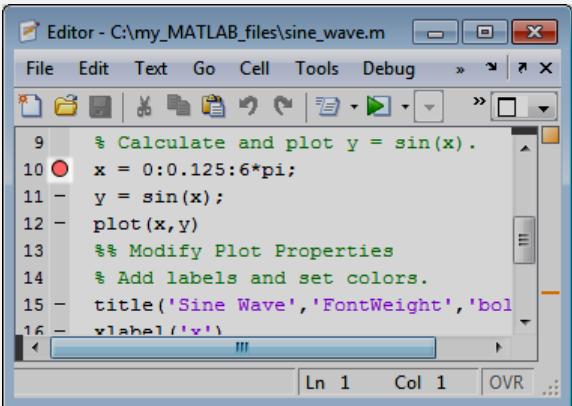
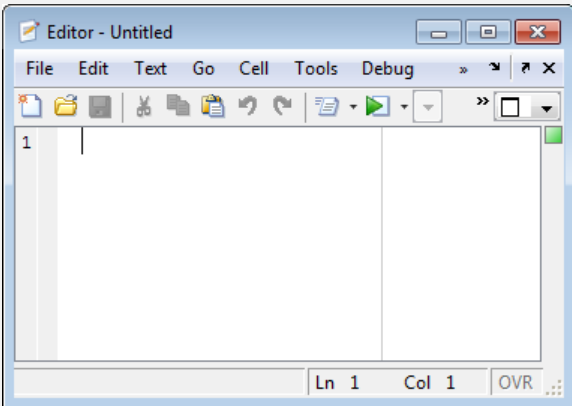
This behavior preserves the formatting required for publishing.

Colors in the MATLAB Editor – What Do They Mean?

Colors in the Editor help you to read code, identify code elements, and evaluate sections of code. To find out why certain portions of your code appear in color, how to change the color, or learn more about the features highlighted in color, see the table that follows.

Sample (Using Default Colors)	What the Color Indicates
<pre> %create a file for output ! touch testFile.txt fid = fopen('testFile.txt', 'w'); for i = 1:10 fprintf(fid, '%6.2f \n,i); end </pre>	<p>Different types of language elements, such as keywords, comments, and strings appear in different colors. This is called <i>syntax highlighting</i>.</p> <p>See also, “Highlight Syntax to Help Ensure Correct Entries in the Editor” on page 8-84</p>
<pre> function test(input) persistent multfactor; multfactor = multfactor +1; a_var = nestedfunction(multfactor) unused_var = a_var; function out = nestedfunction(factor) out=input*factor; end end </pre>	<p>Teal blue characters indicate variables with shared scope</p> <p>Sky blue shading indicates function, subfunction, or variable names that match the name in which the cursor is currently placed.</p> <p>See also, “Avoid Variable and Function Scoping Problems” on page 8-110.</p>
<pre> if isempty(data{3}) ... (length(unique(data{1}(:)))==1) ... (length(unique(data{2}(:)))==1) ... (length(unique(data{3}(:)))==1) ... data{3} = zeros(size(data{1})); dime(n1) = 2; else dime(n1) = 3; end </pre>	<p>Orange and red wavy underlines indicate warning and error conditions, respectively.</p> <p>Orange shading indicates coding issues that MATLAB can correct for you.</p> <p>See also, “Check Code for Errors and Warnings” on page 8-91</p>

Sample (Using Default Colors)	What the Color Indicates
<pre data-bbox="140 338 675 694">% Plot Sine Wave % Calculate and plot a sine wave. % Calculate and Plot Sine Wave % Define the range for x. % % \$\$00<= x <= pi\$\$ % % Calculate and plot y = sin(x). x = 0:0.125:6*pi; y = sin(x); plot(x,y) % Modify Plot Properties % Add labels and set colors.</pre>	<p data-bbox="742 303 1328 329">Yellow highlighting indicates code cells, which:</p> <ul data-bbox="742 364 1328 503" style="list-style-type: none"><li data-bbox="742 364 1328 425">• Help you visually identify subsections of code<li data-bbox="742 442 1328 503">• Enable you to publish and run subsections of code. <p data-bbox="742 538 1328 564">See also, “What Are Code Cells?” on page 8-58</p>

Sample (Using Default Colors)	What the Color Indicates
 <p>The screenshot shows the MATLAB Editor window titled 'Editor - C:\my_MATLAB_files\sine_wave.m'. The code is as follows:</p> <pre> 9 % Calculate and plot y = sin(x). 10 x = 0:0.125:6*pi; 11 y = sin(x); 12 plot(x,y) 13 %% Modify Plot Properties 14 % Add labels and set colors. 15 title('Sine Wave','FontWeight','bold') 16 xlabel('x') </pre> <p>A red dot is placed on the left margin of line 10, indicating a breakpoint. The status bar at the bottom shows 'Ln 1 Col 1 OVR'.</p>	<p>Red dots represent breakpoints, which you use in debugging.</p> <p>If you attempt to run your code, MATLAB stops at the first breakpoint it encounters.</p> <p>See also, “Debugging Process and Features” on page 8-116</p>
 <p>The screenshot shows the MATLAB Editor window titled 'Editor - Untitled'. The editor is currently empty, with a single line of text at the top. A gray vertical line is positioned on the right side of the editor, indicating a text limit. The status bar at the bottom shows 'Ln 1 Col 1 OVR'.</p>	<p>A gray vertical line indicates the location of a particular column in the Editor that you can use to limit line widths.</p> <p>The Editor does not enforce the limit.</p> <p>See also, “Right-Side Text Limit Indicator” on page 8-10</p>

Code Contains %#ok – What Does That Mean?


If code contains the string `%#ok` at the end of a line of code, it indicates that one or more Code Analyzer messages is suppressed. For more information, see “Understand Code Containing Suppressed Messages” on page 8-104.



File Navigation



In this section...
“Navigate to a Specific Location” on page 8-34
“Set Bookmarks” on page 8-38
“Navigate Backward and Forward in Files” on page 8-38
“Open a File or Variable from Within a File” on page 8-39






Navigate to a Specific Location

This table summarizes the steps for navigating to a specific location within a file open in the Editor. In some cases, different sets of steps are available for navigating to a particular location. Choose the set that works best with your workflow.

Go To	Steps	Notes
Line Number	<ol style="list-style-type: none"> 1 Select Go > Go To 2 Specify the line to which you want to navigate. 	None
Function definition	<ol style="list-style-type: none"> 1 Click the Show Functions button  on the Editor toolbar. 2 From the list that appears, select the subfunction or nested function to which you want to navigate. <p>Includes subfunctions and nested functions</p>	<p>Includes subfunctions and nested functions</p> <p>For both class and function files, the functions list in alphabetical order—except that in function files, the name of the main function always appears at the top of the list.</p>

Go To	Steps	Notes
	<ol style="list-style-type: none"> 1 Select Go > Go To 2 Select the subfunction or nested function to which you want to navigate. <p>Includes subfunctions and nested functions</p>	<p>Functions list in alphabetical order. To order them by their position in the file, click the Line column heading.</p> <p>The list shows lines in the current file that begin with a function statement. The list does not include functions that the file calls.</p>
	<ol style="list-style-type: none"> 1 In the Current Folder browser, click the name of the file open in the Editor. 2 Click the up arrow  at the bottom of Current Folder browser to open the detail panel. 3 In the detail panel, double-click the function icon  corresponding to the title of the function or subfunction to which you want to navigate. 	<p>Functions list in order of appearance within your file.</p>

Go To	Steps	Notes
Function reference	<ol style="list-style-type: none"> 1 Click in any instance of the function name. 2 Press Alt+Up or Alt+Down to go to the next or previous function reference, respectively. 	<p>Alt+Up and Alt+Down are the default keyboard shortcuts for the actions Go to Previous Underline or Highlight or Go to Next Underline or Highlight, respectively.</p> <p>For more information, see “Keyboard Shortcuts” on page 2-36.</p>
Variable reference	<ol style="list-style-type: none"> 1 Click in any instance of the variable name. 2 Press Alt+Up or Alt+Down to go to the next or previous variable reference, respectively. 	
Code Analyzer Message	<p>Press Alt+Up or Alt+Down to go to the next or previous code analyzer message, respectively.</p>	
Code Cell	<ol style="list-style-type: none"> 1 Click the Show Cell Titles button  on the Editor Cell Mode toolbar. 2 From the list that appears, select the title of the code cell to which you want to navigate. 	<p>For more information, see “Define Code Cells” on page 8-61</p>
	<ol style="list-style-type: none"> 1 Select Go > Go To 2 Select the title of the code cell to which you want to navigate. 	
	<ol style="list-style-type: none"> 1 In the Current Folder browser, click the name of the file that is open in the Editor. 2 Click the up arrow  at the bottom of Current Folder browser to open the detail panel. 	

Go To	Steps	Notes
	<ol style="list-style-type: none"> 3 In the detail panel, double-click the cell icon  corresponding to the title of the cell to which you want to navigate. 	
Property	<ol style="list-style-type: none"> 1 In the Current Folder browser, click the name of the file that is open in the Editor. 2 Click the up arrow  at the bottom of Current Folder browser to open the detail panel. 3 On the detail panel, double-click the property icon  corresponding to the name of the property to which you want to navigate. 	For more information, see “How to Use Properties”
Method	<ol style="list-style-type: none"> 1 In the Current Folder browser, click the name of the file that is open in the Editor. 2 Click the up arrow  at the bottom of Current Folder browser to open the detail panel. 3 In the detail panel, double-click the icon  corresponding to the name of the method to which you want to navigate. 	For more information, see “How to Use Methods”
Bookmark	Select Go > Next Bookmark , or Go > Previous Bookmark .	For information on setting and clearing bookmarks, see “Set Bookmarks” on page 8-38.

Set Bookmarks

You can set a bookmark at any line in a file in the Editor so you can quickly navigate to the bookmarked line. This is particularly useful in long files. For example, suppose while working on a line, you want 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 use the bookmark to return.

To set a bookmark:

- 1 Position the cursor anywhere on the line.
- 2 Select **Go > Set/Clear Bookmark**.

A bookmark icon  appears to the left of the line.

To clear a bookmark, position the cursor anywhere on the line and select **Go > Select/Clear Bookmark**.

MATLAB does not maintain bookmarks after you close a file.

Navigate Backward and Forward in Files

To access lines in a file in the same sequence that you previously navigated or edited them, do one of the following:

- Select **Go > Back** or click .
- Select **Go > Forward** or click .

Interrupting the Sequence of Go Back and Go Forward

The **Go > Back** or **Go > Forward** feature sequence is interrupted if you:

- 1 Select **Go > Back**.
- 2 Select **Go > Forward**.
- 3 Edit a line or navigate to another line using the list of features described in “Navigate to a Specific Location” on page 8-34.

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.
- 2 Edit line 2, line 4, and line 6.
- 3 Use **Go > Back** to return to line 4, and then to return to line 2.
- 4 Use **Go > Forward** to return to lines 4 and 6.
- 5 Use **Go > Back** to return to line 1.
- 6 Edit at 3.

This interrupts the sequence. You can no longer use **Go > Forward** to return to lines 4 and 6. You can, however, use **Go > Back** to return to line 1.

Open a File or Variable from Within a File

You can open a subfunction, function, file, variable, or Simulink model from within a file in the Editor. Position the cursor on 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 this table.

Item	Action
Subfunction	Navigates to the subfunction within the current file, if that file is a MATLAB code file. If no subfunction by that name exists in the current file, the Editor runs the <code>open</code> function on the selection, which opens the selection in the appropriate tool.
Text file	Opens in the Editor.
Figure file (.fig)	Opens in a figure window.

Item	Action
MATLAB variable that is in the current workspace	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.

Find and Replace Text in Files

In this section...

“Find Any Text in the Current File” on page 8-41

“Find and Replace Functions or Variables in the Current File” on page 8-41

“Automatically Rename All Functions or Variables in a File” on page 8-43

“Find and Replace Any Text” on page 8-45

“Find Text in Multiple File Names or Files” on page 8-45

“Function Alternative for Finding Text” on page 8-45

“Perform an Incremental Search in the Editor” on page 8-45

Find Any Text in the Current File

- 1 Within the current file, select the text you want to find.
- 2 From the **Edit** menu, select **Find Selection**.

The next occurrence of that text is selected.

- 3 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.

Find and Replace Functions or Variables in the Current File

To search for references to a particular function or variable, use the automatic highlighting feature for variables and functions. This feature is more efficient than using the text finding tools. Function and variable highlighting indicates only references to a particular function or variable, not other occurrences. For instance, it does not find instances of the function or variable name in comments. Furthermore, variable highlighting only includes references to

the *same* variable. That is, if two variables use the same name, but are in different scopes, highlighting one does not cause the other to highlight.

- 1** Select **File > Preferences > Colors > Programming Tools**.
- 2** Under **Variable and function colors**, select **Automatically highlight**, deselect **Variables with shared scope**, and then click **Apply**.
- 3** In a file open in the Editor, click an instance of the variable you want to find throughout the file.

MATLAB indicates all occurrences of that variable within the file by:

- Highlighting them in teal blue (by default) throughout the file
- Adding a marker for each in the indicator bar

If a code analyzer indicator and a variable indicator appear on the same line in a file, the marker for the variable takes precedence.

- Specifying the number of usages it finds in the status bar
- 4** Hover over a marker in the indicator bar to see the line it represents.
 - 5** Click a marker in the indicator bar to navigate to that occurrence of the variable.

Replace an instance of a function or variable by editing the occurrence at a line to which you have navigated.

The following image shows an example of how the Editor looks with variable highlighting enabled. In this image, the variable `i` appears highlighted in sky blue, the indicator bar contains three variable markers, and the status bar (on the lower left edge of the tool) indicates 5 usages of `i` found.

```

1  function rowTotals = rowsum
2  % Add the values in each row and
3  % store them in a new array.
4
5  x = ones(2,10);
6  [n, m] = size(x);
7  rowTotals = zeros(1,n);
8  for i = 1:n
9      rowTotals(i) = addToSum;
10 end
11
12     function colsum = addToSum
13         colsum = 0;
14         thisrow = x(i,:);
15         for i = 1:m
16             colsum = colsum + thisrow(i);
17         end
18     end
19
20 end

```

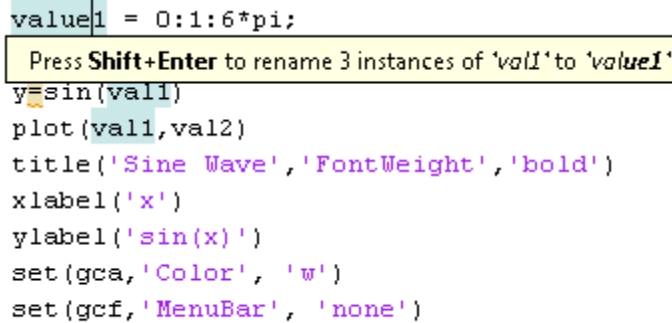
Automatically Rename All Functions or Variables in a File

To help prevent typographical errors, MATLAB provides a feature that helps rename multiple references to a function or variable within a file when you manually change any of the following:

Function or Variable Renamed	Example
Function name in a function declaration	Rename foo in: function foo(m)
Input or output variable name in a function declaration	Rename y or m in: function y = foo(m)
Variable name on the left side of assignment statement	Rename y in: y = 1

As you rename such a function or variable, a tooltip opens if there is more than one reference to that variable or function in the file. The tooltip indicates that MATLAB will rename all instances of the function or variable in the file when you press **Shift + Enter**.


```
value1 = 0:1:6*pi;
y=sin(value1)
plot(value1,value2)
title('Sine Wave','FontWeight','bold')
xlabel('x')
ylabel('sin(x)')
set(gca,'Color','w')
set(gcf,'MenuBar','none')
```



Typically, multiple references to a function appear when you use nested functions or subfunctions.

Note MATLAB does *not* prompt you when you change:

- The name of a global variable.
 - The function input and output arguments, `varargin` and `varargout`.
-

To undo automatic name changes, click the **Undo** button  once.

By default, this feature is enabled. To disable it:

- 1** Select **File > Preferences > Editor/Debugger > Language**.
- 2** In the **Language** field, select **MATLAB**.
- 3** Clear **Enable automatic variable and function renaming**.

Find and Replace Any Text

To search for, and optionally replace specified text within a file, select **Edit > Find and Replace** to open and use the Find & Replace dialog box.

Find Text in Multiple File Names or Files

To find folders and file names that include specified text, or whose contents contain specified text, select **Edit > Find Files**. For details, see “Finding Files and Folders” on page 6-28.

Function Alternative for Finding Text

Use `lookfor` to search for the specified text in the first line of help for all files with the `.m` extension on the search path.

Perform an Incremental Search in the Editor

When you perform an incremental search, 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. In the Editor, incremental search uses the same controls as incremental search in the Command Window. For details, see “Use Incremental Search in the Command Window” on page 3-54.

Run MATLAB Files in the Editor

In this section...

“Run Files with No Input Arguments in the Editor” on page 8-46

“Run Files with Input Arguments in the Editor” on page 8-46

“Create and Use a Run Configuration” on page 8-47

“Use a Previously Created Run Configuration” on page 8-48

“Create and Execute Multiple Run Configurations for a File” on page 8-49

“About the run_configurations.m File” on page 8-52

“Find Configurations” on page 8-53

“Remove Configurations” on page 8-55

“Reassociate and Rename Configurations” on page 8-56

Note See also, “Evaluate Subsections of Files Using Code Cells” on page 8-58

Run Files with No Input Arguments in the Editor

To run a MATLAB script or function file that is open in the Editor and requires no input argument values, do one of the following:

- Click the Run button  on the toolbar.

The button’s tooltip includes the name of the file that will run, which is useful when you have multiple files open. If the file has unsaved changes, the Editor automatically saves the changes before running it.

- Select **Debug > Run *filename***.
- Select **Debug > Save File and Run *filename***.

Run Files with Input Arguments in the Editor

To provide values for MATLAB function input arguments, create and use a run configuration. You can associate multiple run configurations with a function 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 8-52.)

You can also use run configurations to provide preparatory or setup information before running a MATLAB file.

Note Run configurations use the base MATLAB workspace. Therefore, a value that you assign to a variable in a run configuration overwrites the value for that variable (if a value exists) in the base workspace.

Create and Use a Run Configuration

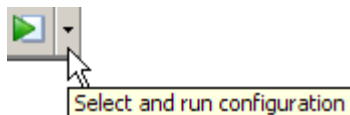
This section demonstrates how to create and use a run configuration by stepping you through an example. The steps specify Editor toolbar buttons, but you can also use equivalent options in the **Debug** menu.

- 1 Open `collatzplot_new.m`:

```
edit(fullfile(matlabroot, 'help', 'techdoc', ...
    'matlab_env', 'examples', 'collatzplot_new.m'))
```

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`.

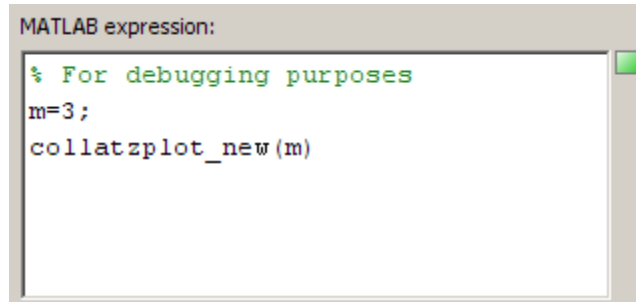
- 2 Click the down arrow on the Run button in the Editor toolbar,



and then select **Edit Run Configurations for `collatzplot_new.m`**.

- 3 In the **MATLAB expression** area of the Edit Configurations dialog box, replace the existing comment and text with:

```
% For debugging purposes
m=3;
collatzplot_new(m)
```

A screenshot of a MATLAB expression editor window. The window has a title bar that says "MATLAB expression:". Inside the window, there is a text area containing the following code:

```
% For debugging purposes  
m=3;  
collatzplot_new(m)
```

The code is displayed in a monospaced font with syntax highlighting: the comment line is green, the variable assignment is black, and the function call is black. There is a small green square button in the top right corner of the text area.

4 Click **Run** to execute the statements in the **MATLAB expression** field.

`collatzplot_new(3)` runs, and a Figure window displays the plot.


5 Change `m = 3` to `m = 3`, and then click **Run**.

You can also modify the code file, save the changes, and click **Run**.

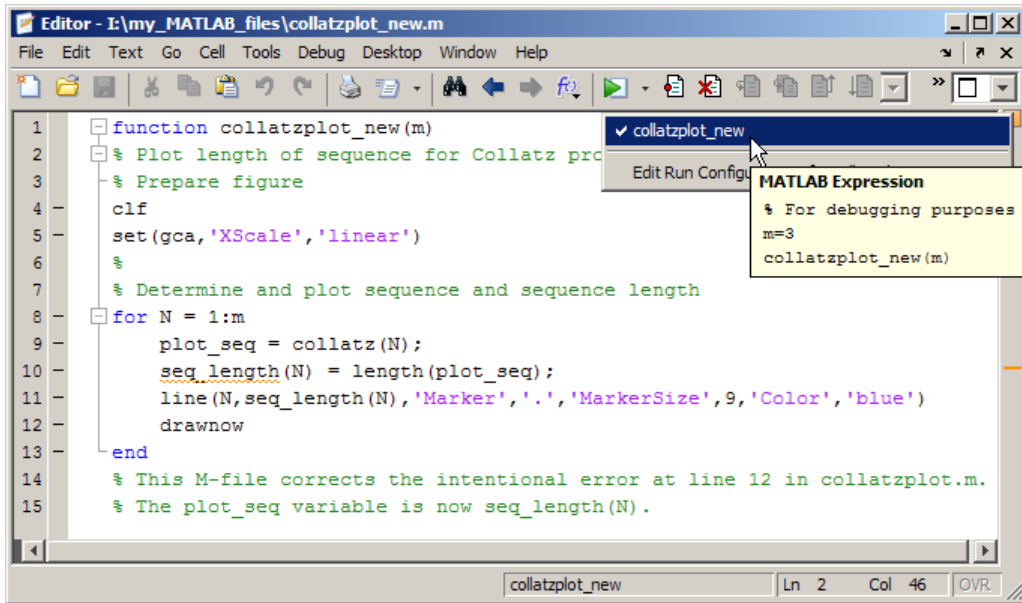
6 Click **Close**.

Use a Previously Created Run Configuration

After creating a run configuration, you can view and use the configuration without opening the Edit Configurations dialog box:

1 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 **MATLAB Expression** associated with the run configuration.





2 Select a run configuration name.

MATLAB runs the configuration.

Create and Execute Multiple Run Configurations for a File

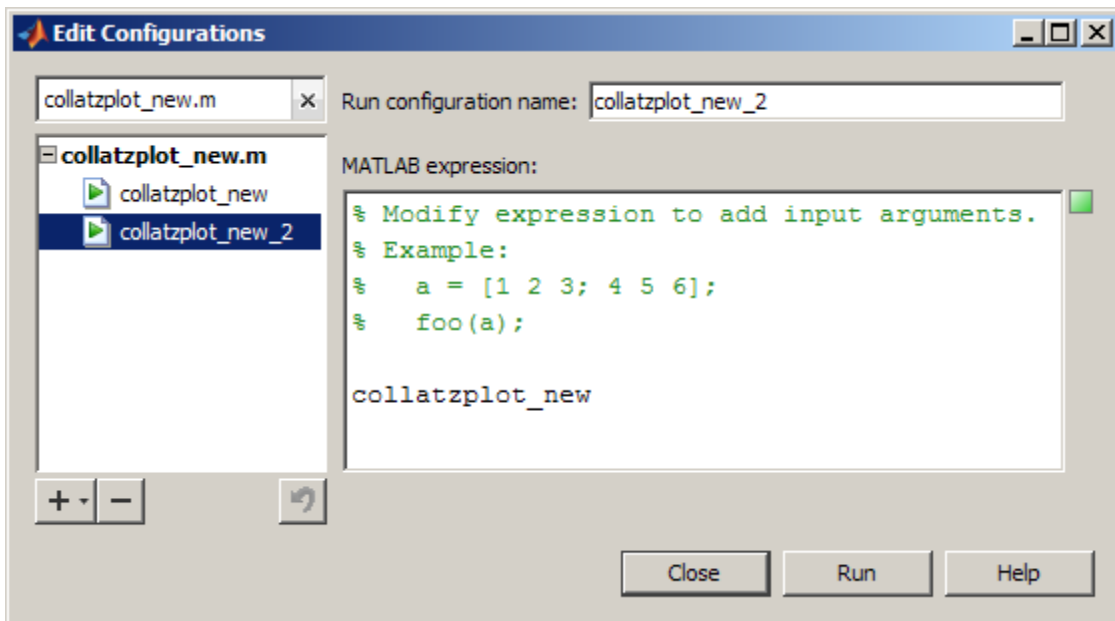
You can create multiple run configurations for a given file, allowing you to run with different values for input arguments, each for a different purpose. Create a named run configuration for each purpose, all associated with the same file. Then, any time you open that file, choose and execute the run 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 In the Editor toolbar, click the down arrow on the Run button , and then select **Edit run configuration for collatzplot_new**:
 - a Select the file to which you want to add a run configuration, or select a configuration already associated with that file.
 - b Click the Add button  (under the list of configurations), and then click **Run Configuration**.

MATLAB creates a new default run configuration template, in this example, `collatzplot_new_2`.

The example shows `collatzplot_new_2` and its default expression, as well as one previously created run configuration associated with `collatzplot_new.m`, `collatzplot_new`.



- 2 In the Edit Configurations dialog box, modify, run, and name the new run configurations as you did for the initial run configuration, `collatzplot_new`, as described in “Create and Use a Run Configuration” on page 8-47.

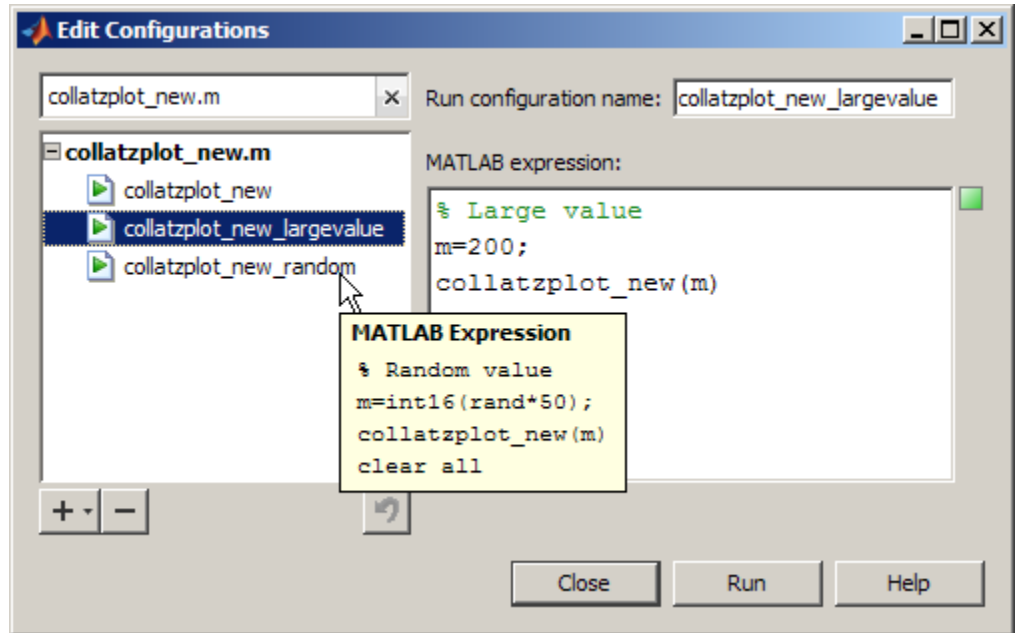
For example, rename `collatzplot_new_2` to `collatzplot_new_largevalue`, and replace the default template expression with:

```
% Large value
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.
- 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_2` and you can see the statements in it.



- 5 To close the Edit Configurations dialog box, click **Close**. MATLAB saves the configurations and their associations with the file in `run_configurations.m` in your preferences folder.

For more information, see “About the `run_configurations.m` File” on page 8-52.

About the `run_configurations.m` File

When you create one or more run configurations using the Edit Configurations dialog box, the Editor creates or updates the `run_configurations.m` file in your preferences folder. (MATLAB returns the preferences folder when you run `prefdir`.) The `run_configurations.m` file is a text file that you can view and use to evaluate MATLAB code 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, only do this if you have not already created configurations on the second system. Also, because this file might contain references to file paths, ensure that the MATLAB file and paths it specifies exist on the second system.

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 Configurations dialog box might be overwritten.

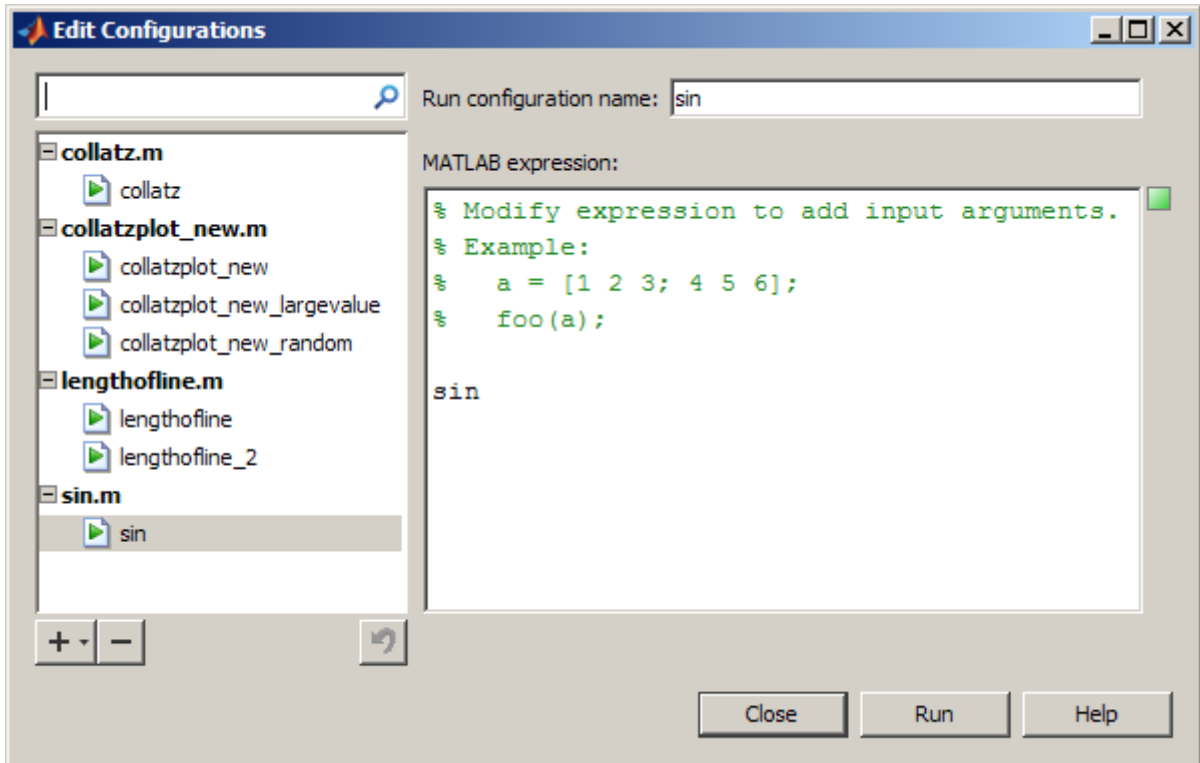
Each time you change a run configuration using the Edit Configurations dialog box, MATLAB updates the `run_configurations.m` file as well as the `publish_configurations.m` file. See “About the `publish_configurations.m` File” on page 10-107 for more information about that file.

Find Configurations

Follow these steps to find run or publish configurations. (For information on publish configurations, see “Specify Output Preferences for Publishing” on page 10-66.)

- 1 Open any MATLAB code file in the Editor.
- 2 In the Editor toolbar, click the down arrow on the Run button , and then select **Edit Run Configurations** for any file name.
- 3 In the Edit Configurations dialog box, click the Clear search button within the search field  to clear the search field.

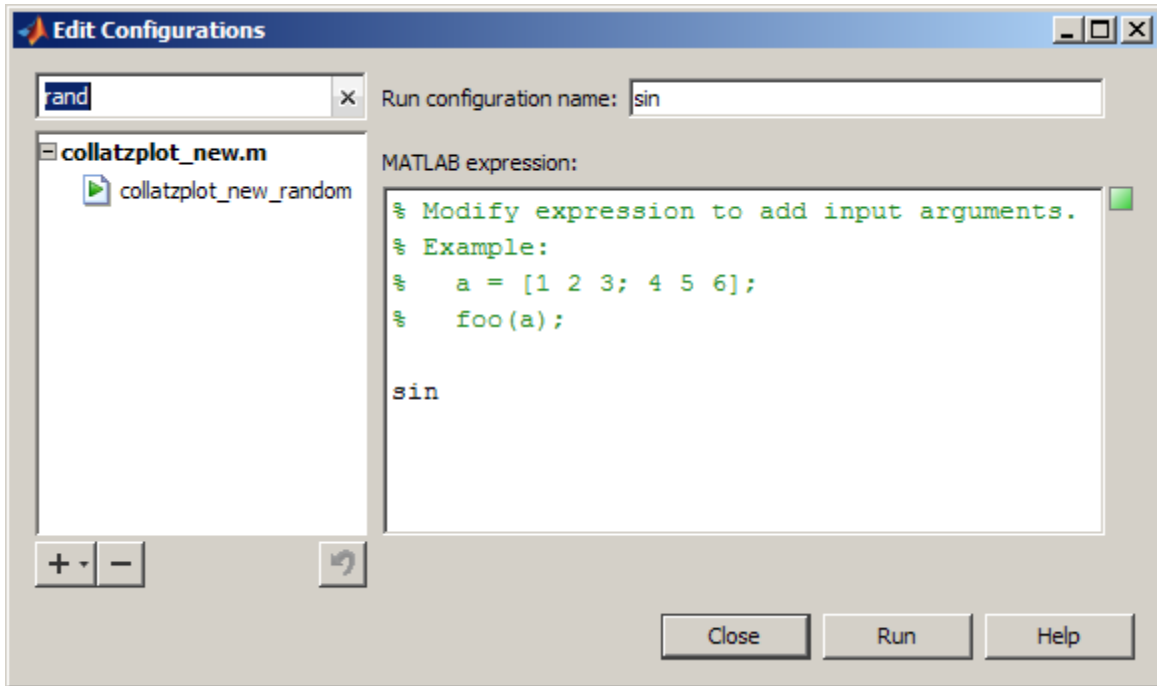
In the left pane, MATLAB lists all files with configurations.



- 4 Type a term in the search field to find a file or configuration by name.

MATLAB displays only those files whose names contain the term, or whose associated configurations contain the term in their name. As you type, MATLAB filters out files any configurations that do not contain the term.


For example, type `rand`. In this example, only one file, `collatzplot_new.m`, has a configuration that contains the term `rand`.






- 5 View the expression in that configuration by positioning the mouse pointer over the name.
- 6 As you type additional letters in the search field, fewer files remain in the list of results. Use the **Backspace** key to modify the term. If there are no 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 file with which it is associated, consider deleting the configuration. (For information on publish configurations, see “Specify Output Preferences for Publishing” on page 10-66.)

- 1 Open any MATLAB code file in the Editor.
- 2 In the Editor toolbar, click the down arrow on the Run button , and then select **Edit Run Configurations** for any file name.




- 3 In the Edit Configurations dialog box, click the Clear search button within the search field  to clear the search field.
- 4 In the Edit Configurations dialog box, do one of the following in the pane on the left:
 - To remove a single configuration, select that configuration.
 - To remove all the run and publish configurations for the file, select the file.
- 5 Click the Remove button .
- 6 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 file. If you move or rename a file that has configurations, redefine the association. If you delete a file, consider deleting the associated configurations, or associate them with a different file.

When MATLAB cannot associate a configuration with a file, the Edit Configurations dialog box displays the file name in red, displays a **File Not Found** message, and enables you to find the file to which you want to associate the configuration.

To reassociate a configuration:

- 1 In the Editor toolbar, click the down arrow on the Run button  , and then select **Edit Run Configurations** for any file name.
- 2 In the Edit Configurations dialog box, click the Clear search button within the search field  to clear the search field.
- 3 In the left pane, select the file for which you want to reassociate configurations, and then click **Choose**.
- 4 In the resulting Open dialog box, navigate to and select the file with which you now want to reassociate the configurations. Click **Open**.

In the Edit Configurations dialog box, the **Associated file** value reflects the change you made and the **File Not Found** message no longer appears.

5 Rename the configurations to be consistent with the new file name

Select a configuration from the list in the left pane. In the right pane, edit the value for the configuration name. Repeat this step for all run and publish configurations associated with the file.

6 For a file name change, you might need to modify the configuration statements to run correctly. For example, change a function call to reflect the new file name for that function.

Evaluate Subsections of Files Using Code Cells

In this section...

- “What Are Code Cells?” on page 8-58
- “Scenarios for Evaluating Sections of Code” on page 8-59
- “Process for Evaluating Sections of Files” on page 8-60
- “Define Code Cells” on page 8-61
- “Nested Code Cells” on page 8-68
- “Navigate Among Code Cells in a File” on page 8-76
- “Evaluate Code Cells” on page 8-77

What Are Code Cells?

MATLAB files often have a natural structure consisting of multiple sections. In large 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 files to others, often you describe the sections of the code. To facilitate these processes, use *code cells*. A code cell contains the contiguous lines of code that you want to evaluate as a whole in a MATLAB script. A code cell has boundaries to define its start and end. Because code cell features operate on code cells, it is important to understand how you define boundaries explicitly, how MATLAB defines boundaries implicitly, and how implicitly and explicitly defined code cell boundaries interact to create code cells, as described in “Define Code Cells” on page 8-61

Specifically, MATLAB software uses code cells for:

- Evaluating sections of code in the Editor — This makes the experimental phase of your work with MATLAB scripts easier. This process is sometimes referred to as rapid code iteration. For details, see “Scenarios for Evaluating Sections of Code” on page 8-59.
- Publishing MATLAB files — This enables you to include code and results in a presentation format such as HTML. For more information, see “Overview of Publishing MATLAB Code” on page 10-2.

MATLAB Code Cells in Files

Denote MATLAB code cells with two comment characters (%%) at the start of a line and save the file using `.m` as the extension. If you open a file that contains MATLAB code cells when cell mode is enabled, then highlighting and horizontal lines appear in the file.

If you do not want cell mode enabled, select **Cell > Disable Cell Mode**.

The first time you open a file that contains cells, an information bar appears below the cell toolbar, providing links for details about cell mode.

Scenarios for Evaluating Sections of Code

When working with a MATLAB file, you often experiment with your code—modifying it, testing it, and updating it—until you have a file that does what you want. For example:

- Suppose you have code that plots data. You might break the code into two code cells: the code in the first cell creates the basic results, while the code in the second cell labels the plot. The two code cells enable 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 Code Cells” on page 8-80.
- Suppose you have two images that you want to add together, and then display the result. 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 those 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 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.

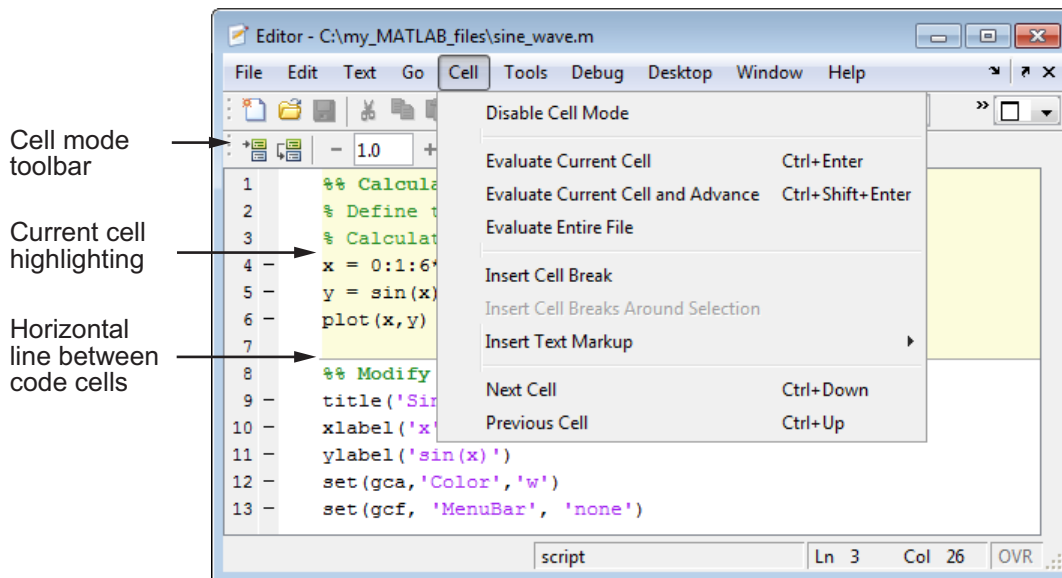
Use the MATLAB code cell features with MATLAB scripts to facilitate this process. You also can use code cell features with MATLAB function files, but there are some restrictions—see “Code Cells in MATLAB Function Files” on page 8-79.

Note Cell mode is supported for use with MATLAB code files (.m files) only. It is not for use with plain text files. When used with plain text files, results are unpredictable.

Process for Evaluating Sections of Files

This is the overall process of using code cells to evaluate sections of code:

- 1 In the MATLAB Editor, select **Cell > Enable Cell Mode**.
- 2 Define the boundaries of the cells in a MATLAB script using cell features. Code cells are denoted by a specialized comment syntax, `%%`. For details, see “Define Code Cells” on page 8-61.
- 3 After you define the code cells, use cell features. Cell features enable you 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 affect the result. For details, see “Evaluate Code Cells” on page 8-77.



Define Code Cells

You define code 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, if there is white space between the %% characters and the text. For details, see “Define Code Cell Boundaries Explicitly” on page 8-62.

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. 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 or while statement), 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 “Nested Code Cells” on page 8-68.

This section includes the following topics:

- “Define Code Cell Boundaries Explicitly” on page 8-62
- “Create Titles for Code Cells” on page 8-62
- “Highlight Code Cells” on page 8-63
- “Example of Defining Code Cells” on page 8-63


- “Fix Code Cell Highlighting Problems” on page 8-65
- “Remove Code Cells” on page 8-67
- “Summary of Cell Mode and Code Cell Requirements” on page 8-67

Define Code Cell Boundaries Explicitly

To define code cell boundaries explicitly, insert cell breaks:

1 Select **Cell > Enable Cell Mode**.

2 Do one of the following to insert 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**.

If cell breaks do not appear in your Editor document, check preferences for **Cell display options** by selecting **File > Preferences > Colors > Programming Tools**.

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.

Create Titles for Code 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.

Highlight Code 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*. The current cell is used, for example, when you select the **Evaluate Current Cell** option from the **Cell** menu.

Turn Off Code Cell Highlighting.

- 1 Select **File > Preferences > Colors > Programming Tools**.
- 2 Under **Cell display options**, clear **Highlight cells**.

Set a Color for Code Cell Highlighting.

- 1 Select **File > Preferences > Colors > Programming Tools**.
- 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 Code Cells

This example defines two cells for a file called `sine_wave.m`, shown in the figure that follows. To open this code in your Editor, run the following command and then save the file to a local folder:

```
edit(fullfile(matlabroot, 'help', 'techdoc', ...  
'matlab_env', 'examples', 'sine_wave.m'))
```

The steps that follow insert a cell break 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 enable 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.

- 1 Select **Cell > Enable Cell Mode**.

When cell mode is enabled, the **Cell** menu displays **Disable Cell Mode**.

2 Select **File > Preferences > Colors > Programming Tools**, and then select **Highlight cells** and **Show lines between cells**.

3 Position the cursor at the start of the first line. Select **Cell > Insert Cell Break**.

The Editor inserts a cell break `%%` 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 that you have that display preference for cells selected.

4 After the cell break, type a space, and then enter a cell title.

```
%% Calculate and Plot Sine Wave
```

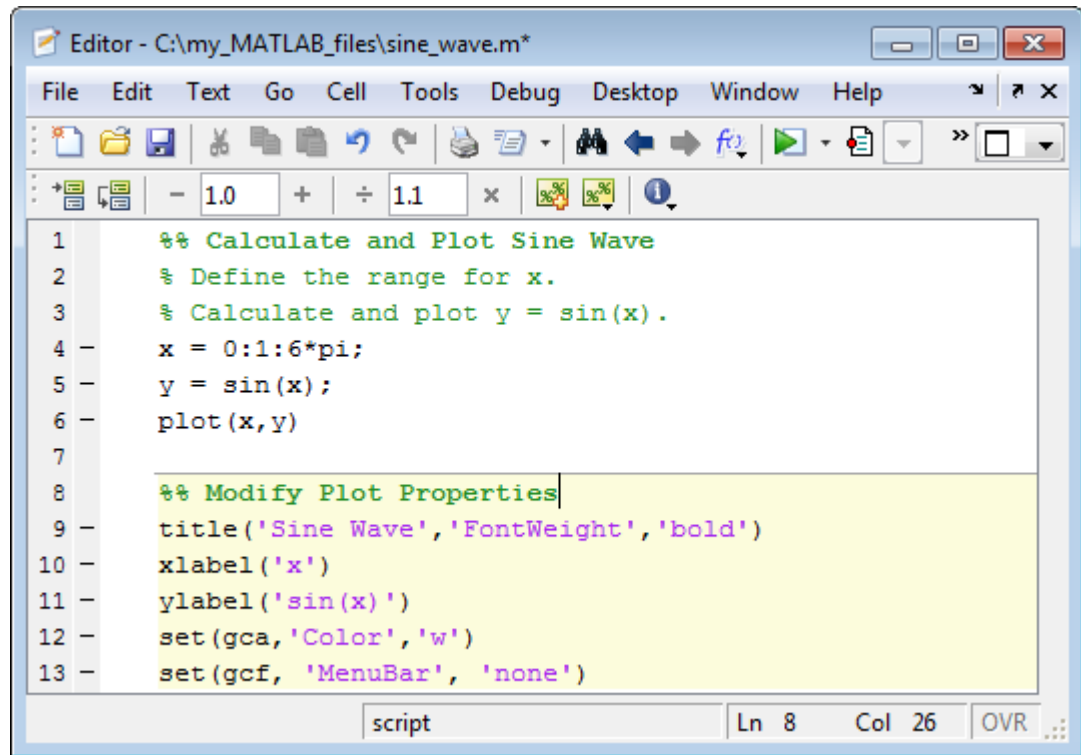
5 Position the cursor at the start of line 7, `title...`, and then select **Cell > Insert Cell Break**.

The Editor inserts a line containing only a cell break (`%%`) at line 7 and moves the remaining lines down one line. A horizontal line that helps you distinguish the two cells appears above the cell break line. Lines 7 through 12 appear highlighted in yellow, indicating they comprise the current cell.

6 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.



```
1 %% Calculate and Plot Sine Wave
2 % Define the range for x.
3 % Calculate and plot y = sin(x).
4 x = 0:1:6*pi;
5 y = sin(x);
6 plot(x,y)
7
8 %% Modify Plot Properties
9 title('Sine Wave','FontWeight','bold')
10 xlabel('x')
11 ylabel('sin(x)')
12 set(gca,'Color','w')
13 set(gcf,'MenuBar','none')
```

Fix Code Cell Highlighting Problems

If you introduce an error into a file, such as a syntax error, cell highlighting and dividers might 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 Code Cells” on page 8-63. Make sure that automatic code analysis is enabled by selecting **File > Preferences > Code Analyzer**, and then selecting **Enable integrated warning and error messages**.

- 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 **Select File > Preferences > Colors > Programming Tools**, and then making choices for **Cell Display options**.
- If code analysis finds errors in a file, cell dividers and highlighting might not appear as you expect. For details, see “Fix Code Cell Highlighting Problems” on page 8-65.

For more information, see “Avoid Mistakes While Editing Code” on page 8-84.

Nested Code 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 a file:

- “File Without Explicit Code Cell Breaks” on page 8-68
- “How Nesting Code Cell Breaks Result in Cells” on page 8-69
- “Example File with Nested Code Cell Breaks” on page 8-69
- “Associate Code Cell Breaks with Subfunctions” on page 8-74

File Without Explicit Code Cell Breaks

The following code when viewed in the Editor displays no cells or highlighting. It is a single, implicit code 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

```

To follow this example, save the code to a local folder with the file name `fourier.m`.

How Nesting Code 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 8

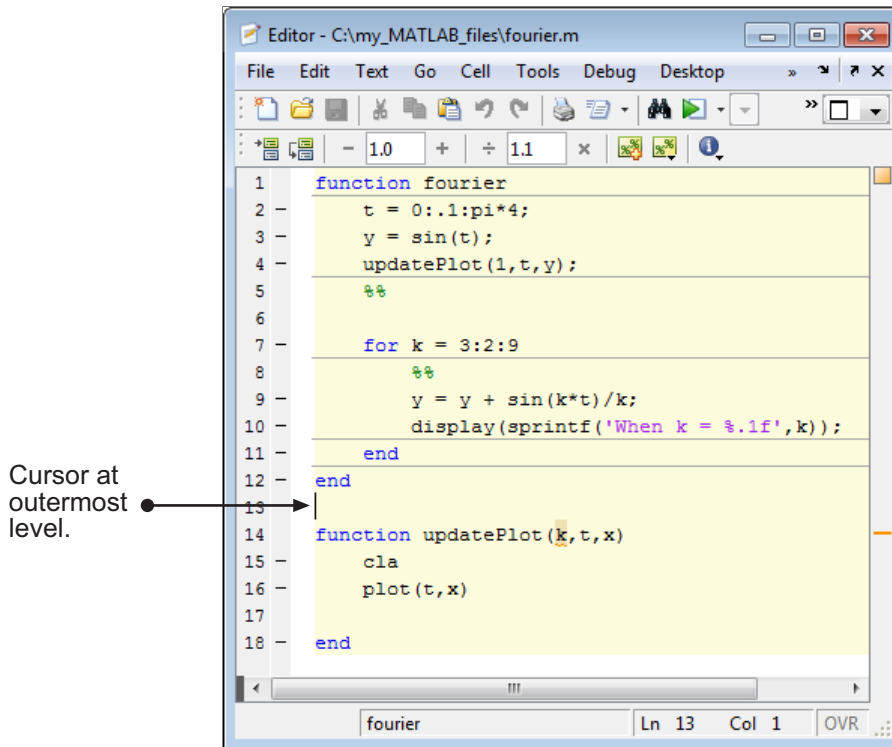
This results in the following cells, which are illustrated in “Example File with Nested Code Cell Breaks” on page 8-69:

- 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.

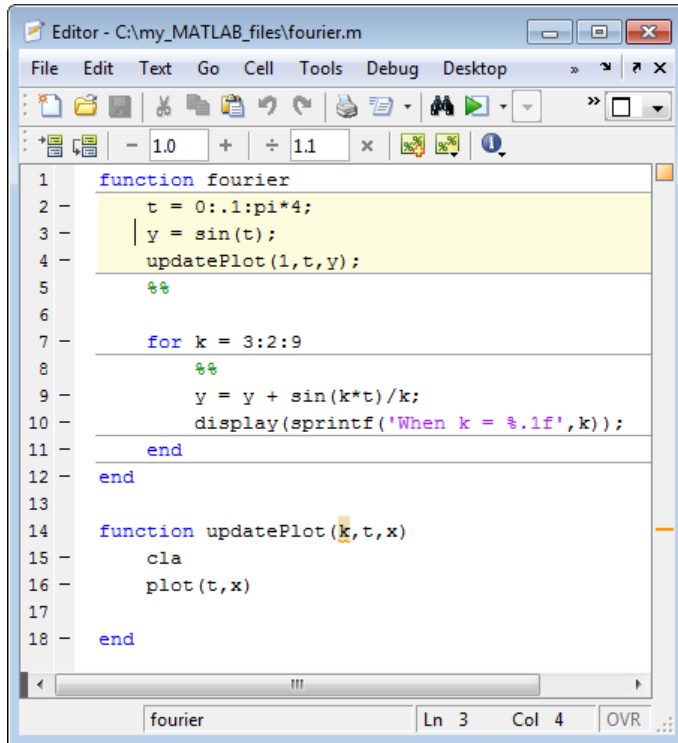
Example File with Nested Code Cell Breaks

The following images illustrate how inserting explicit cell breaks, as described in “How Nesting Code Cell Breaks Result in Cells” on page 8-69, affect the appearance of the 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.



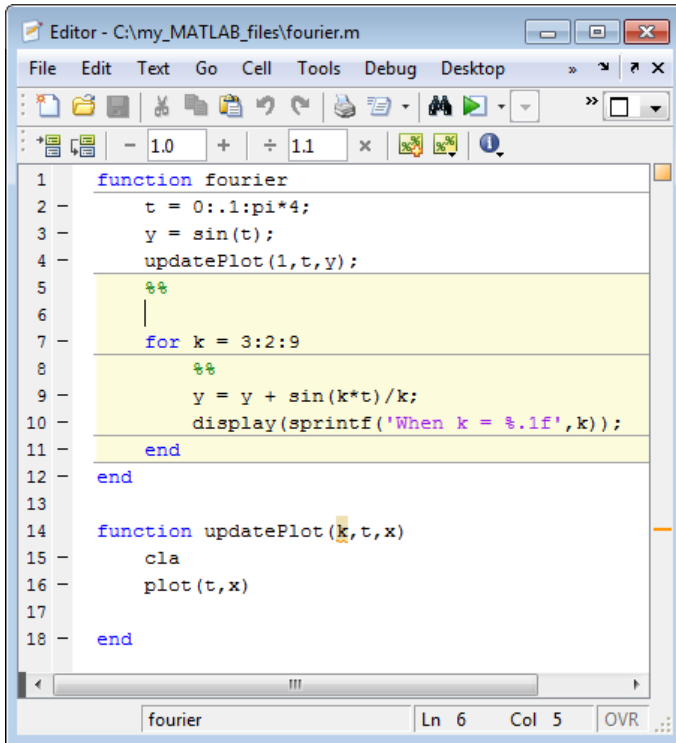
MATLAB only defines implicit cell breaks in a code block if you specify an explicit cell break within that code block. Therefore, because function `updatePlot` in this example has no explicit (and therefore, no implicit) cell breaks defined for it, when you place the cursor within that function, MATLAB considers the cursor to be within the cell that encloses the whole file.



The screenshot shows the MATLAB Editor window with the following code:

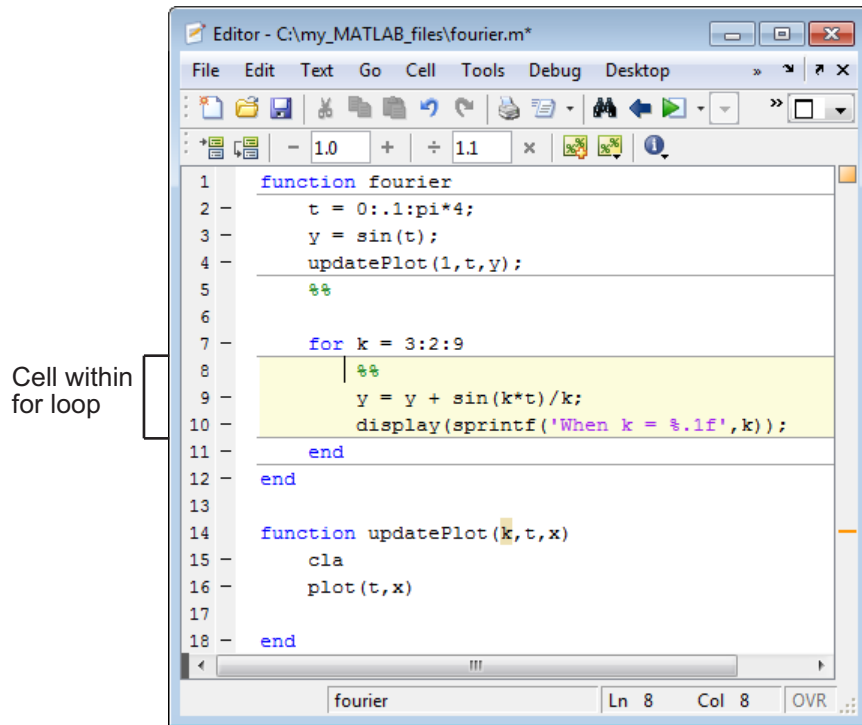
```
1 function fourier
2     t = 0:.1:pi*4;
3     y = sin(t);
4     updatePlot(1,t,y);
5     %%
6
7     for k = 3:2:9
8         %%
9         y = y + sin(k*t)/k;
10        display(sprintf('When k = %.1f',k));
11    end
12 end
13
14 function updatePlot(k,t,x)
15     cla
16     plot(t,x)
17
18 end
```

The status bar at the bottom indicates the file name is 'fourier', the cursor is at line 3, column 4, and the view is 'OVR'.



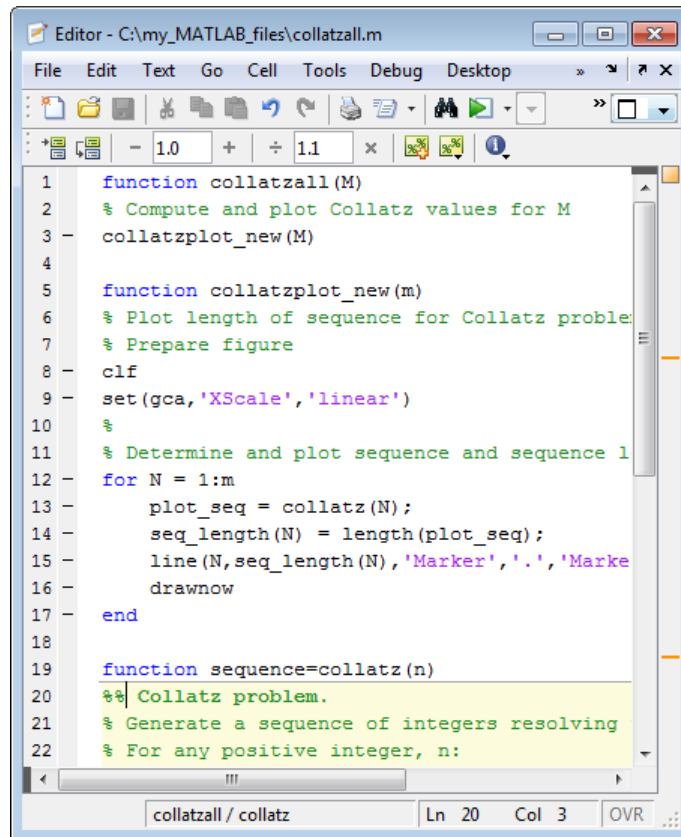
```
1 function fourier
2     t = 0:.1:pi*4;
3     y = sin(t);
4     updatePlot(1,t,y);
5     %%
6     |
7     for k = 3:2:9
8         %%
9         y = y + sin(k*t)/k;
10        display(sprintf('When k = %.1f',k));
11    end
12 end
13
14 function updatePlot(k,t,x)
15     cla
16     plot(t,x)
17
18 end
```

- **Third level of nesting** — When you place the cursor within the for loop, the cell within this loop is highlighted.



Associate Code Cell Breaks with Subfunctions

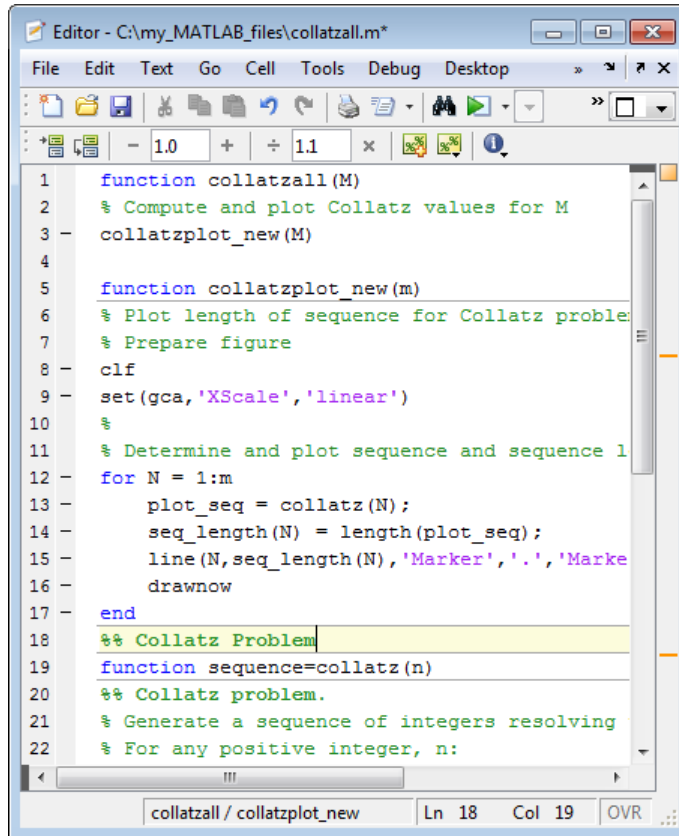
If you want a cell break to be associated with a subfunction, place the cell break within the subfunction, rather than above the subfunction declaration. Otherwise, the cell break creates a single cell within the code block that precedes the subfunction. The following two images demonstrate the difference.



The image shows a MATLAB Editor window titled "Editor - C:\my_MATLAB_files\collatzall.m". The window contains a script with the following code:

```
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 length
12 for N = 1:m
13     plot_seq = collatz(N);
14     seq_length(N) = length(plot_seq);
15     line(N,seq_length(N),'Marker','.', 'MarkerSize',10)
16     drawnow
17 end
18
19 function sequence=collatz(n)
20 %% Collatz problem.
21 % Generate a sequence of integers resolving
22 % For any positive integer, n:
```

The status bar at the bottom of the window indicates "collatzall / collatz", "Ln 20", "Col 3", and "OVR".






```

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 length
12 for N = 1:m
13     plot_seq = collatz(N);
14     seq_length(N) = length(plot_seq);
15     line(N,seq_length(N),'Marker','.', 'MarkerSize',10);
16     drawnow
17 end
18 %% Collatz Problem
19 function sequence=collatz(n)
20 %% Collatz problem.
21 % Generate a sequence of integers resolving
22 % For any positive integer, n:

```

Navigate Among Code Cells in a File

You can navigate among cells in a file without evaluating the code within those cells, as described in the table that follows. This can be useful when you want to jump quickly from cell to cell within a file. You might do this, for example, to find the point at which you want to begin cell evaluation. By default, the cell mode navigation buttons are not on the Editor Cell Mode toolbar. For information on how to add them, see “Toolbar Customization” on page 2-11.

Operation	Instructions
Move to the next cell.	Select Cell > Next Cell or click the Next Cell button  .
Move to the previous cell.	Select Cell > Previous Cell or click the Previous Cell button  .
Move to a specific cell.	<p>Do either of the following:</p> <ul style="list-style-type: none"> • Use the Editor Cell Mode toolbar: <ol style="list-style-type: none"> 1 Click the Show cell titles button . 2 Select the cell title to which you want to move. • Use the Go menu: <ol style="list-style-type: none"> 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.

Evaluate Code Cells

As you develop a MATLAB file, you can use the Editor cell features to evaluate the 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:




- “Evaluate Code Cells in a File” on page 8-78
- “Processing Considerations When Evaluating Code Cells” on page 8-78
- “Modify Values in a Code Cell” on page 8-79
- “Example of Evaluating Code Cells” on page 8-80

Evaluate Code Cells in a 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.

The following table provides instructions on evaluating code cells.

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 Cell Mode toolbar. See “Toolbar Customization” on page 2-11 for information on how to add it.
	<hr/> <p>Note A beep indicates there is an error. See the Command Window for the error message.</p> <hr/>

Processing Considerations When Evaluating Code Cells

This section describes processing considerations to take into account when you evaluate code cells in MATLAB files.

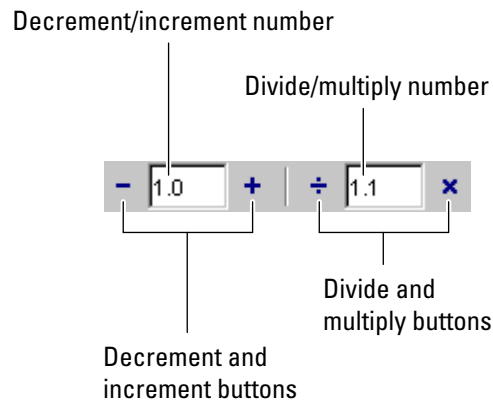
Set Breakpoints. While you can set breakpoints and debug a file containing cells, when you evaluate a file from the **Cell** menu or cell toolbar, breakpoints are ignored. To run the file and stop at breakpoints, use **Run/Continue** in the **Debug** menu. This means you cannot debug while running a single cell.

Code Cells in MATLAB Function Files. You can define and evaluate cells in MATLAB function files as long as the variables referenced in the code 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.

Modify Values in a Code Cell

You can use code cell features to modify numbers in a cell, which also automatically reevaluates the cell. This helps you experiment with and fine-tunes 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.



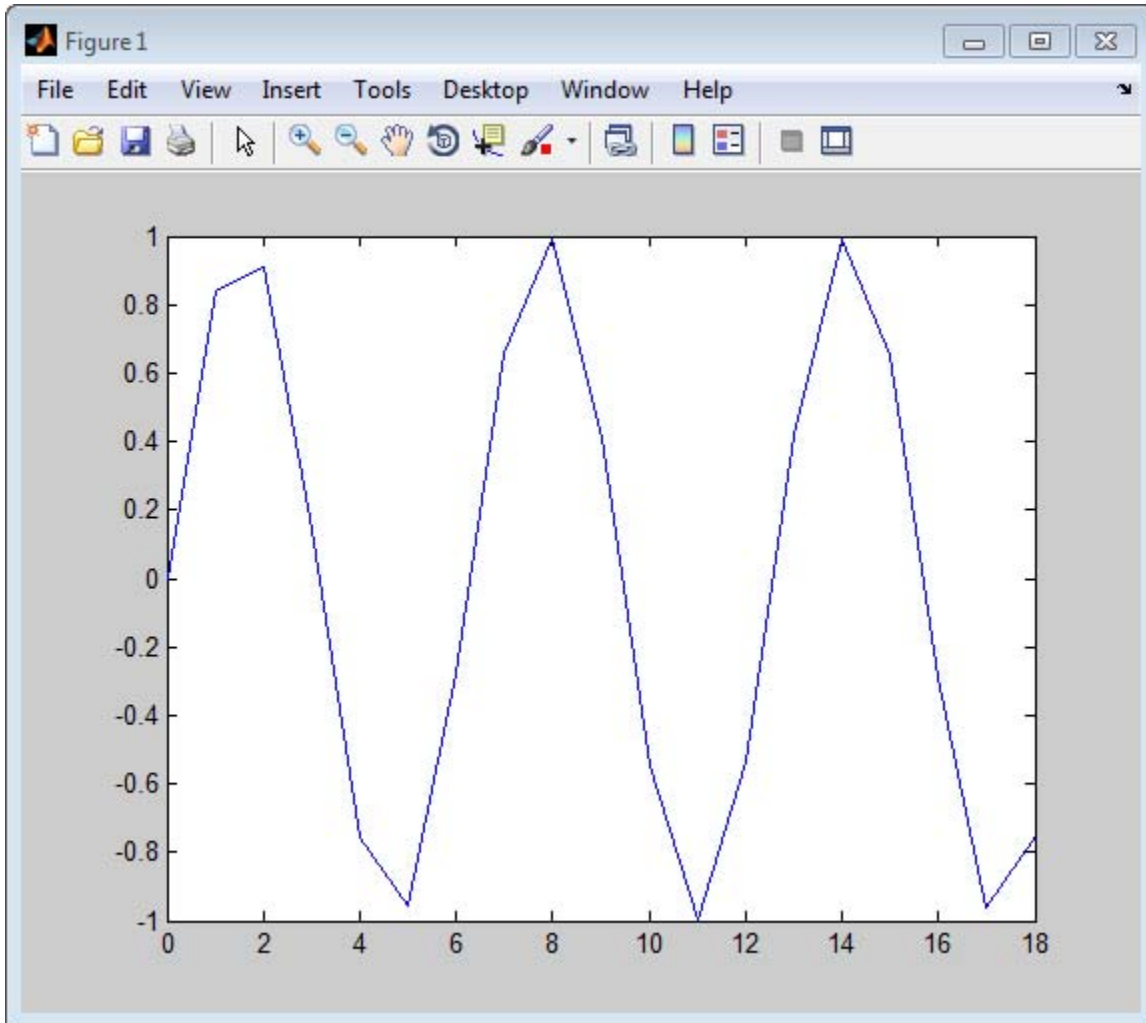
You can use the numeric keypad operator keys (-, +, /, and *) instead of the operator buttons on the toolbar.

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 Code 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.



- 2** Assume that 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 \div .

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;
```

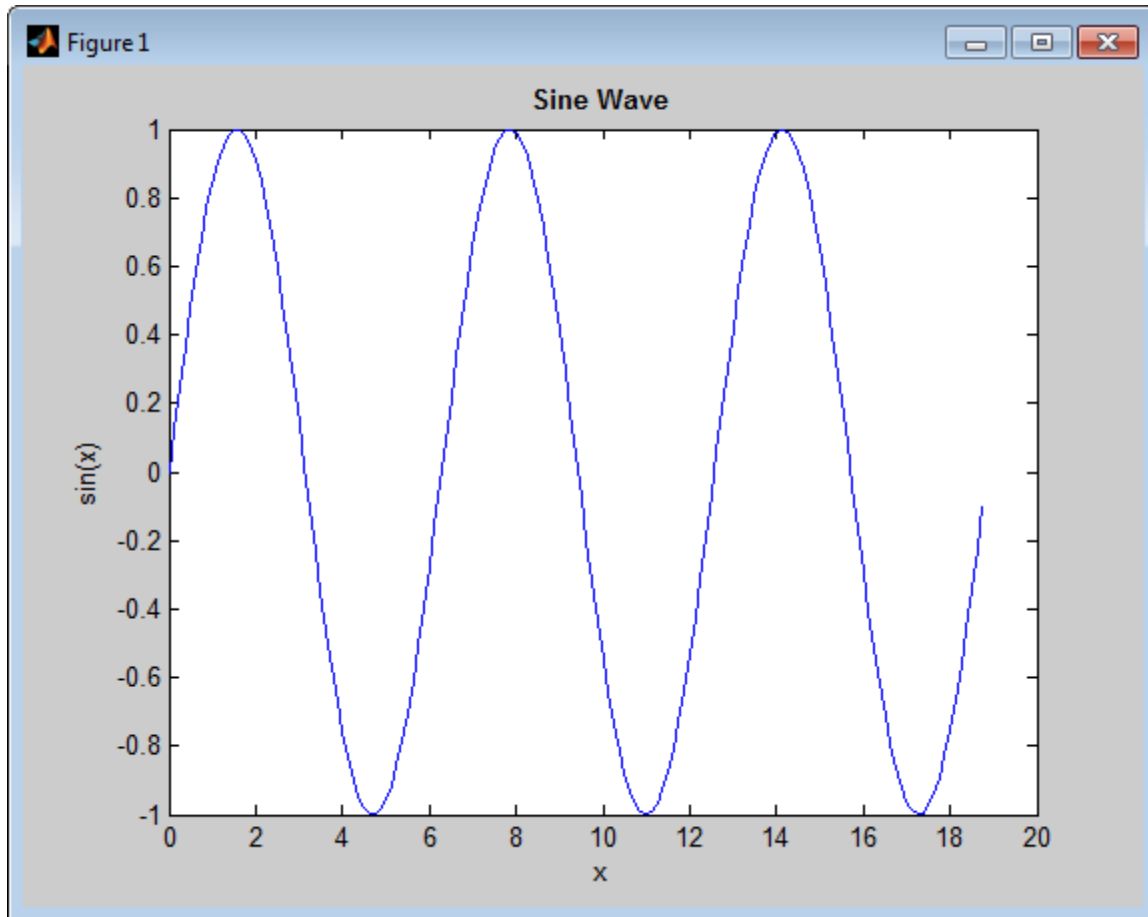
The curve is smooth, but because there are more values, processing time is slower. It would be better to find a smaller x that still produces a smooth curve.

- 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 select **Cell > Evaluate Current Cell** to evaluate the current cell.

MATLAB updates the figure.



Avoid Mistakes While Editing Code

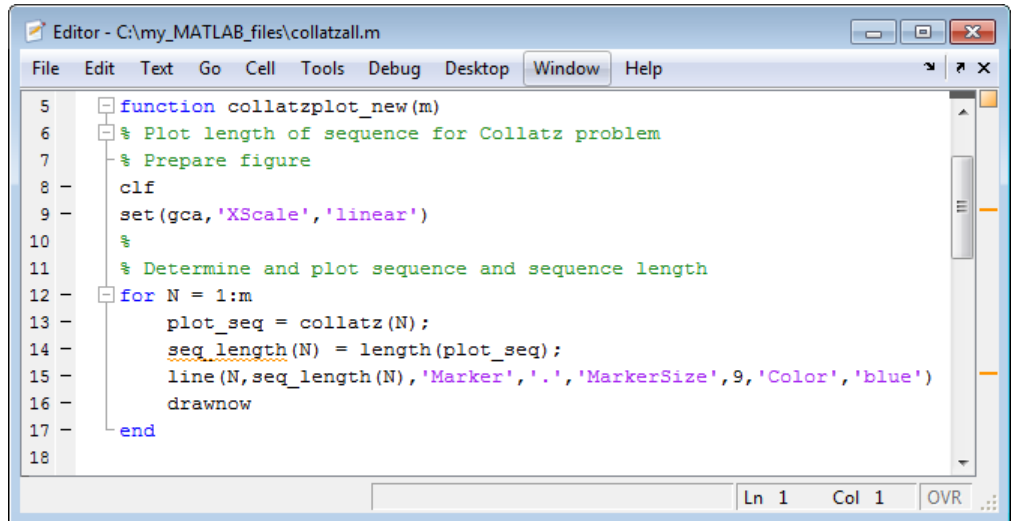
In this section...
“Highlight Syntax to Help Ensure Correct Entries in the Editor” on page 8-84
“Complete Names in the Editor Using the Tab Key” on page 8-85
“Check Code for Errors and Warnings” on page 8-91
“Avoid Variable and Function Scoping Problems” on page 8-110

Note See also:

- “Avoid Mismatched Parentheses, Brackets, Braces, and Paired Keywords” on page 3-42
 - “View Function Syntax Hints While Entering a Statement” on page 3-49
-

Highlight Syntax to Help Ensure Correct Entries in the Editor

By default, some language elements 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.



```
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 length
12 for N = 1:m
13     plot_seq = collatz(N);
14     seq_length(N) = length(plot_seq);
15     line(N,seq_length(N),'Marker','.', 'MarkerSize',9,'Color','blue')
16     drawnow
17 end
18
```

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.

To enable, disable, or change syntax highlighting preferences:

- 1 Select **File > Preferences > Editor/Debugger > Languages**.
- 2 From the **Languages** drop-down menu, select the language for which you want to set preferences.
- 3 Change the **Syntax highlighting** options, and then click **OK**.

Complete Names in the Editor Using the Tab Key

MATLAB helps you automatically complete names as you type them in the Editor. Names supported include functions (including subfunctions and nested functions within the active document), models, class and package

names that are on the MATLAB search path or in the current folder, variables (including structures and objects), and Handle Graphics properties.

To complete names in the Editor:

1 Select **File > Preferences > Keyboard**.

2 Select **Enable in Editor/Debugger**.

3 Type the first few characters of the name you want to complete, and then press the **Tab** key.

MATLAB completes the name or offers a list of names from which you can select one.

4 Do one the following, depending on how MATLAB responds. If MATLAB:

- Completes the name you want

You are done.

- Presents a list of possible matches

To select the name you want, use the arrow keys, and then press the **Tab** key. See “Example of Name Completion in the Editor for a Variable” on page 8-87.

To clear the list without selecting anything, press the **Esc** (escape) key.

To narrow a long list before making a selection, add additional characters to your original term.

- Completes the first part of a name that uses dot notation.

In the Editor, type a dot, and then press the **Tab** key. Repeat until the name is complete. “Example of Name Completion in the Editor for a Structure” on page 8-88.

- Completes one property of several

In the Editor, type a comma and the first part of the next property, and then press the **Tab** key. Repeat for all properties you want to specify. See “Example of Name Completion in the Editor for Figure Properties” on page 8-89.

Be aware that MATLAB

- Completes nested functions only when they are available at the current location of the cursor.
- Completes names of variables and properties of figures that are in the current workspace.

The current workspace appears in the **Stack** on the toolbar.

- Completes names of variables defined in the active document. The variable must be valid at the current location of the cursor (that is, already defined).
- Does not complete the field names of structure arrays defined only within the active file.
- Does not complete method or property names for objects defined only within the active file.

For more information, see the following examples:

- “Example of Name Completion in the Editor for a Variable” on page 8-87
- “Example of Name Completion in the Editor for a Structure” on page 8-88
- “Example of Name Completion in the Editor for Figure Properties” on page 8-89
- “Use the Tab Key for Spacing” on page 8-90

Example of Name Completion in the Editor for a Variable

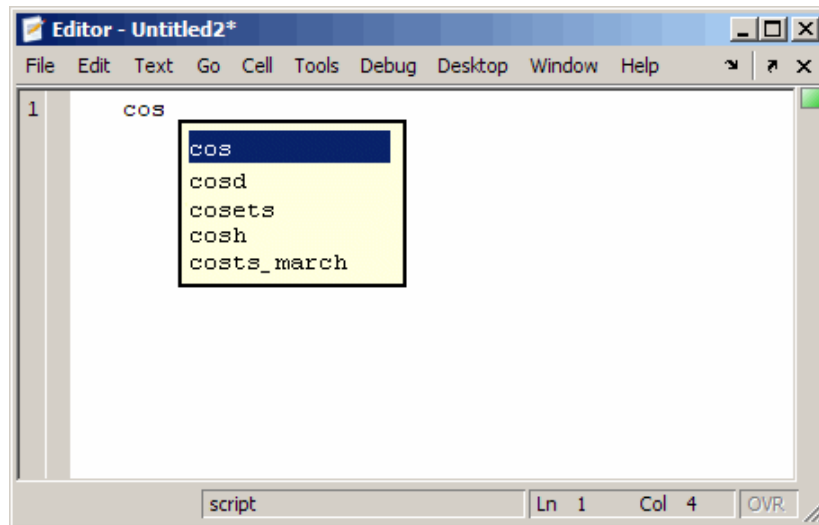
This example shows how to use tab completion for a variable name when there is more than one name that starts with the characters you type. The process is similar for other names that do not use dot notation.

- 1** Select **File > Preferences > Keyboard**.
- 2** Select **Enable in Editor/Debugger**.
- 3** In the Command Window create a variable, `costs_march`, by typing the following, and then pressing **Enter**.

```
costs_march = 100;
```

- 4** In the Editor, type `cos`, and then press the **Tab** key.

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



5 Navigate the list of possible completions using the up and down arrow keys until `costs_march` is selected.

6 Choose `costs_march` by pressing the **Tab** key.

MATLAB completes the name in the Editor.

Example of Name Completion in the Editor for a Structure

This example shows how to use tab completion for a structure that is in the current workspace. The process is similar for other names that use dot notation.

1 Select **File > Preferences > Keyboard**.

2 Select **Enable in Editor/Debugger**.

3 In the Command Window create a structure, `description`, by typing the following, and then pressing **Enter**.

```
description = struct('type', {'big','little'}, 'color', {'red'}, ...
```

```
'x', {3 4})
```

4 In the Editor, type `descrip`, and then press the **Tab** key.

MATLAB completes the name `description`.

5 In the Editor, type a dot, `.`, after `description`, and then press the **Tab** key.

MATLAB displays all fields of `description`: `color`, `type`, and `x`.

6 Navigate the list of possible completions using the up and down arrow keys until `type` is selected, and then press the **Tab** key.

The Editor contains `description.type`.

Example of Name Completion in the Editor for Figure Properties

This example shows how to use tab completion for a figure that is in the current workspace.

1 Select **File > Preferences > Keyboard**.

2 Select **Enable in Editor/Debugger**.

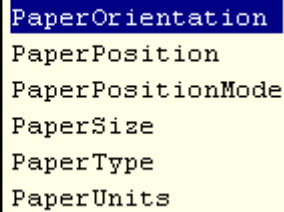
3 In the Command Window create a figure, `myfig`, by typing the following, and then pressing **Enter**.

```
f = figure('Name','Simulation Plot Window','NumberTitle','off')
```

4 In the Editor, type `set(f, 'pap`, and then press the **Tab** key.

The Editor displays

```
set(f, 'paper|
```



- PaperOrientation
- PaperPosition
- PaperPositionMode
- PaperSize
- PaperType
- PaperUnits

- 5 Select the `PaperUnits` property from the list by typing an additional letter, `u`, which makes the string in the Editor unique.

MATLAB completes the property, including the closing quote: `set(f, 'paperunits'`

- 6 Continue adding to the statement by typing `, 'c`, and then press the **Tab** key

MATLAB automatically completes the property, `centimeters`, because it is the only possible completion.

Use the Tab Key for Spacing

If the preference for tab completion is selected, and you want to use the **Tab** key to add spacing within your statements also, 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.

Check Code for Errors and Warnings

MATLAB Code Analyzer can automatically check your code for coding problems, as described in the following sections:

- “Automatically Check Code in the Editor — Code Analyzer” on page 8-91
- “Create a Code Analyzer Message Report” on page 8-99
- “Adjust Code Analyzer Message Indicators and Messages” on page 8-100
- “Understand Code Containing Suppressed Messages” on page 8-104
- “Understand the Limitations of Code Analysis” on page 8-106
- “Enable MATLAB Compiler Deployment Messages” on page 8-109

Automatically Check Code in the Editor – Code Analyzer

You can view warning and error messages about your code, and modify your file based on the messages. The messages update automatically and continuously so you can see if your changes addressed the issues noted in the messages. Some messages offer additional information, automatic code correction, or both.

To use continuous code checking in a MATLAB code file in the Editor:

- 1** Select **File > Preferences > Code Analyzer**, and then select the **Enable integrated warning and error messages** check box.
- 2** Set the **Underlining** option to Underline warnings and errors, and then click **OK**.
- 3** Open a MATLAB code file in the Editor. This example uses the sample file `lengthofline.m` that ships with the MATLAB software:

- a** Open the example file:

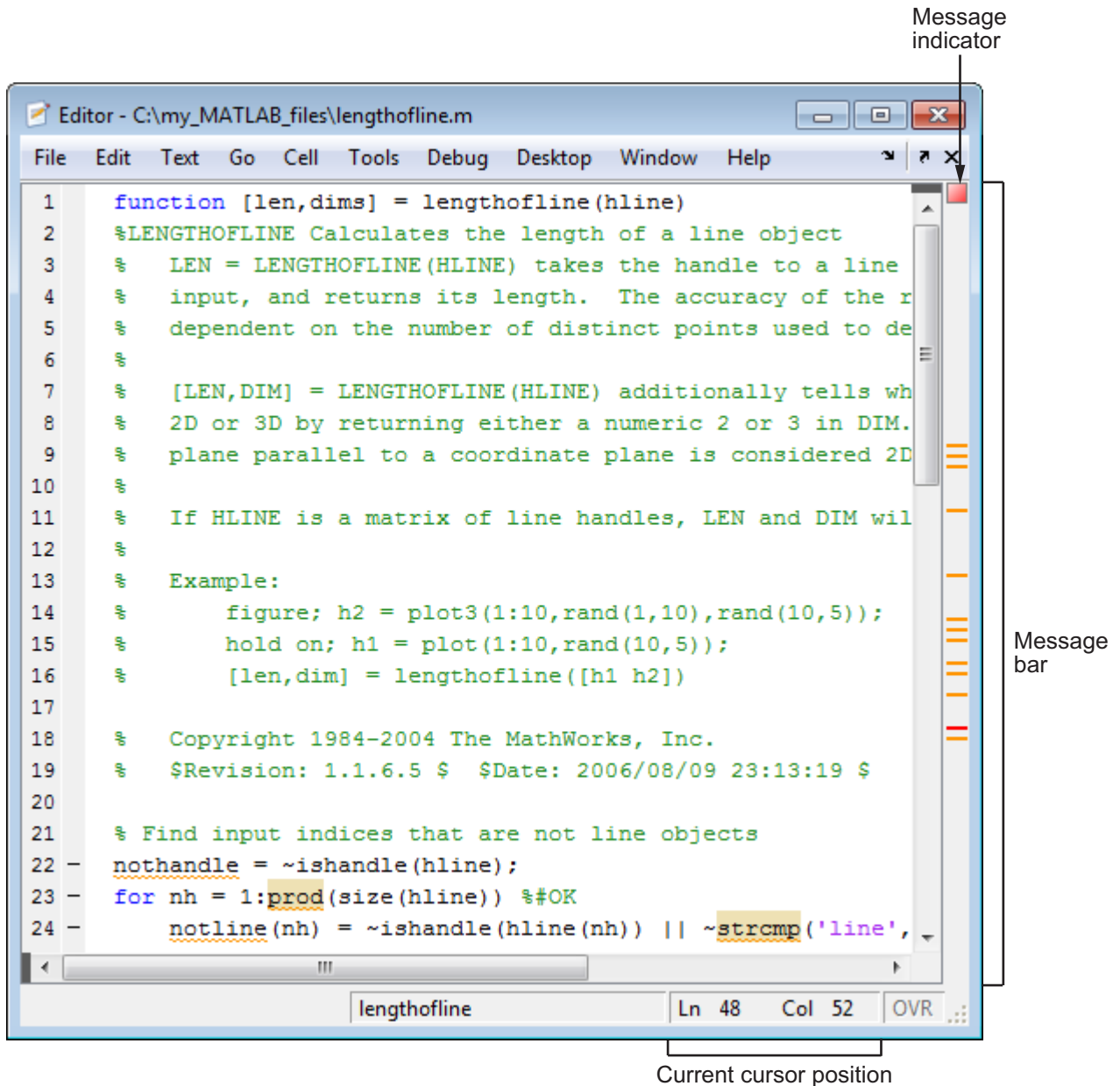
```
open(fullfile(matlabroot,'help','techdoc','matlab_env',...  
            'examples','lengthofline.m'))
```

- b** Save the example file to a folder to which you have write access. For the example, `lengthofline.m` is saved to `C:\my_MATLAB_files`.

4 Examine the message indicator at the top of the message bar to see the Code Analyzer 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.

In this example, the indicator is red, meaning that there is at least one error in the file.



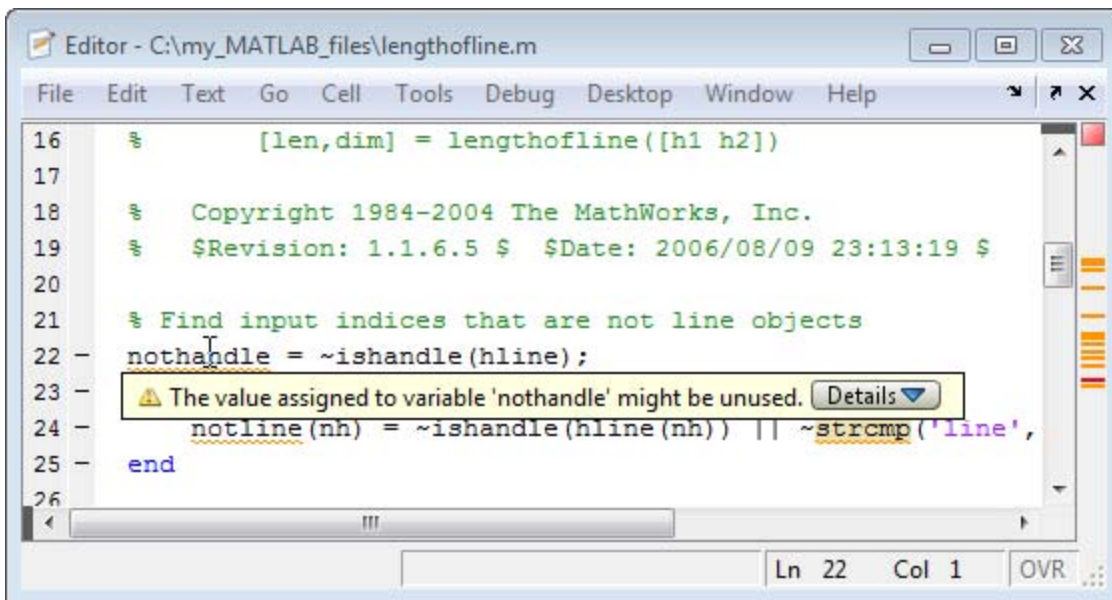
- 5 Click the message indicator to go to the next code fragment containing a message. The next code fragment is relative to the current cursor position, viewable in the status bar.

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 a message is underlined in either red for errors or orange for warnings and improvement opportunities.

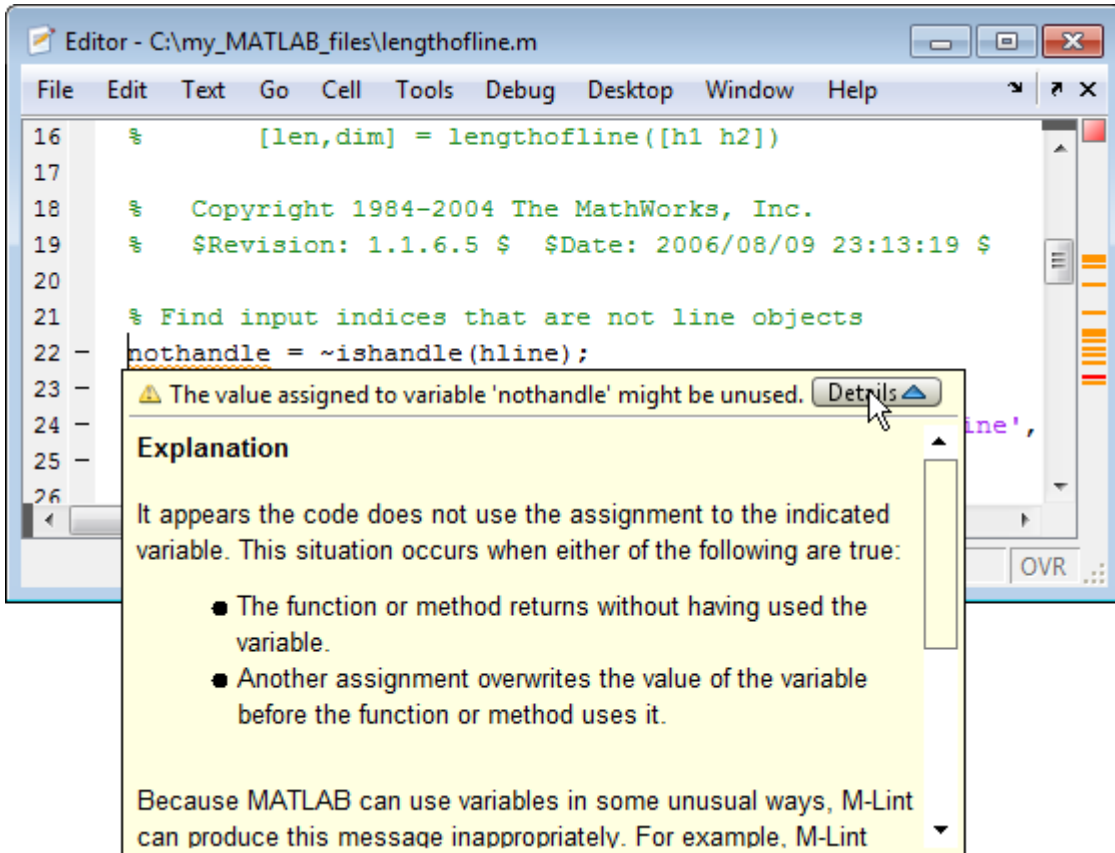
- 6 View the message by moving the mouse pointer within the underlined code fragment.

The message opens in a tooltip and contains a **Details** button that provides access to additional information by extending the message. Not all messages have additional information.



- 7 Click the **Details** button.

The window expands to display an explanation and user action.

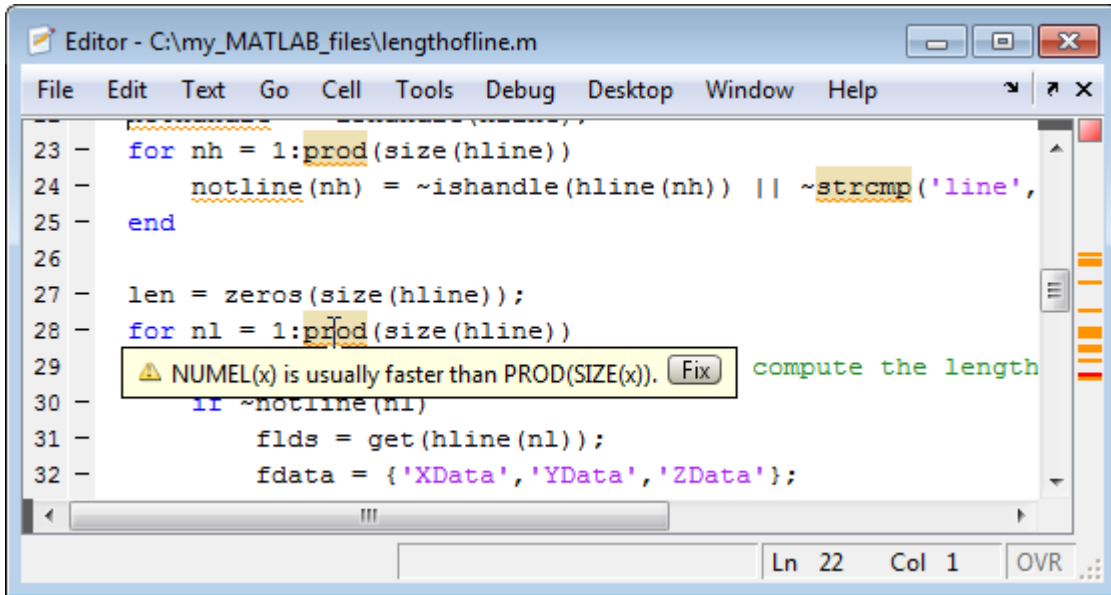


- 8 Modify your code, if needed.

The message indicator and underlining automatically update to reflect changes you make, even if you do not save the file.

- 9 On line 28, hover over `prod`.

The code is underlined because there is a warning message, and it is highlighted because an automatic fix is available. When you view the message, it provides a button to apply the automatic fix.

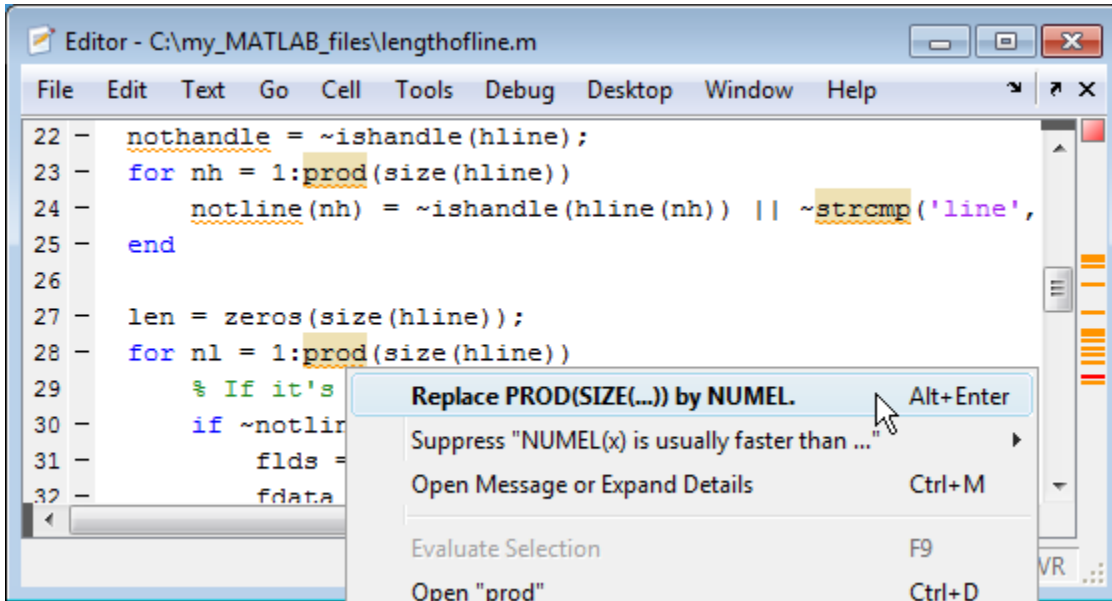


10 Fix the problem by doing one of the following:

- If you know what the fix is (from previous experience), click **Fix**.
- If you are unfamiliar with the fix, view, and then apply it as follows:
 - a Right-click the highlighted code (for a single-button mouse, press **Ctrl**+ click), and then view the first item in the context menu.
 - b Click the fix.

MATLAB automatically corrects the code.

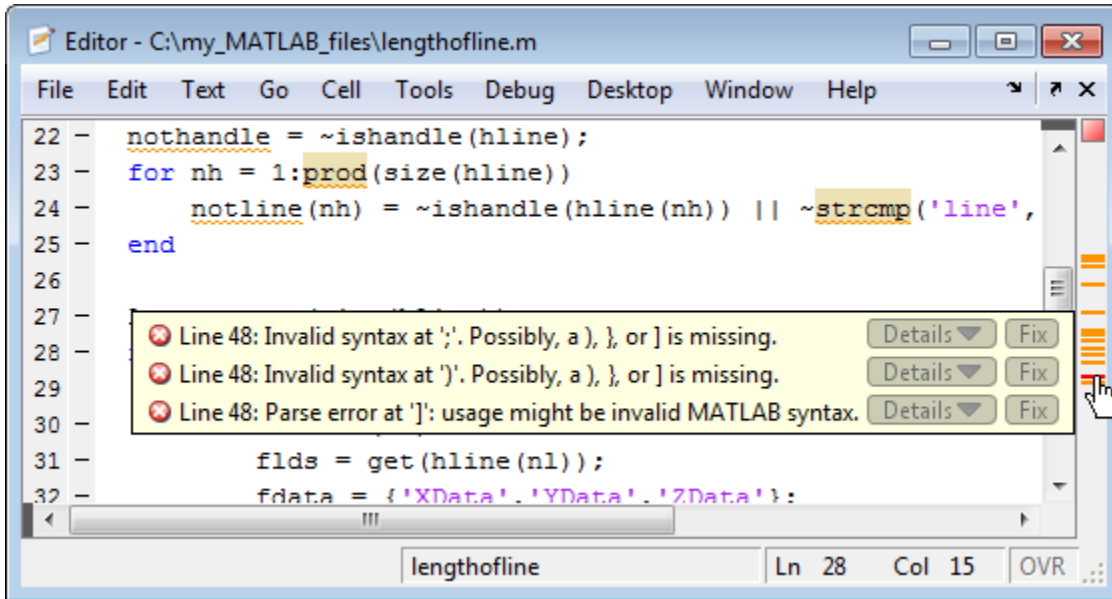
In this example, MATLAB replaces `prod(size(hline))` with `numel(hline)`.



11 Go to a different message by doing one of the following:

- To go to the next message, click the message indicator or the next underlined code fragment.
- To go to a line that a marker represents, click a red or orange line in the indicator bar .

To see the first error in `lengthofline`, click the first red marker in the message bar. The cursor moves to the first suspect code fragment in line 48. The **Details** and **Fix** buttons are dimmed, indicating that there is no more information about this message and there is no automatic fix.



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.

- 12 Modify the code to address the problem noted in the message—the message indicators update automatically.

In the `lengthofline`, the message suggests a delimiter imbalance. You can investigate that as follows:

- a Select **File > Preferences > Keyboard**.
- b Under **Delimiter Matching**, select **Match on arrow key**, and then click **OK**.
- c In the Editor, move the arrow key over each of the delimiters to see if MATLAB indicates a mismatch.

In the example, it might appear that there are no mismatched delimiters. However, code analysis detects the semicolon in parentheses: `data{3} (;)`, and interprets it as the end of a statement. The message

reports that the two statements on line 48 each have a delimiter imbalance.

- In line 48, change `data{3} (;)` to `data{3} (:)`.

Now, the underline no longer appears in line 48. The single change addresses the issues in both of the messages for line 48.


Because the change removed the only error in the file, the message indicator at the top of the bar changes from red to orange, indicating that only warnings and potential improvements remain.

After modifying the code to address all the messages, or disabling designated messages, the message indicator becomes green. The example file with all messages addressed has been saved as `lengthofline2.m`. Open the corrected example file with the command:

```
open(fullfile(matlabroot, 'help', 'techdoc', ...  
             'matlab_env', 'examples', 'lengthofline2.m'))
```

Create a Code Analyzer Message Report

You can create a report messages for an individual file, or for all files in a folder, using one of these methods:

- Run a report for an individual MATLAB code file:
 - 1 From a file in the Editor, select **Tools > Code Analyzer > Show Code Analyzer Report**.
 - 2 Modify your file based on the messages in the report.
 - 3 Save the file.
 - 4 Rerun the report to see if your changes addressed the issues noted in the messages.
- Run a report for all files in a folder:
 - 1 In the Current Folder browser, click the Actions button .
 - 2 Select **Reports > Code Analyzer Report**.
 - 3 Modify your files based on the messages in the report.

For details, see “Using the MATLAB Code Analyzer Report” on page 9-22.

- 4 Save the file.
- 5 Rerun the report to see if your changes addressed the issues noted in the messages.

Adjust Code Analyzer Message Indicators and Messages

Depending on the stage at which you are in completing a MATLAB file, you might want to restrict the code underlining. You can do this by using the Code Analyzer preference referred to in step 1, in “Avoid Mistakes While Editing Code” on page 8-84. For example, when first coding, you might prefer to underline only errors because warnings would be distracting.

Code analysis does not provide perfect information about every situation and in some cases, you might not want to change the code based on a message. If you do not want to change the code, and you do not want to see the indicator and message for that line, suppress them. For the `lengthofline` example, in line 49, the first 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. Code analysis 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 the message suggests.

There are a few different ways to suppress (turn off) the indicators for warning and error messages:

- “Suppress an Instance of a Message in the Current File” on page 8-101
- “Suppress All Instances of a Message in the Current File” on page 8-101
- “Suppress All Instances of a Message in All Files” on page 8-102
- “Save and Reuse Code Analyzer Message Settings” on page 8-103

You cannot suppress error messages such as syntax errors. Therefore, instructions on suppressing messages do not apply to those types of messages.

Suppress an Instance of a Message in the Current File. You can suppress a specific instance of a Code Analyzer message in the current file. For example, using the code presented in “Avoid Mistakes While Editing Code” on page 8-84, follow these steps:

- 1 In line 49, right-click at the first underline (for a single-button mouse, press **Ctrl+click**).
- 2 From the context menu, select **Suppress 'Terminate statement with semicolon...' > On This Line**.

The comment `%#ok<NOPRT>` appears at the end of the line, which instructs MATLAB not to check for a terminating semicolon at that line. The underline and mark in the indicator bar for that message disappear.

- 3 If there are two messages on a line that you do not want to display, right-click separately at each underline and select the appropriate entry from the context menu.

The `%#ok` syntax expands. For the example, in the code presented in “Avoid Mistakes While Editing Code” on page 8-84, ignoring both messages for line 49 adds `%#ok<NBRAK,NOPRT>`.

Even if Code Analyzer 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 to check for a terminating semicolon at that line, delete the `%#ok<NOPRT>` string from the line.

Suppress All Instances of a Message in the Current File. You can suppress all instances of a specific Code Analyzer message in the current file. For example, using the code presented in “Avoid Mistakes While Editing Code” on page 8-84, follow these steps:

- 1 In line 49, right-click at the first underline (for a single-button mouse, press **Ctrl+click**).
- 2 From the context menu, select **Suppress 'Terminate statement with semicolon...' > In This File**.

The comment `%ok<*NOPRT>` appears at the end of the line, which instructs MATLAB to not check for a terminating semicolon throughout the file. All underlines, as well as marks in the message indicator bar that correspond to this message disappear.

If there are two messages on a line that you do not want to display anywhere in the current file, right-click separately at each underline, and then select the appropriate entry from the context menu. The `%ok` syntax expands. For the example, in the code presented in “Avoid Mistakes While Editing Code” on page 8-84, ignoring both messages for line 49 adds `%ok<*NBRAK, *NOPRT>`.

Even if Code Analyzer 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 to check for a terminating semicolon in the file, delete the `%ok<*NOPRT>` string from the line.

Suppress All Instances of a Message in All Files. You can disable all instances of a Code Analyzer message in all files. For example, using the code presented in “Avoid Mistakes While Editing Code” on page 8-84, follow these steps:

- 1 In line 49, right-click at the first underline (for a single-button mouse, press **Ctrl+click**).
- 2 From the context menu, select **Suppress 'Terminate statement with semicolon...' > In All Files**.

This modifies the Code Analyzer preference setting.

If you know which message or messages you want to suppress, you can disable them directly using Code Analyzer preferences, as follows:

- 1 Select **File > Preferences > Code Analyzer**.
- 2 Search the messages to find the ones you want to suppress.
- 3 Clear the check box associated with each message you want to suppress in all files.
- 4 Click **OK**.

Save and Reuse Code Analyzer Message Settings. You can specify that you want certain Code Analyzer messages enabled or disabled, and then save those settings to a file. When you want to use a settings file with a particular file, you select it from the Code Analyzer preferences pane. That setting file remains in effect until you select another settings file. Typically, you change the settings file when you have a subset of files for which you want to use a particular settings file.

Follow these steps:

1 Select File > Preferences > Code Analyzer

The Preferences dialog box opens and displays the Code Analyzer preferences pane.

2 Enable or disable specific messages, or categories of messages.

3 Click the Actions button , select Save as, and then save the settings to a txt file.

4 Click OK.

You can reuse these settings for any MATLAB file, or provide the settings file to another user.

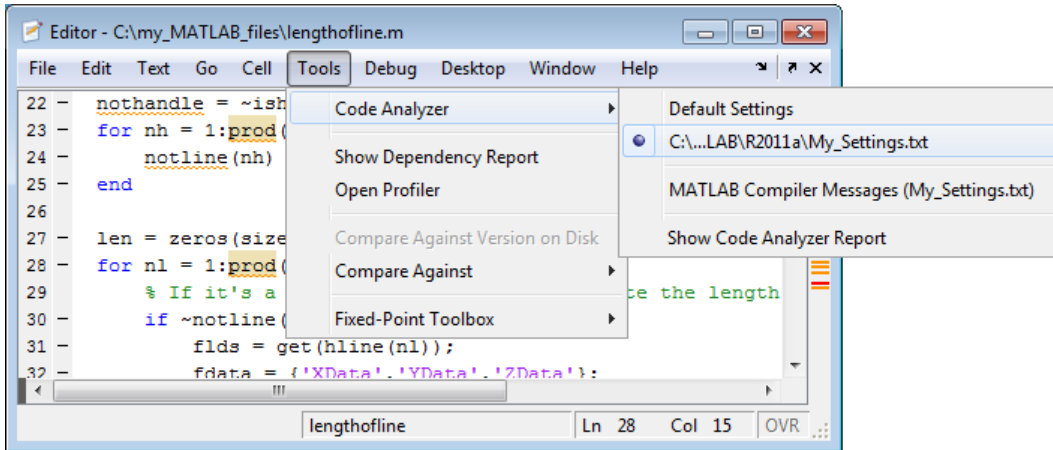
To use the saved settings:

1 In the Editor, select Tools > Code Analyzer.

The currently selected setting choice displays, preceded by a bullet point.

2 Choose from any of the settings files, such as the My_Settings example, as shown here.

The settings you choose are in effect for all MATLAB files until you select another set of Code Analyzer settings.



Understand Code Containing Suppressed Messages

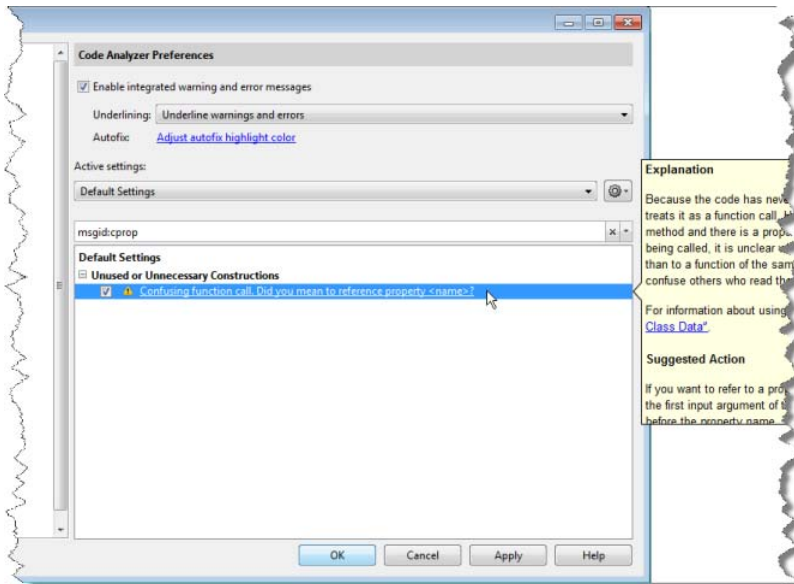
If you receive code that contains suppressed messages, you might want to review those messages without the need to unsuppress them first. A message might be in a suppressed state for any of the following reasons:



- One or more `%#ok<message-ID>` directives are on a line of code that elicits a message specified by `<message-ID>`.
- One or more `%#ok<*message-ID>` directives are in a file that elicits a message specified by `<message-ID>`.
- It is cleared in the Code Analyzer preferences pane.
- It is disabled by default.

To determine the reasons why some messages are suppressed:

- 1 Search the file for the `%#ok` directive and create a list of all the message IDs associated with that directive.
- 2 Open the Code Analyzer preferences dialog box by selecting **File > Preferences > Code Analyzer**.
- 3 In the search field, type `msgid:` followed by 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, click it to see an explanation and suggested action for the message. This can provide insight into why the message is suppressed or disabled. The following image shows how the Preferences dialog box appears when you enter `msgid:CPROP` in the search field.



- 4 Click the  button to clear the search field, and then repeat step 3 for each message ID you found in step 1.
- 5 Click the  button to clear the search field.
- 6 Display messages that are disabled by default and disabled in the Preferences pane by clicking the down arrow to the right of the search field. Then, click **Show Disabled Messages**.
- 7 Review the message associated with each message ID to understand why it is suppressed in the code or disabled in Preferences.

Understand the Limitations of Code Analysis

Code analysis is a valuable tool, but there are some limitations:

- Sometimes, it fails to produce Code Analyzer messages where you expect them.

By design, code analysis attempts to minimize the number of incorrect messages it returns, even if this behavior allows some issues to go undetected.

- Sometimes, it produces messages that do not apply to your situation.

When provided with message, click the **Detail** button for additional information, which can help you to make this determination. Error messages are almost always problems. However, many warnings are suggestions to look at something in the code that is unusual and therefore suspect, but might be correct in your case.

Suppress a warning message if you are certain that the message does not apply to your situation. If your reason for suppressing a message is subtle or obscure, include a comment giving the rationale. That way, those who read your code are aware of the situation.

For details, see “Adjust Code Analyzer Message Indicators and Messages” on page 8-100.

These sections describe code analysis limitations with respect to the following:

- “Distinguish Function Names from Variable Names” on page 8-107
- “Distinguish Structures from Handle Objects” on page 8-108
- “Distinguish Built-In Functions from Overloaded Functions” on page 8-108
- “Determine the Size or Shape of Variables” on page 8-108
- “Analyze Class Definitions with Superclasses” on page 8-108
- “Analyze Class Methods” on page 8-109

Distinguish Function Names from Variable Names. Code analysis cannot always distinguish function names from variable names. For the following code, if the Code Analyzer message is enabled, code analysis returns the message, Code Analyzer cannot determine whether xyz is a variable or a function, and assumes it is a function. Code analysis cannot make a determination because xyz has no obvious value assigned to it. However, the program might have placed the value in the workspace in a way that code analysis cannot detect.

```
function y=foo(x)
    .
    .
    .
    y = xyz(x);
end
```

For example, in the following code, xyz can be a function, or can be a variable loaded from the MAT-file. Code analysis has no way of making a determination.

```
function y=foo(x)
    load abc.mat
    y = xyz(x);
end
```

Variables might also be undetected by code analysis when you use the `eval`, `evalc`, `evalin`, or `assignin` functions.

If code analysis mistakes a variable for a function, do one of the following:

- Initialize the variable so that code analysis does not treat it as a function.
- For the `load` function, specify the variable name explicitly in the `load` command line. For example:

```
function y=foo(x)
    load abc.mat xyz
    y = xyz(x);
end
```

Distinguish Structures from Handle Objects. Code analysis cannot always distinguish structures from handle objects. In the following code, if `x` is a structure, you might expect a Code Analyzer message indicating that the code never uses the updated value of the structure. If `x` is a handle object, however, then this code can be correct.

```
function foo(x)
    x.a = 3;
end
```

Code analysis cannot determine whether `x` is a structure or a handle object. To minimize the number of incorrect messages, code analysis returns no message for the previous code, even though it might contain a subtle and serious bug.

Distinguish Built-In Functions from Overloaded Functions. Code analysis does not use the MATLAB path information because it can be different, depending on whether you are editing or running the program. If some built-in functions are overloaded in a class or on the path, Code Analyzer messages might apply to the built-in function, but not to the overloaded function you are calling. In this case, suppress the message on the line where it appears or suppress it for the entire file.

For information on suppressing messages, see “Adjust Code Analyzer Message Indicators and Messages” on page 8-100.

Determine the Size or Shape of Variables. Code analysis has a limited ability to determine the type of variables and the shape of matrixes. Code analysis might produce messages that are appropriate for the most common case, such as for vectors. However, these messages might be inappropriate for less common cases, such as for matrixes.

Analyze Class Definitions with Superclasses. Because code analysis looks at one file at a time and does not use the path, it has no way to analyze superclasses. Therefore, the amount of checking that code analysis can provide for a class definition with superclasses is limited. In general, code analysis cannot always tell whether the class is a handle class. It makes an educated guess, but often cannot get enough information, to make a determination for certain.

Analyze Class Methods. Most class methods must contain at least one argument that is an object of the same class as the method. But it does not always have to be the first argument. When it is, code analysis can determine that an argument is an object of the class you are defining, and it can do various checks. For example, it can check that the property and method names exist and are spelled correctly. However, when code analysis cannot determine that an object is an argument of the class you are defining, then it cannot provide these checks.

Enable MATLAB Compiler Deployment Messages

You can switch between showing or hiding Compiler deployment messages when you work on a file. Change the Code Analyzer preference for this message category. Your choice likely depends on whether 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, if the dialog box is open at the time you modify the setting in the Editor, you will not see the changes reflected in the Code Analyzer preferences dialog box. Whether you change the setting from the Editor or from the Code Analyzer preferences dialog box, it applies to the Editor and to the Code Analyzer Report.

To enable MATLAB Compiler™ deployment messages:

- 1** Select **File > Preferences > Code Analyzer**.
- 2** Click the down arrow next to the search field, and then select **Show Messages in Category > MATLAB Compiler (deployment) messages**.
- 3** Click the **Enable Category** button.
- 4** Clear individual messages that you do not want to display for your code (if any).
- 5** Decide if you want to save these settings, so you can reuse them next time you work on a file to be deployed.

The settings.txt file, which you can create as described in “Save and Reuse Code Analyzer Message Settings” on page 8-103, includes the status of this setting.

Avoid Variable and Function Scoping Problems

Scoping issues can be the source of some coding problems. For instance, if you are unaware that nested functions share a particular variable, the results of running your code might not be as you expect. Similarly, mistakes in usage of local, global, and persistent variables can cause unexpected results.

The Code Analyzer does not always indicate scoping issues because sharing a variable across functions is not an error—it may be your intent. Use MATLAB function and variable highlighting features to identify when and where your code uses functions and variables. If you have an active Internet connection, you can watch the Variable and Function Highlighting video demo for an overview of the major features.

For conceptual information on nested functions and the various types of MATLAB variables, see “Variable Scope in Nested Functions” and “Share Data Between Workspaces”.

Use Automatic Function and Variable Highlighting

By default, the Editor indicates functions, local variables, and variables with shared scope in various shades of blue. Variables with shared scope include: global variables, persistent variables, and variables within nested functions. (For more information, see “Nested Functions”.)

To enable and disable highlighting or to change the colors, select **File > Preferences > Colors > Programming tools**.

By default, the Editor:

- Highlights all instances of a given function or local variable in sky blue when you place the cursor within a function or variable name. For instance:

```
collatz
```

- Displays a variable with shared scope in teal blue, regardless of the cursor location. For instance:

```
x
```


Example of Using Automatic Function and Variable Highlighting

Consider the code for a function rowsum:

```
function rowTotals = rowsum
% Add the values in each row and
% store them in a new array

x = ones(2,10);
[n, m] = size(x);
rowTotals = zeros(1,n);
for i = 1:n
    rowTotals(i) = addToSum;
end

    function colsum = addToSum
        colsum = 0;
        thisrow = x(i,:);
        for i = 1:m
            colsum = colsum + thisrow(i);
        end
    end

end
```

When you run this code, instead of returning the sum of the values in each row and displaying:

```
ans =
    10    10
```

MATLAB displays:

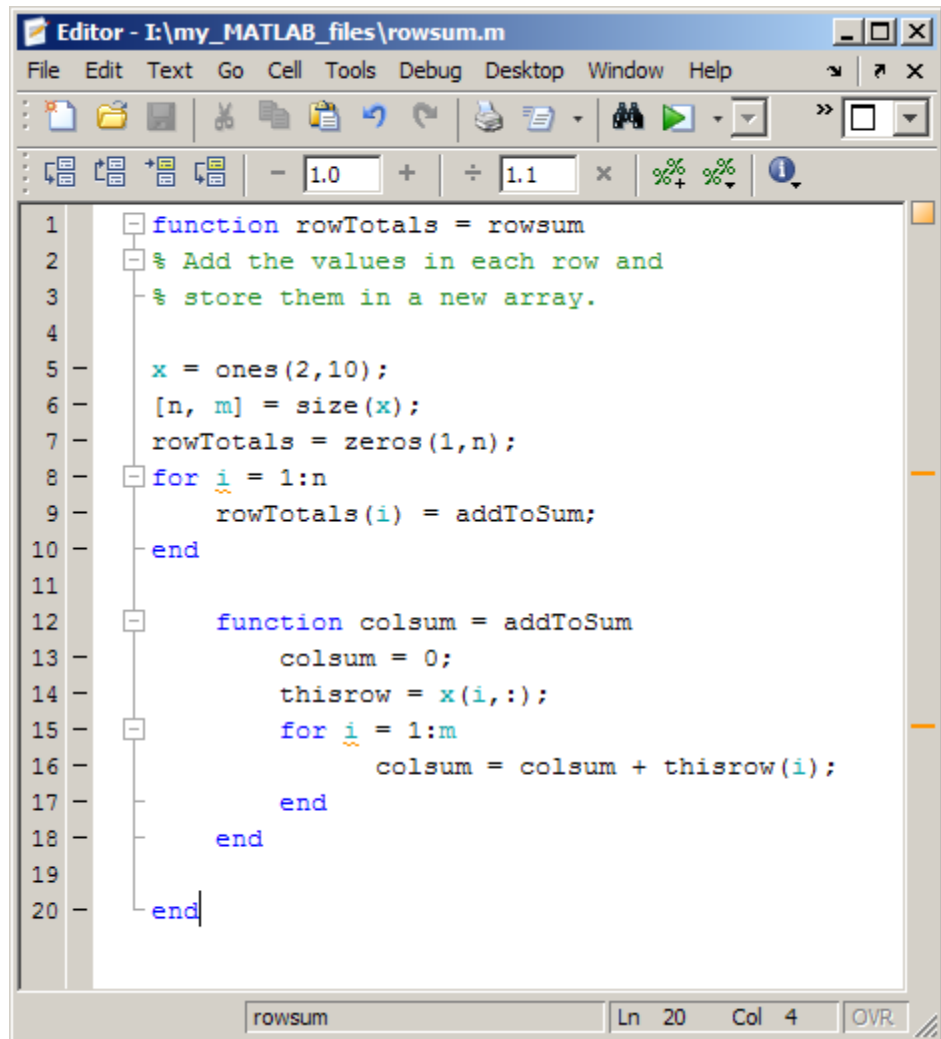
```
ans =
    0    0    0    0    0    0    0    0    0    10
```

Examine the code by following these steps:

- 1** Select **File > Preferences > Colors > Programming Tools**, and make sure **Automatically Highlight** and **Variables with shared scope** are selected.
- 2** Copy the rowsum code into the Editor.

Notice the variable `i` appears in teal blue, which indicates `i` is not a local variable. Both the `rowTotals` function and the `addToSum` functions set and use the variable `i`.

The variable `n`, at line 6 appears in black, indicating that it does not span multiple functions.



```
1 function rowTotals = rowsum
2 % Add the values in each row and
3 % store them in a new array.
4
5 x = ones(2,10);
6 [n, m] = size(x);
7 rowTotals = zeros(1,n);
8 for i = 1:n
9     rowTotals(i) = addToSum;
10 end
11
12     function colsum = addToSum
13         colsum = 0;
14         thisrow = x(i,:);
15         for i = 1:m
16             colsum = colsum + thisrow(i);
17         end
18     end
19
20 end
```

The image shows a MATLAB Editor window titled "Editor - I:\my_MATLAB_files\rowsum.m". The window contains a script with a function definition and a nested function. The main function, `rowsum`, is defined on lines 1-20. It initializes `x` as a 2x10 matrix of ones, gets the size of `x`, and initializes `rowTotals` as a 1x`n` array of zeros. It then enters a `for` loop over `i` from 1 to `n`. Inside this loop, it calls a nested function `addToSum`. The `addToSum` function is defined on lines 12-18 and takes no arguments. It initializes `colsum` to 0, gets the `i`-th row of `x` (`thisrow`), and then enters a `for` loop over `i` from 1 to `m`, adding `thisrow(i)` to `colsum`. The `addToSum` function ends on line 17, and the `rowsum` function ends on line 20. A tooltip is visible over the variable `i` in the `for` loop on line 8, indicating its scope across multiple functions.

3 Hover the mouse pointer over an instance of variable `i`.

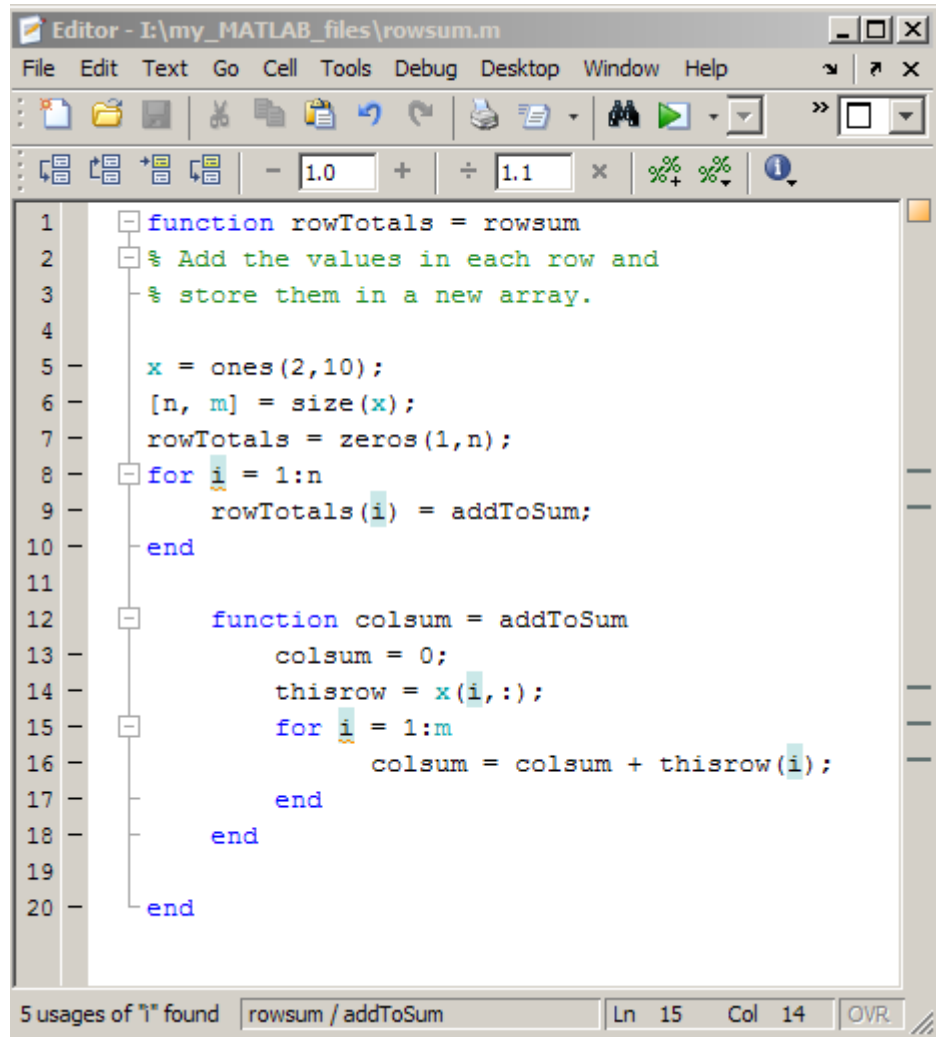
A tooltip appears: The scope of variable `i` spans multiple functions.

4 Click the tooltip link for information about variables whose scope span multiple functions.

5 Click an instance of `i`.

Every reference to `i` highlights in sky blue and markers appear in the indicator bar on the right side of the Editor. The status bar displays:

5 usages of `i` found



```
1 function rowTotals = rowsum
2 % Add the values in each row and
3 % store them in a new array.
4
5 x = ones(2,10);
6 [n, m] = size(x);
7 rowTotals = zeros(1,n);
8 for i = 1:n
9     rowTotals(i) = addToSum;
10 end
11
12 function colsum = addToSum
13     colsum = 0;
14     thisrow = x(i,:);
15     for i = 1:m
16         colsum = colsum + thisrow(i);
17     end
18 end
19
20 end
```

5 usages of 'i' found | rowsum / addToSum | Ln 15 Col 14 | OVR

6 Hover over one of the indicator bar markers.

A tooltip appears and displays the name of the function or variable and the line of code represented by the marker.

7 Click a marker to navigate to the line indicated in tooltip for that marker.

This is particularly useful when your file contains more code than you can view at one time in the Editor.

Fix the code by changing the instance of `i` at line 15 to `y`.

You can see similar highlighting effects when you click on a function reference. For instance, click on `addToSum`.

Debugging Process and Features

In this section...

“Ways to Debug MATLAB Files” on page 8-116

“Preparing for Debugging” on page 8-116

“Set Breakpoints” on page 8-119

“Run a File with Breakpoints” on page 8-123

“Step Through a File” on page 8-125

“Examine Values” on page 8-127

“Correct Problems and End Debugging” on page 8-133

“Conditional Breakpoints” on page 8-141

“Breakpoints in Anonymous Functions” on page 8-143

“Breakpoints in Methods That Overload Functions” on page 8-144

“Error Breakpoints” on page 8-145

Ways to Debug MATLAB Files

You can debug MATLAB 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:

- 1 Open the file — To use the Editor for debugging, open it with the file to run.
- 2 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. Therefore, you do not see the results of your changes.

- 3 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 required files are in the same folder, you can instead make that folder 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 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 resolves to 1 for all positive integers. The 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 you can study further.

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

Files for the Collatz Problem. Following are the two files you use for the debugging example. To create these files on your system, open two new files. Select and copy the following code from the Help browser and paste it into the 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.

Open the files by issuing the following commands, and then saving each file to a local folder:

```
open(fullfile(matlabroot, 'help', 'techdoc', 'matlab_env', ...  
    'examples', 'collatz.m'))
```

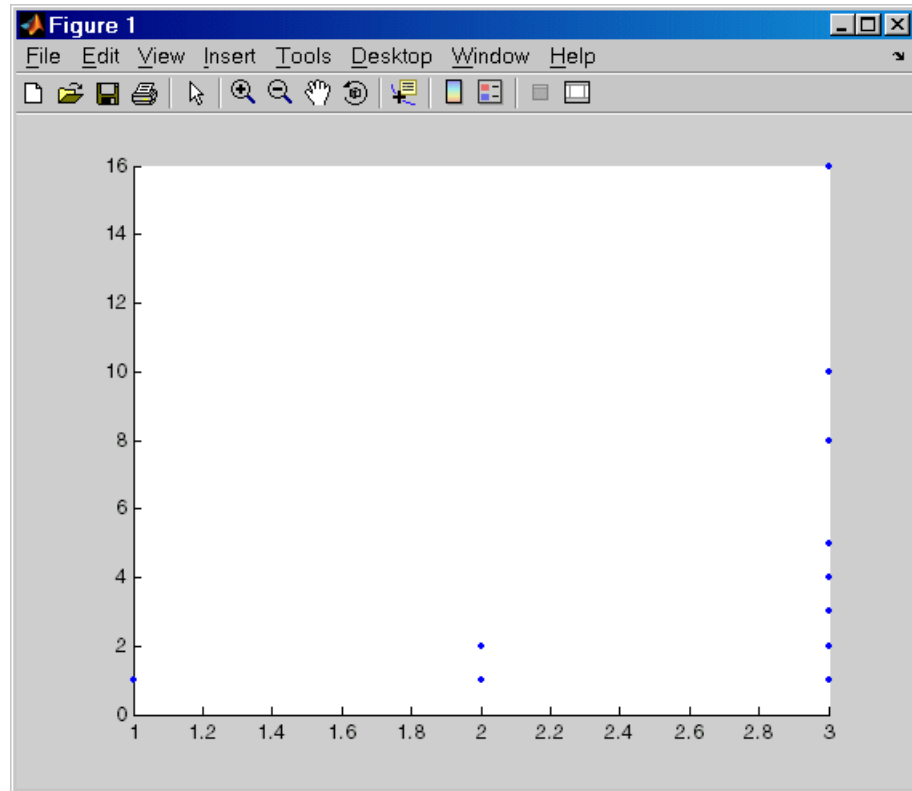
```
open(fullfile(matlabroot, 'help', 'techdoc', 'matlab_env', ...  
    'examples', 'collatzplot.m'))
```

Trial Run for the Example. Open the file `collatzplot.m`. Make sure that 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. 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.

Set Breakpoints

Set breakpoints to pause execution of the MATLAB file so you can examine values where you think the problem can be. You can the Editor, using functions in the Command Window, or both.

There are three basic types of breakpoints you can set in MATLAB files:

- A standard breakpoint, which stops at a specified line in a file. For details, see “Set Standard Breakpoints” on page 8-121.
- A conditional breakpoint, which stops at a specified line in a file only under specified conditions. For details, see “Conditional Breakpoints” on page 8-141.
- An error breakpoint that stops in any file when it produces the specified type of warning, error, or NaN or infinite value. For details, see “Error Breakpoints” on page 8-145.

You can disable standard and conditional breakpoints so that MATLAB temporarily ignores them, or you can remove them. For details, see “Disable and Clear Breakpoints” on page 8-134. Breakpoints do not persist 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 a file, unless that file is paused at a breakpoint.


Set 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** menu, select **Set/Clear Breakpoint**.

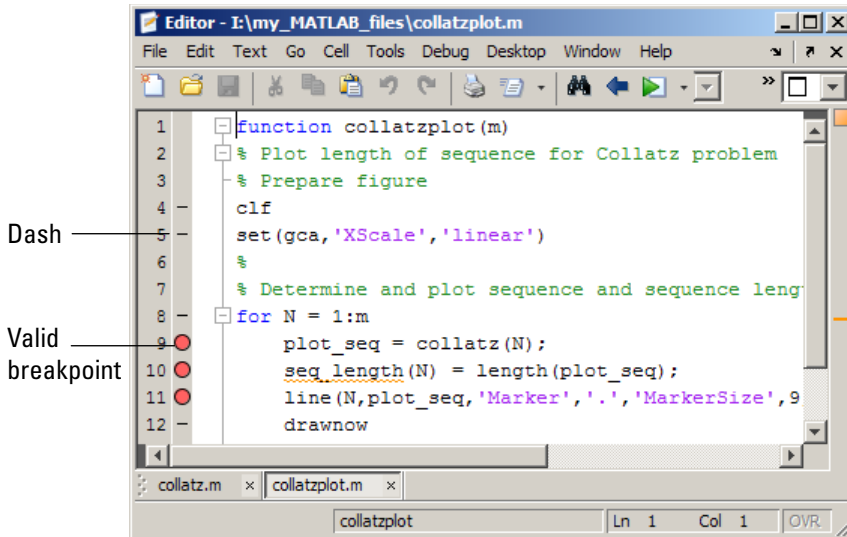
Set Breakpoints for the Example. It is unclear whether the problem in the example is in `collatzplot` or `collatz`. To start, follow these steps:

- 1 In `collatzplot.m`, click the dash in the breakpoint alley at line 9 to set a breakpoint.

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.

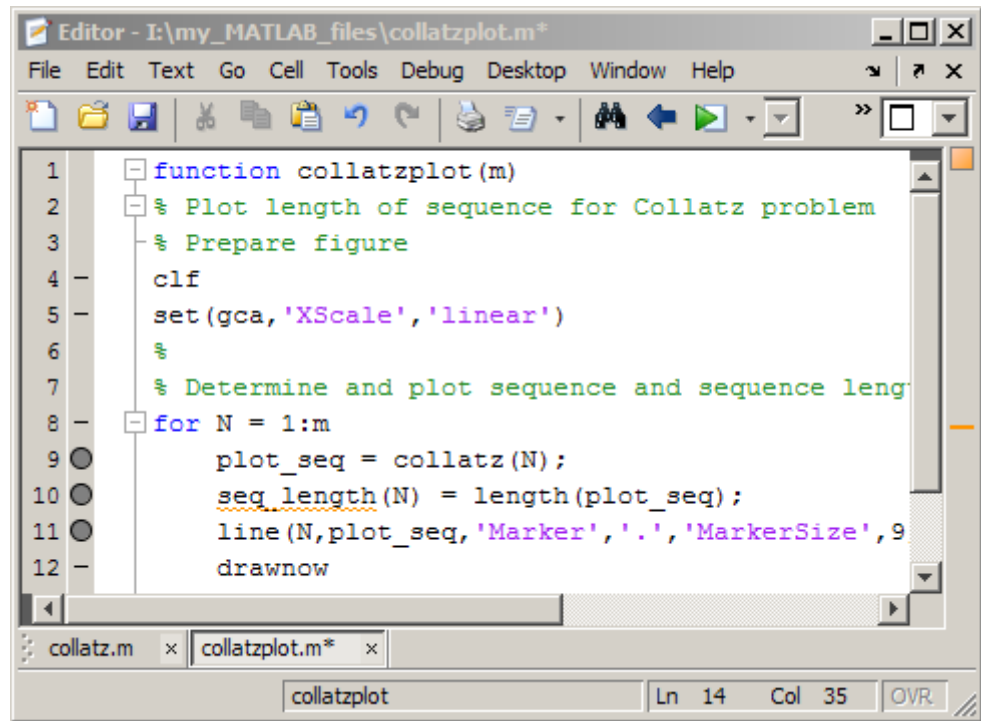


Valid (Red) and Invalid (Gray) Breakpoints. Red breakpoints indicate valid standard breakpoints.

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.

The following image shows three invalid breakpoints at lines 9, 10, and 11. Notice the asterisk next to the file name in the title bar. It indicates the file has unsaved changes.



```
1 function collatzplot(m)
2 % Plot length of sequence for Collatz problem
3 % Prepare figure
4 clf
5 set(gca,'XScale','linear')
6 %
7 % Determine and plot sequence and sequence length
8 for N = 1:m
9     plot_seq = collatz(N);
10    seq_length(N) = length(plot_seq);
11    line(N,plot_seq,'Marker','.', 'MarkerSize',9
12    drawnow
```

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
```

Run a File with Breakpoints

After setting breakpoints, run the file from the Command Window or the Editor.

Run the Example

For the example, run `collatzplot` for the simple input value, 3, by typing the following 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 a File Containing Breakpoints

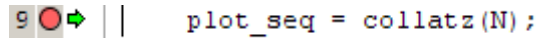
Running the file results in the following:

- 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. In the example, the pause is at line 9 of `collatzplot`, as shown here:



```
9 | plot_seq = collatz(N);
```

The line at which you are paused displays in the Command Window. For the example:

```
9 plot_seq = collatz(N);
```

- The function the **Stack** field displays 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.

MATLAB software could become nonresponsive if it stops at a breakpoint while displaying a modal dialog box or figure that your file creates. In that event, press **Ctrl+C** to go the MATLAB prompt.


Step Through a File






While debugging, you can step through a MATLAB file, pausing at points where you want to examine values.

Use any of the following methods:


- The Editor toolbar buttons
If the toolbar is not displaying, right-click the Editor menu bar, and then select **Editor Toolbar**.
- Step options in the Editor **Debug** menu
- The `dbstep` or `dbcont` function

Details about these methods appear in the following table.


Toolbar Button	Debug Menu Item	Description	Function Alternative
	Run <i>file</i> or Run Configuration for <i>file</i>	Commence execution of file and run until completion or until a breakpoint is encountered. The Run Configurations for <i>file</i> menu option provides a submenu. The submenu enables you to select a particular run configuration or to edit the run configurations for the MATLAB file. If you choose Run <i>file</i> ,	None

Toolbar Button	Debug Menu Item	Description	Function Alternative
		MATLAB uses the default run configuration.	
None	Go Until Cursor	Continue execution of file until the line where the cursor is positioned. Also available on the context menu.	None
	Step	Execute the current line of the file.	dbstep
	Step In	Execute the current line of the file and, if the line is a call to another function, step into that function.	dbstep in
	Continue	Resume execution of 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
	Exit Debug Mode	Exit debug mode.	dbquit 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.

Step into the 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 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.

Examine 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 you expect, then that line, or a previous line, contains an error. Use the following methods to examine values:

- “Select the Workspace” on page 8-127
- “View Values as Data Tips in the Editor” on page 8-128
- “View Values in the Command Window” on page 8-129
- “View Values in the Workspace Browser and Variable Editor” on page 8-130
- “Evaluate a Selection” on page 8-131
- “Examine Values in the Example” on page 8-131
- “Problems Viewing Variable Values from the Parent Workspace” on page 8-132

Many of these methods are used in “Examine Values in the Example” on page 8-131.

Select 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`.

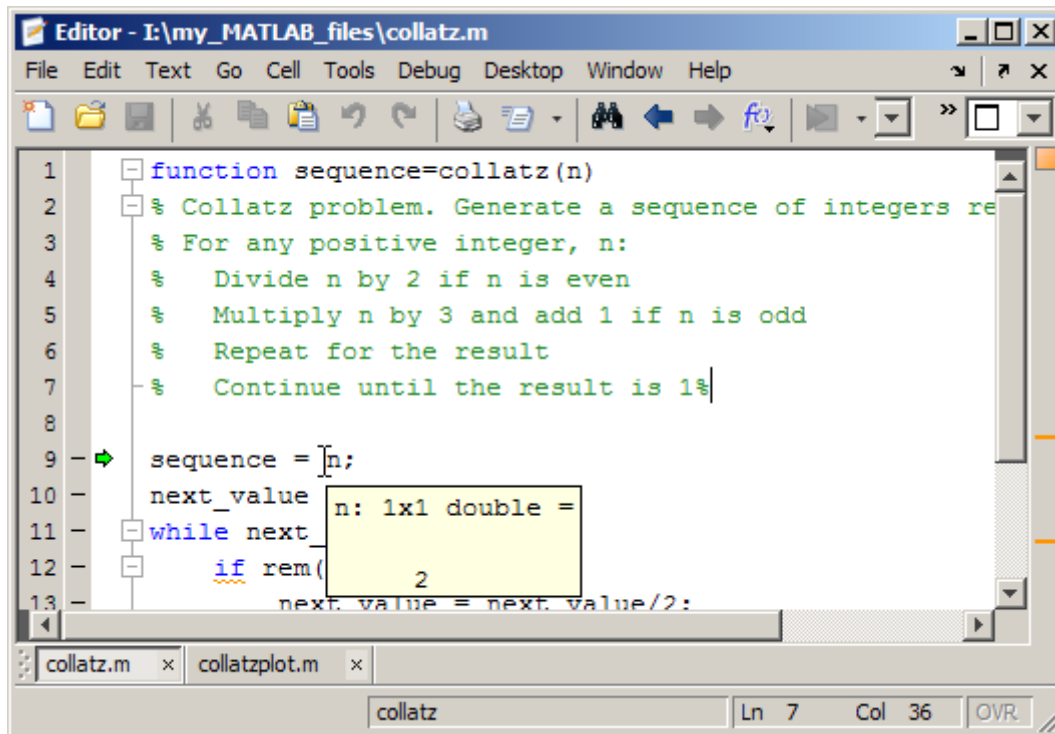
Workspace in the Example. At line 9 of `collatzplot`, you stepped in, and the current line is 9 in `collatz`. The **Stack** field shows that `collatz` is the current workspace.

View Values as Data Tips in the Editor

To view the current value of a variable in the Editor, first enable `datatips`, then use the mouse to display a `datatip`:

- 1** Select **File > Preferences > Editor/Debugger > Display**.
- 2** Select **Enable datatips in edit mode**.
- 3** In the Editor, position the mouse pointer to the left of a variable.

The variable's current value appears in a data tip. 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.



View 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
```

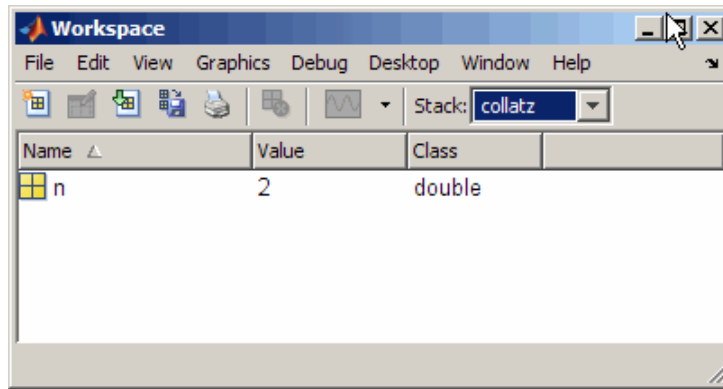
The Command Window displays the expected result

```
n =
    2
```

and displays the debug prompt, `K>>`.

View 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 **Stack** in the Workspace browser to change to another workspace and view its variables.

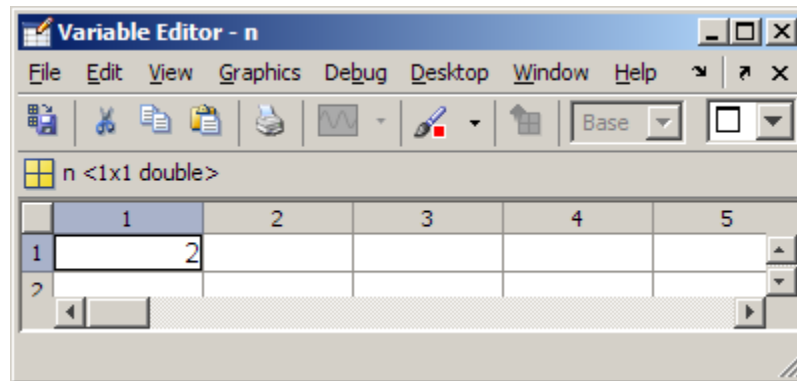


The **Value** column does not show all details for all variables. To see details, double-click a variable in the Workspace browser. The Variable Editor opens, displaying the content for that variable. You can open the Variable Editor directly for a variable using `openvar`.

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.



Evaluate a Selection

Select a variable or equation in a MATLAB file in the Editor. Right-click and select **Evaluate Selection** from the context menu (for a single-button mouse, press **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 a file.

Examine 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 \blacktriangledown . 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 to 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 the 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.

Correct Problems and End Debugging

The following are some of the ways to correct problems and end the debugging session:

- “Change Values and Check Results” on page 8-133
- “End Debugging” on page 8-133
- “Disable and Clear Breakpoints” on page 8-134
- “Save Breakpoints” on page 8-136
- “Correct Problems in a MATLAB File” on page 8-136
- “Complete the Example” on page 8-137
- “Run Sections in MATLAB Files That Have Unsaved Changes” on page 8-140

“Complete the Example” on page 8-137 uses many of these features.

Change Values and Check 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.

End Debugging

After identifying a problem, end the debugging session. You must end a debugging session if you want to change and save a file to correct a problem, or if you want to run other functions in MATLAB.

Note Quit debug mode before editing a file. If you edit a file while in debug mode, you can get unexpected results when you run the file. If you do edit a file while in debug mode, breakpoints turn gray, indicating that results might not be reliable. See “Valid (Red) and Invalid (Gray) Breakpoints” on page 8-122 for details.

If you attempt to save an edited 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.

Disable and Clear Breakpoints


Disable a breakpoint to ignore it temporarily. Clear a breakpoint to remove it.

Disable and Enable 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 8-141.

To disable a breakpoint, do one of the following:

- Right-click the breakpoint icon and select **Disable Breakpoint** from the context menu.
- Click anywhere in a line and select **Enable/Disable Breakpoint** from the **Debug** menu.

To disable a conditional breakpoint, use either of the methods in the preceding list or click the conditional breakpoint icon. An X appears through the breakpoint icon as shown here.

```
9  plot_seq = collatz(N);
```

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 reenable it to make it active again or you can clear it.

To reenable a breakpoint, do either of the following:

- Right-click the breakpoint icon and select **Enable Breakpoint** from the context menu.
- Click anywhere in a line and select **Enable/Disable Breakpoint** from the **Debug** menu.

The X no longer appears on the breakpoint icon and program execution will pause at that line.


Clear (Remove) Breakpoints. All breakpoints remain in a file until you clear (remove) them or until they are cleared automatically. 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** menu.
- Click a standard breakpoint icon, or a disabled conditional breakpoint icon.
- Use the `dbclear in 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 clear automatically when you:

- End the MATLAB session.
- Clear the file using `clear name` or `clear all`.

Note When `clear name` or `clear all` is in a statement in a file that you are debugging, it clears the breakpoints.

Save 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.

Correct Problems in a MATLAB File

To correct a problem in a MATLAB file:

- 1 Quit debugging.

Do not modify a file while MATLAB is in debug mode. If you do, breakpoints turn gray, indicating that results might not be reliable. See “Valid (Red) and Invalid (Gray) Breakpoints” on page 8-122 for details.

- 2 Modify the file.
- 3 Save the file.

- 4 Set, disable, or clear breakpoints, as appropriate.
- 5 Run the file again to be sure that it produces the expected results.

Complete the Example

To correct the problem in the example:

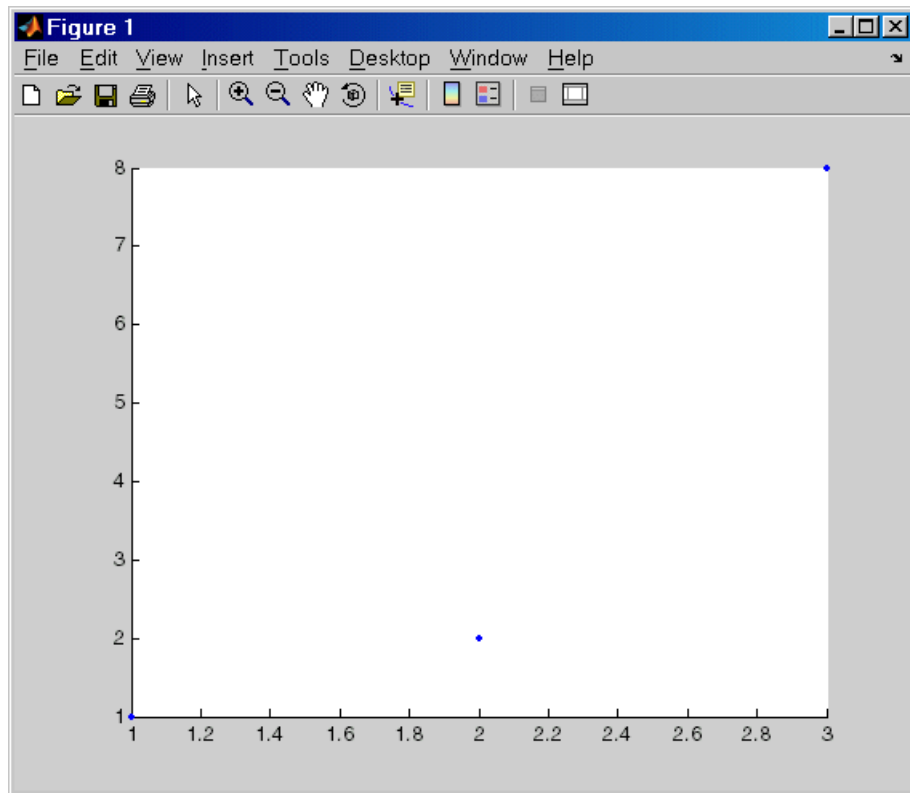
- 1 End the debugging session. One way to do this is to select **Exit Debug Mode** from the **Debug** menu.
- 2 In `collatzplot.m` line 11, change the string `plot_seq` to `seq_length(N)` and save the file.
- 3 Clear the breakpoints in `collatzplot.m`. One way to do this is by typing

```
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 that 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)
```

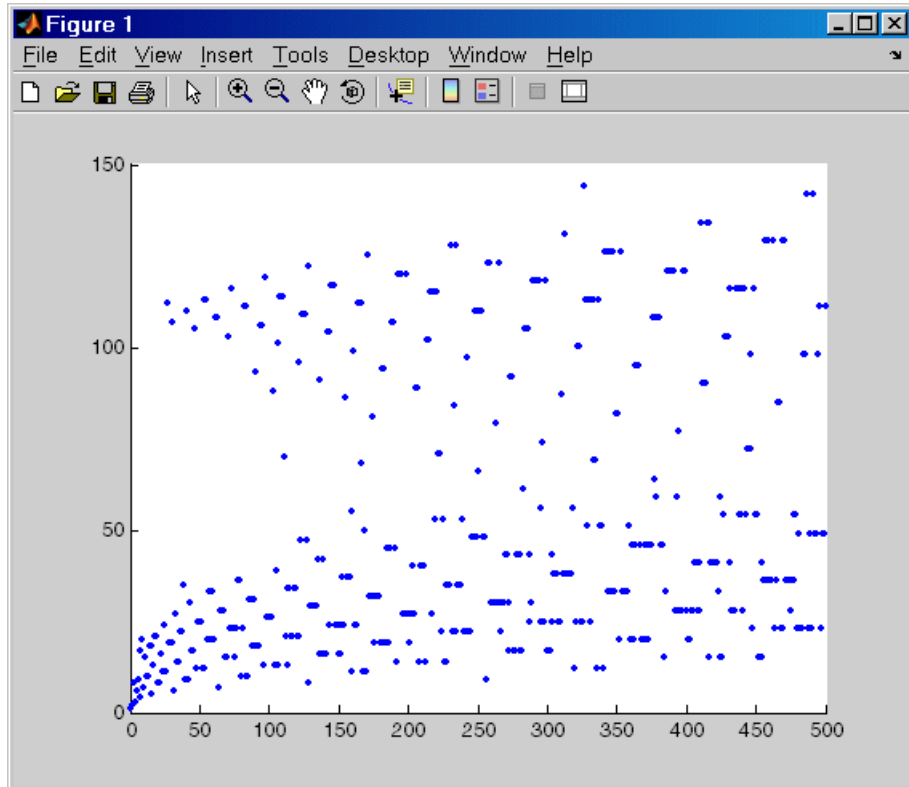
- 7 To make debugging easier, you ran `collatzplot` for a small value of `m`. Now that you know it works correctly, run `collatzplot` for a larger value to produce more interesting results. Before doing so, you consider disabling output for the line you just added in step 6, line 19 of `collatz.m`, by 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 $n = 500$.



Run Sections in MATLAB Files That Have Unsaved Changes

It is a good practice to modify a MATLAB file after you quit debugging, and then save the modification and run the file. Otherwise, you might get unexpected results. However, there are situations where you want to experiment during debugging. Perhaps you want to modify a part of the file that has not yet run, and then run the remainder of the file without saving the change. Follow these steps:

- 1 While stopped at a breakpoint, modify a part of the file that has not yet run.

Breakpoints turn gray, indicating they are invalid.

- 2 Select all of the code after the breakpoint, right-click, and then select **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 stop only if `N` is greater than or equal to 2. This section covers the following topics:

- “Set Conditional Breakpoints” on page 8-141
- “Modify, Disable, or Clear Conditional Breakpoints” on page 8-142
- “Function Alternatives for Manipulating Conditional Breakpoints” on page 8-142

Set Conditional Breakpoints

To set a conditional breakpoint:

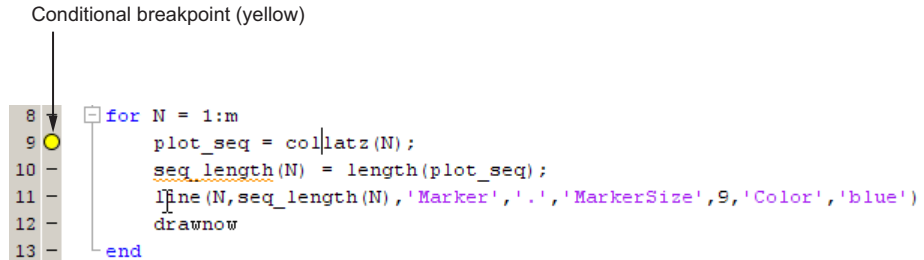
- 1 Click in the line where you want to set the conditional breakpoint.
- 2 From the **Debug** menu, select **Set/Modify Conditional Breakpoint**. If a standard breakpoint exists at that line, use this same method to make it conditional.

The MATLAB Editor conditional breakpoint dialog box opens.

- 3 Type a condition in the dialog box, where a condition is any valid 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 the following as the condition:

```
N>=2
```

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.

Modify, Disable, or Clear Conditional Breakpoints

The following table describes how to adjust conditional breakpoints.

To	Do This
Modify a condition for a breakpoint in the current line.	Right-click the conditional breakpoint icon, and then from the context menu, select Set/Modify Condition .
Disable a conditional breakpoint.	Click the associated conditional breakpoint icon.
Clear a conditional breakpoint.	Double-click the associated conditional breakpoint icon.

Function Alternatives for Manipulating Conditional Breakpoints


The following table lists the functions available for adjusting conditional breakpoints from the Command Window.

To	Use This Function
Set a conditional breakpoint.	<code>dbstop</code>
Clear a conditional breakpoint.	<code>dbclear</code>
View a list of currently set breakpoints, including the conditional expression for each conditional breakpoint.	<code>dbstatus</code>

Breakpoints in Anonymous Functions

You can set multiple breakpoints in a line of MATLAB code that contains anonymous functions. You can do both of the following:

- Set a breakpoint for the line itself (MATLAB software pauses at the start of the line).
- Set a breakpoint for each anonymous function in that line.

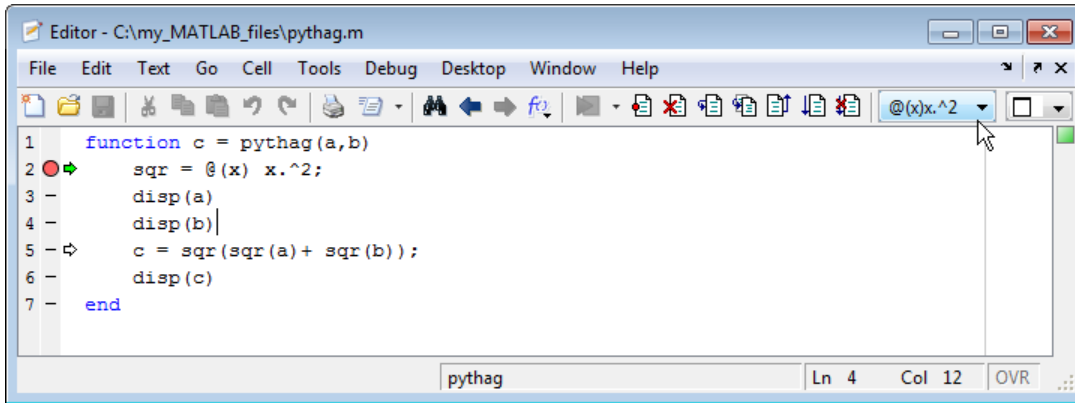
When you add a breakpoint to a line containing an anonymous function, the Editor asks where in the line you want to add the breakpoint. If there is more than one breakpoint in a line, the breakpoint icon is blue , regardless of the status of any of the breakpoints on the line.

To display information in a tooltip about all breakpoints on a line, position the pointer on the blue icon.

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 then select the action.

When you set a breakpoint in an anonymous function, MATLAB pauses 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. When you run the file, MATLAB pauses each time that the anonymous function `sqr` executes. The green arrow shows where the code defines `sqr`. The white arrow, on line 5, indicates where the code calls `sqr`.



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:

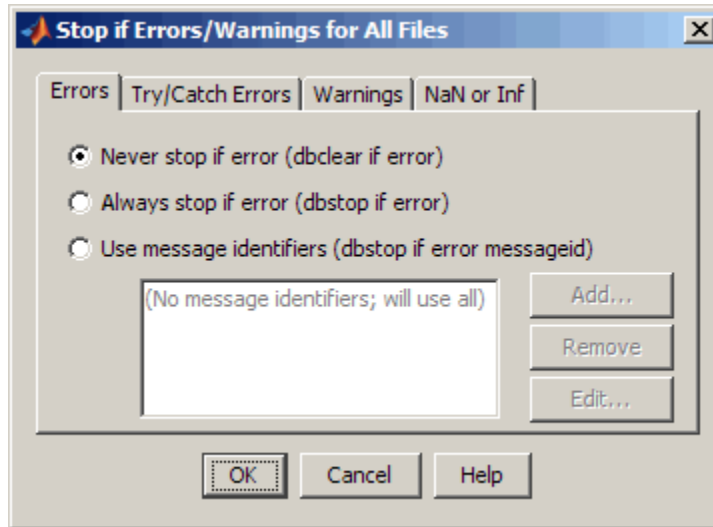
- “Set and Clear Error Breakpoints” on page 8-145
- “Error Breakpoint Types and Options” on page 8-146
- “Examples of Setting Warning and Error Breakpoints” on page 8-147
- “Function Alternative for Manipulating Error Breakpoints” on page 8-149

Set and Clear 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 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 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 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 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:

- Click **Never stop if error...** on a tab to clear that type of breakpoint.
- Click **Always stop if error...** 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 Click the **Warnings** tab.
- 2 Click **Always stop if warning**, and then click **OK**.

Now, when you run a file and MATLAB produces a warning, execution pauses and MATLAB enters debug mode. The file opens in the Editor at the line that produced the warning.

Set Breakpoints for a Specific Error. To stop execution for a specific error add a message identifier:

- 1** Click the **Errors, Try/Catch Errors, or Warnings** tab.
- 2** Click **Use Message Identifiers**.
- 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:narginchk:notEnoughInputs`). Then click **OK**.

The message identifier you specified appears in the Stop if Errors/Warnings for All Files dialog box.

- 5** Click **OK**.

Obtain Error Message Identifiers. To obtain an error message identifier generated by a MATLAB function, run the function to produce the error, and then call `MException.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:narginchk:notEnoughInputs'
  message: 'Not enough input arguments.'
  cause: {}
  stack: [1x1 struct]
```

Methods

Obtain 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:concatenation:integerInteraction`.

Function Alternative for Manipulating Error Breakpoints

The function equivalent for each option in the Stop if Errors/Warnings for All Files dialog box, appears 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 listed in the following table.

To	Use This Function
Set error breakpoints.	<code>dbstop</code>
Clear error breakpoints.	<code>dbc clear</code>
View a list of currently set breakpoints, including the condition and message identifier for each error breakpoint.	<code>dbstatus</code>

Debugging Functions

- `dbstop`—Set breakpoints
- `dbclear`—Clear breakpoints
- `dbcont`—Resume execution
- `dbdown`—Reverse workspace shift performed by `dbup`, while in debug mode
- `dbmex`—Enable MEX-file debugging (on UNIX platforms)
- `dbstack`—Function call stack
- `dbstatus`—List breakpoints
- `dbstep`—Execute one or more lines from current breakpoint
- `dbtype`—List text file with line numbers
- `dbup`—Shift current workspace to workspace of caller, while in debug mode
- `dbquit`—Quit debug mode

About MATLAB Code Files

MATLAB code files are files with a `.m` extension that contain MATLAB statements and functions. The Editor provides tools for creating and debugging these files.

Editor/Debugger Preferences

In this section...

“General Preferences for the Editor/Debugger” on page 8-152

“Editor/Debugger Display Preferences” on page 8-153

“Editor/Debugger Tab Preferences” on page 8-154

“Editor/Debugger Language Preferences” on page 8-155

“Editor/Debugger Code Folding Preferences” on page 8-158

“Editor/Debugger Autosave Preferences” on page 8-159

General Preferences for the Editor/Debugger

You can specify which editor MATLAB uses, as well as how the MATLAB Editor behaves under various circumstances.

Select **File > Preferences > Editor/Debugger**, and then adjust preference options as described in the table below.

Preference	Usage
Editor	<p>Select which editor you want the MATLAB desktop to use when you edit a file:</p> <ul style="list-style-type: none"> • MATLAB Editor • Text editor <p>If you select Text editor, specify the full path for the editor application you want to use, such as Emacs or vi. For example, <code>c:/Applications/Emacs.exe</code>.</p>
Most recently used file list	<p>In the Number of entries field, type the number of files that you want to appear in the list of recently used files at the bottom of the File menu.</p>

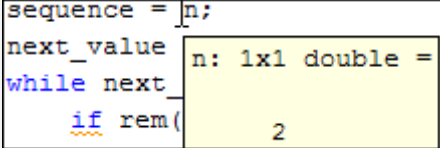
Preference	Usage
Opening files in editor	Select On restart open files from previous MATLAB sessions if you want the Editor and the files it contained during your last MATLAB session to reopen when you restart MATLAB.
Automatic file changes	Select Reload unedited files that have been externally modified if you want the Editor to automatically reload the version of a file that you opened and edited outside of MATLAB when the file currently open in the Editor has no unsaved changes.
	Select Add line termination at end of file to have MATLAB add a new empty line (sometimes referred to as a <CR>) to the end of a file automatically if the last line in the file is not empty.

Editor/Debugger Display Preferences

You can change the appearance of the Editor.

Select **File > Preferences > Editor/Debugger > Display**, and then adjust preference options as described in the table below..

Preference	Usage
General display option	Select Highlight Current Line and select a color to highlight the row with the cursor (also called the caret).
	Select Show line numbers to display line numbers along the left edge of the Editor window.
	Select Enable data tips in edit mode to display data tips when you are editing a MATLAB code file. (Data tips are always enabled in debug mode.)

Preference	Usage
	 <p>sequence = n; next_value n: 1x1 double = while next_ if rem(2</p> <p>For details see “View Values as Data Tips in the Editor” on page 8-128.</p>
Right-hand text limit	<p>Select Show line to display a vertical line with the specified Width and Color at the specified column (Placement) in the Editor.</p> <p>For details see “Right-Side Text Limit Indicator” on page 8-10.</p>

Editor/Debugger Tab Preferences

You can specify the size of tabs and indents and details about how tabs behave in the Editor.

Select **File > Preferences > Editor/Debugger > Tab**, and then adjust preference options as described in the table below.

Option	Usage
Tab size	<p>Specify the amount of space inserted when you press the Tab key.</p> <p>When you change the Tab size, it changes the tab size for existing lines in that file, unless you also select Tab key inserts spaces.</p>
Indent size	<p>Specify the indent size for <i>smart indenting</i>. Smart indenting is one of the “Editor/Debugger Language Preferences” on page 8-155.</p>

Option	Usage
Tab key inserts spaces	Select to insert a series of spaces when you press the Tab key. Otherwise, a tab acts as one space whose length is equal to the Tab size .
Emacs-style Tab key smart indenting	<p>Specifies an indenting style similar to the style that the Emacs editor uses.</p> <p>Lines indent 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.</p> <p>Smart indenting is one of the “Editor/Debugger Language Preferences” on page 8-155.</p> <p>If you select this preference, you cannot insert tabs within a line.</p>

Editor/Debugger Language Preferences

You can specify how various languages appear in the Editor. MATLAB applies language preferences based on the file extension of the file open in the Editor.

Select **File > Preferences > Editor/Debugger > Language**, and then adjust preference options as described in the table below.

Note Not all preferences are available for all languages.

Preference	Usage
Language	Select the language for which you want to set preferences.
Syntax highlighting	<p>Select Enable syntax highlighting to have the Editor use different colors for different language constructs. Then, adjust the colors you want to use for each language element.</p> <p>Access color options for the MATLAB language by clicking Set syntax colors.</p> <p>For all other languages, color options appear under Enable syntax highlighting.</p> <p>For details, see “Highlight Syntax to Help Ensure Correct Entries in the Editor” on page 8-84</p>
Variable and function renaming MATLAB Language only	<p>Select Enable automatic variable and function renaming to have MATLAB prompt you to rename all instances of a function or variable in a file when you rename a function or variable.</p> <div data-bbox="675 916 1243 1038" style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre>value1 = 0:1:6*pi; y=sin(val1) plot(val1,val2)</pre> <p style="text-align: center; margin: 0;">Press Shift+Enter to rename 3 instances of 'val1' to 'value1'</p> </div> <p>For details on when MATLAB prompts you, see Automatically Renaming All Functions or Variables in a File.</p>

Preference	Usage
<p>Comment formatting MATLAB Language only</p>	<p>In the Maximum column width field, enter the maximum number of characters you want to allow in a line of comments, and then select where you want counting to begin.</p> <p>Consider selecting:</p> <ul style="list-style-type: none"> • Start from beginning of line when the absolute width of the comments is important. For example, set 75 columns from the start of the line to match the width that fits on a printed page when you use the default font for the Editor. • Start from beginning of comment when comments are indented, and you want each block of comments to have a consistent indent and width. <hr/> <p>Select Wrap comments automatically while typing to automatically wrap comments at the Maximum column width value when you type comments in an Editor document.</p> <p>If you clear this option, you can still wrap comments manually, as described in “Wrap Comments Manually” on page 8-29.</p>
<p>Indenting</p>	<p>Select Apply smart indenting while typing to automatically:</p> <ul style="list-style-type: none"> • Indent the body of loops within the start and end of the loop statement. • Align subsequent lines with lines you indent using tabs or spaces. • Indent functions as specified with the Function indenting format option. <p>This is called <i>smart indenting</i>. You also can manually apply smart indenting after you type the code.</p>

Preference	Usage
	<p>For more information, see “Indent Code” on page 8-17.</p> <p>Select an option from Function Indenting Format (MATLAB Language only) to specify how functions indent in the Editor, as follows:</p> <ul style="list-style-type: none"> • 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. <p>For more information and examples of each indenting format, see “Indent Code” on page 8-17.</p>
File extensions	Add one or more file extensions to associate with the Language . The preferences you set for that language apply to all files with the listed extensions.

Editor/Debugger Code Folding Preferences

Code folding enables you to expand and collapse blocks of MATLAB code that you want to hide when you are not currently working on them.

Select **File > Preferences > Editor/Debugger > Code Folding**, and then adjust preference options as described in the table below.

For examples and detailed information about code folding, see “Code Folding — Expand and Collapse Code Constructs” on page 8-20

Option	Usage
Enable Code Folding	Specifies whether you want code folding enabled for the programming constructs that have their corresponding Enable check box selected.
Enable	Specifies whether you want code folding enabled for the corresponding Programming Construct . If you select this option for any construct, but clear the Enable Code Folding option, the construct will not have code folding enabled.
Fold Initially	Specifies whether the corresponding Programming Construct displays collapsed (folded) the first time that you open a MATLAB file.

Editor/Debugger Autosave Preferences

Can you specify if, when, and how you want MATLAB to automatically save files that are open in the Editor.

Select **File > Preferences > Editor/Debugger > Autosave**, and then adjust preference options as described in the table below.

Preference	Usage
Enable autosave in the MATLAB Editor	Select to have MATLAB automatically save a copy of the files you are currently editing.
Save options	Save every n minutes specifies how often you want MATLAB to save a copy of the file you are editing.
	Save untitled files saves a copy of new, untitled, files to <code>Untitled.asv</code> . When there is more than one untitled file, each additional file is saved to <code>Untitledn.asv</code> (where n is an integer value).

Preference	Usage
	For details, see “Autosaving Files” on page 8-7.
Close options	Automatically delete autosave files directs MATLAB to delete the autosave file when you close the source file in the Editor.
File name	<p>Select the naming convention you want MATLAB to use for autosave files. For example:</p> <ul style="list-style-type: none"> • If you specify Replace with extension: asv, the autosave file for <code>filename.m</code> is <code>filename.asv</code> • If you specify Append file name with ~, the autosave file for <code>filename.m</code> is <code>filename.m~</code>
Location	<p>Source file directories specifies that you want autosave files stored in the same folder as the files being edited.</p> <p>Single directory specifies that you want autosave files stored in a single folder. Specify the full path to that folder and be sure you have write permissions for it.</p>

About Code Analyzer Preferences

In this section...

“Code Analyzer Preferences” on page 8-161



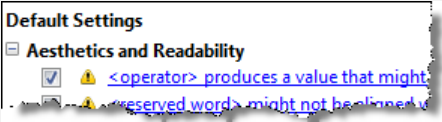
“Searching Messages in the Code Analyzer Preferences Dialog Box” on page 8-162

Code Analyzer Preferences

You can change how Code Analyzer messages appear in the Editor. With a few exceptions, these preferences apply to messages in the Editor, the MATLAB Function Block Editor (if your products use that tool), and the Code Analyzer Report.

Select **File > Preferences > Code Analyzer**. and then adjust preference options as described in the table below.

Option	Usage
Enabled Integrated Warning and Error Messages	Specify whether you want to display Code Analyzer message indicators, such as the underlining of code and the message indicator bar, for documents open in the Editor. For more information, see “Automatically Check Code in the Editor — Code Analyzer” on page 8-91
Underlining	Specify the type of coding issues that you want to have underlined. Regardless of the underlining menu option you choose, the Editor marks errors and warnings in the message indicator bar.
Autofix	Provides a link to a preference panel that enables you to adjust the color highlighting errors and warnings that MATLAB can autofix. You trigger autofix by clicking the Fix button in a Code Analyzer message.



Option	Usage
Active Settings	Select the set of message settings to use. Click the down arrow to select or browse to a previously saved settings file.
Actions button 	Click to open a menu that enables you to select: <ul style="list-style-type: none"> • Save as — Saves the current Code Analyzer message settings to a file. The default location for settings is the MATLAB preferences folder (the folder returned when you run <code>prefdir</code>). • Restore Defaults — Restores default Code Analyzer message settings.
Search field 	Searches the list of Code Analyzer messages that display below the search field. For details, see “Searching Messages in the Code Analyzer Preferences Dialog Box” on page 8-162.
Code Analyzer message settings 	Select or clear messages to enable or suppress their appearance in your Editor documents. To suppress a message on a line-by-line or file-by-file basis, see “Adjust Code Analyzer Message Indicators and Messages” on page 8-100.


Searching Messages in the Code Analyzer Preferences Dialog Box

You can search the list of Code Analyzer 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 Code Analyzer preferences pane does not display the **MATLAB Compiler (deployment) messages** category.

To See a List of Messages ...	Perform this action...	Example Scenario
Containing a given string in the: <ul style="list-style-type: none"> • Short message • Extended message • Message category • Message ID 	Type the string in the search 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 <code>com</code> in the search field to display those messages that contain that string in the short message, extended message, or message ID.
Corresponding to a given message ID	Type <code>msgid:</code> followed by the message ID in the search 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 <code>%#ok<AGROW></code> directive. Type <code>msgid:agrow</code> in the search field. Messages IDs containing <code>AGROW</code> display as links. Click each link for more information about the message. Not all Code Analyzer messages have additional information. These messages do not appear as links.
That you can set using Code Analyzer 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 searched the messages for a given string or search menu option.

To See a List of Messages ...	Perform this action...	Example Scenario
Different from the default setting (of enabled or disabled)	<p>Click the down arrow to the right of the search field, and then click Show Messages Modified from Default.</p> <p>A gray dot precedes a message with a setting different from the default. For example:</p> <p><input checked="" type="radio"/> <input type="checkbox"/>  DATENUM(NOW)</p>	A coworker gave you a settings file and you want to review each message that the coworker changed from its default setting.
In a given category	Click the down arrow to the right of the search field, click Show Messages in Category , and then click the category you want.	<p>You want to review messages that describe coding practices that make it difficult for others to use your code.</p> <p>Click the down arrow to the right of the search field, select Show Messages in Category, and then select Aesthetics and Readability.</p> <p>Click the messages that appear as links for more information. Not all messages appear as links.</p>
That are warnings	Click the down arrow to the right of the search field, and then select 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 search 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 ...	Perform this action...	Example Scenario
Are errors	Click the down arrow to the right of the search field, and then select Show All Errors . By default, an X in a red dot indicates an error message,  .	<p>You want to find a message elicited by a script you worked on previously. All you can recall is that it was an error and it involved <code>parfor</code>.</p> <p>Click the down arrow to the right of the search field, and then select Show All Errors. Then, type a space and <code>parfor</code> in the search field.</p> <p>The Code Analyzer preference pane displays only error messages that contain the word <code>parfor</code>.</p>
Are disabled	Click the down arrow to the right of the search field, and then select Show Disabled Messages .	You want to see the messages that are disabled by default or you have previously disabled.

Example of Searching Messages

To display Code Analyzer error messages that contain the string `variable` and are disabled:


- 1 Click the down arrow in the search field, and then select **Show All Errors**.

The search field contains the string `severity:error`.

- 2 At the end of the string `severity:error`, press the **Space** key, and then type `variable`.

- 3 Click the down arrow in the search field and select **Show Disabled Messages**.

The search field now contains the string `severity:error variable enabled:false`. Only the messages that fulfill those requirements appear in the Preferences pane.

To restore the list of all messages, click the Clear search button .

Tuning and Managing MATLAB Code Files

This set of tools provides useful information about the MATLAB code files in a folder that can help you refine the files and improve performance. The tools can help you polish these files before providing them to others to use.

- “Using MATLAB Reports” on page 9-2
- “Using the MATLAB Code Analyzer Report” on page 9-22
- “Profiling for Improving Performance” on page 9-27

Using MATLAB Reports

In this section...

“Refine and Improve Files Using Reports” on page 9-2

“Annotate Files with Reminders and Generate Report” on page 9-3

“Generating a Summary View of the Help Components in Functions and Scripts” on page 9-7

“Displaying and Updating a Report on the Contents of a Folder” on page 9-10

“Displaying Dependencies Among MATLAB Code Files” on page 9-14

“Identifying How Much of a File Ran When Profiled” on page 9-19

See also “Using the MATLAB Code Analyzer Report” on page 9-22 the Comparison Tool.


Refine and Improve Files Using Reports

Reports help you refine MATLAB code files within a given folder and improve their performance. They are also useful for checking the quality of files before you distribute them for use by others, to share on MATLAB Central, or for a toolbox. (A toolbox is a collection of files for use with MATLAB and related products.)

See also “Using the MATLAB Code Analyzer Report” on page 9-22.

Accessing Reports

To access reports:

- 1 In the Current Folder browser, navigate to the folder containing the files for which you want to produce reports.
- 2 On the Current Folder browser toolbar, click the **Actions** button , and then select the type of report you want to run.

The report runs for all the MATLAB code 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.

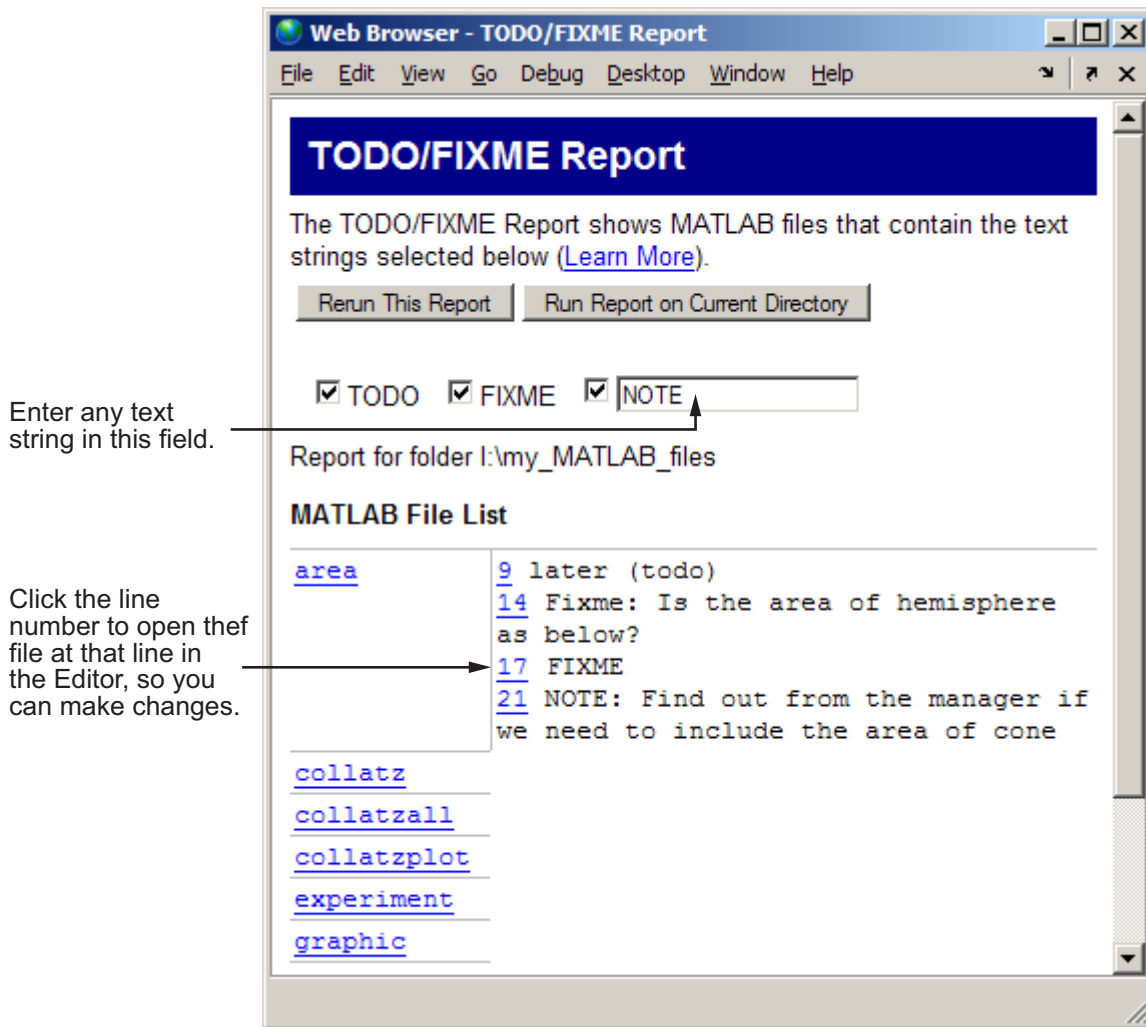
Annotate Files with Reminders and Generate Report

Annotating a file makes it easier to find areas of your code that you intend to improve, complete, or update later.

To annotate a file, add comments with the text `TODO`, `FIXME`, or a string of your choosing.

After you annotate several files, run the `TODO/FIXME` Report, to identify all the MATLAB code files within a given folder that you have annotated.

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 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 can enter NOTE, tbd, or re.*check.

- 4 Run the report on the files in the current folder, by clicking **Rerun This Report**.

The window refreshes and lists all lines in the MATLAB files 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 report window, change the current folder. Then, click **Run Report on Current Folder**.

To open a file in the Editor at a specific line, click the line number in the report. Then you can change the file, as needed.

Suppose you have a file, `area.m`, in the current folder. The code for `area.m` appears in the image that follows.

```

1 function [output] = area(flag,radius)
2 % This function calculates the area of the entity
3 % flag = 1 for calculating the area of a
4 % flag = 2 for calculating the surface area of a sphere
5 % radius = radius of the entity
6
7 switch flag
8     % Modify the function to include the area of square
9     % and rectangle. (todo)
10
11 case 1
12     output = pi * radius^2;
13
14 case 2
15     output = 4 * pi * radius^2;
16     % Fixme: is the area of hemisphere as below?
17     % case 3
18     % output = 2 * pi * radius^2;
19     % fixME
20
21 otherwise
22     disp('Incorrect flag');
23     output = NaN;
24     % NOTE: Find out from manager if we need to include
25     % the area of a cone
26
27 end

```

area Ln 14 Col 28 OVR

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

```

<u>area</u>	<u>9</u> and rectangle. (todo) <u>14</u> Fixme: Is the area of hemisphere as below? <u>17</u> fixme <u>21</u> 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 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 that you select the text field.

Generating a Summary View of the Help Components in Functions and Scripts

A Help Report presents a summary view of the help component of your MATLAB 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 *help component* is all contiguous nonexecutable lines (comment lines and blank lines), starting with the second line of a function file or the first line of a script file. For more information about creating help for your 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 MATLAB 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 class methods**—display help information for class methods created using the `classdef` keyword as well as functions.
- **Description**—display the first line of help in the file. If the first comment line is empty, or if there is no comment before the executable code, then **No description line**, highlighted in pink, appears instead.
- **Examples**—display the line number where the examples section of the help begins. The Help Report performs a case-insensitive search for a help line that contains only the string `example`, and displays that line and any subsequent nonblank comment lines. Select this option to easily locate and go to examples in your files.

If the report does not find examples in the MATLAB code file help, **No example**, highlighted in pink, appears.

- **Show all help** —display complete MATLAB code file help, which is all contiguous nonexecutable lines (comment lines and blank lines), starting with the second line of a function file, or the first line of a script file. The help displayed also includes overloaded functions and methods, which are not actually part of the help comments, but are automatically generated when `help` runs. Description lines are reported twice if you also have also selected **Description**.

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**—display the line number for the `see also` line in the help. The `see also` line in help lists related functions. When the MATLAB Command Window displays the help for a MATLAB code 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 files.

The report looks for a line in the help that begins with the string `See also`. If the report does not find a `see also` line in the help, **No see-also line**, highlighted in pink, appears. This helps you identify

those files without a `see also` line, should you want to include one in each MATLAB code file.

The report also indicates when a 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**—display the line number for the copyright line in the file. The report looks for a comment line in the 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).

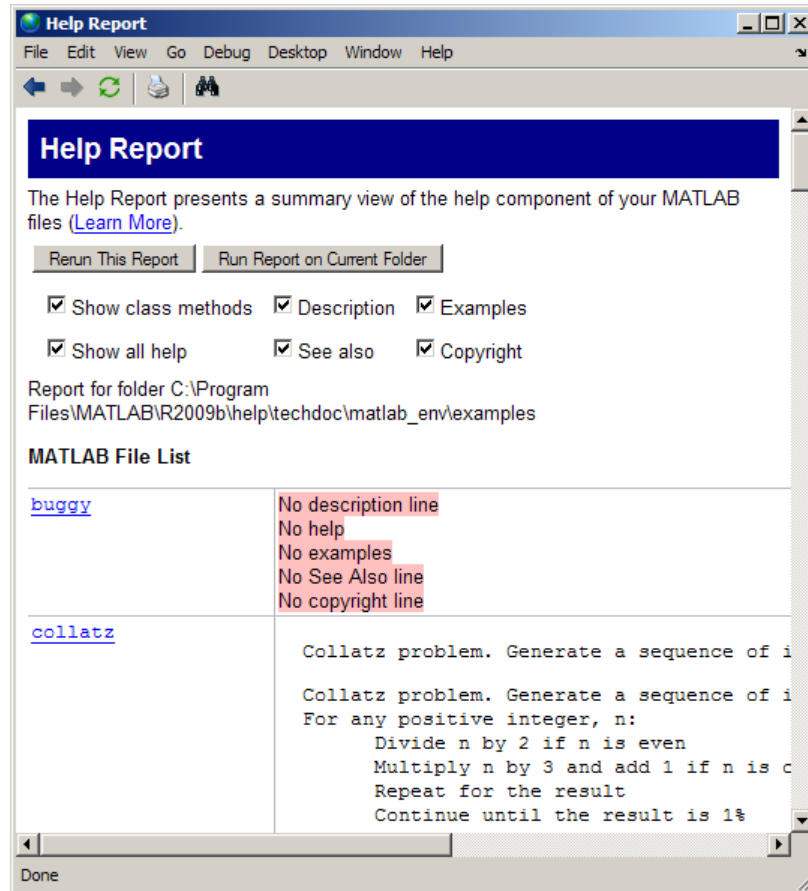
It is a good practice to include a copyright line in the help that notes the year you created the file and the current year. For example, for a file you created in 2001, include this line

```
% Copyright 2001-2008
```

If the report:

- Does not find a copyright line in the 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.




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 MATLAB code 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 MATLAB code 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”.

Working with Contents Reports

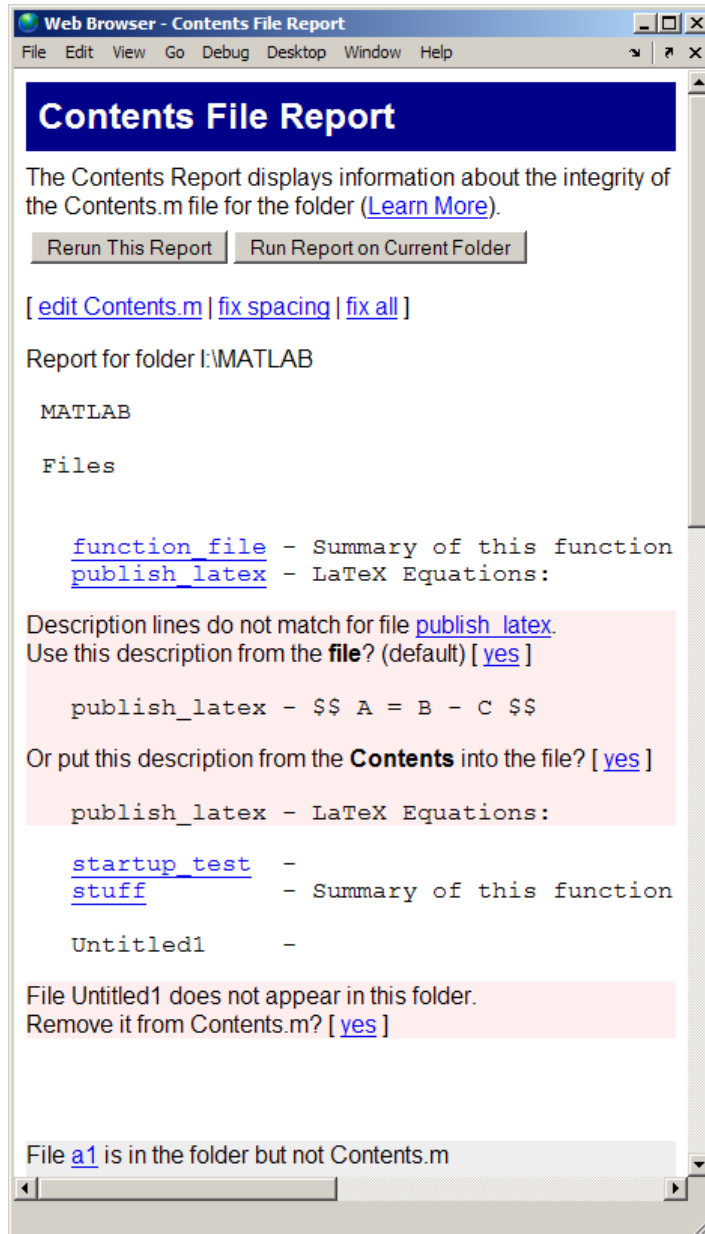
- 1 Select **Desktop > Current Folder** and navigate to the folder containing the files for which you want to produce a Contents report.
- 2 On the Current Folder browser toolbar, click the **Actions** button , and then select **Reports > Contents Report**.

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 entries for files that you do not want to expose when displaying help for the folder, such as code intended for internal use only.

- 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.

Choose from the following options to 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.



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 MATLAB code 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 file's help does not match the description provided for the 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 file. Or select the option to replace the description line in the 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 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 MATLAB Code Files

The Dependency Report shows dependencies among MATLAB code files in a folder. Use this report to determine:

- 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:

- Files in the `toolbox/matlab` folder because every MATLAB user has those files.
Therefore, if you use a function file that shadows a built-in function file, MATLAB excludes both files from the list.
- Files called from anonymous functions.
- The superclass for a class file.
- Files called from `eval`, `evalc`, `run`, `load`, function handles, and callbacks.

MATLAB does not resolve these files until run time, and therefore the Dependency Report cannot discover them.

- Some method files.


The Dependency Report finds class constructors that you call in a MATLAB file. However, any methods you execute on the resulting object are unknown to the report. These methods can exist in the `classdef` file, as separate method files, or files belonging to superclass or superclasses of a method file.

To provide meaningful results, the Dependency Report requires the following:

- 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.

Note Do not use the Dependency Report to determine which MATLAB code files someone else needs to run a particular file. Instead use the `depfun` function.

Creating Dependency Reports

- 1 Select **Desktop > Current Folder** and navigate to the folder containing the 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 MATLAB code files (children) called by each file in the folder (parent), select **Show child functions**.

The report indicates where each child function resides, for example, in a specified toolbox. If the report specifies that the location of a child function is unknown, it can be because:

- The child function is not on the search path.
 - The child function is not in the current folder.
 - The file was moved or deleted.
- To list the files that call each MATLAB code file, select **Show parent functions**.
- The report limits the parent (calling) functions to functions in the current folder.
- To include subfunctions in the report, select **Show subfunctions**. The report lists subfunctions directly after the main function and highlights them in gray.

4 Click **Run Report on Current Folder**.

Reading and Working with Dependency Reports

The following image shows a Dependency Report. It indicates that `chirpy.m` calls two files in Signal Processing Toolbox and one in Image Processing Toolbox. It also shows that `go.m` calls `mobius.m`, which is in the current folder.

Dependency Report

The Dependency Report shows dependencies among MATLAB files in a folder ([Learn More](#)).

Show child functions Show parent functions (current folder only)
 Show subfunctions

Built-in functions and files in toolbox/matlab are not shown

Report for Folder I:\my_MATLAB_files

MATLAB File List	Children (called functions)
chirpy	toolbox : \images\images\erode.m toolbox : \shared\siglib\chirp.m toolbox : \signal\signal\specgram.m
collatz	
collatzall	subfunction : collatzplot_new
collatzplot	current dir : collatz
collatzplot_new	current dir : collatz
go	current dir : mobius
mobius	

The Dependency Report includes the following:

- MATLAB File List

The list of files in the folder on which you ran the Dependency Report. Click a link in this column to open the file in the Editor.

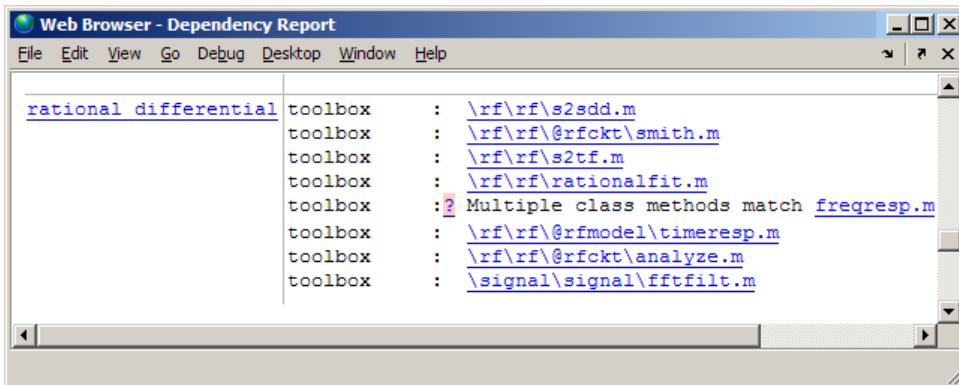
- Children

The function or functions called by the MATLAB file.

Click a link in this column to open the MATLAB file listed in the same row, and go to the first reference to the called function. For instance, suppose your Dependency Report appears as shown in the previous image. Clicking `\images\images\erode.m` opens `chirpy.m` and places the cursor at the first line that references `erode`. In other words, it does not open `erode.m`.

- Multiple class methods

Because the report is a static analysis, it cannot determine run-time data types and, therefore, cannot identify the particular class methods required by a file. If multiple class methods match a referenced method, the Dependency Report inserts a question mark link next to the file name. The question mark appears in the following image.



Click the question mark link to list the class methods with the specified name that MATLAB 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 MATLAB will use. MATLAB determines which method to use depending on the object that the program calls at run time.

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\ctrlobolete\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
```

Identifying How Much of a File Ran When Profiled

When you run the Profiler on a file, some code might not run, such as a block containing an if statement.

To determine how much of a file ran when you profiled it, run the Coverage Report:

- 1 On the MATLAB desktop, select **Desktop > Profiler**.
- 2 Profile a MATLAB code file in the Profiler.


For detailed instructions, see “Profiling for Improving Performance” on page 9-27.

- 3 Ensure the Profiler is not currently profiling.

The Profiler displays a **Start Profiling** button when the Profiler is not running. If the Profiler is running, then do one of the following:

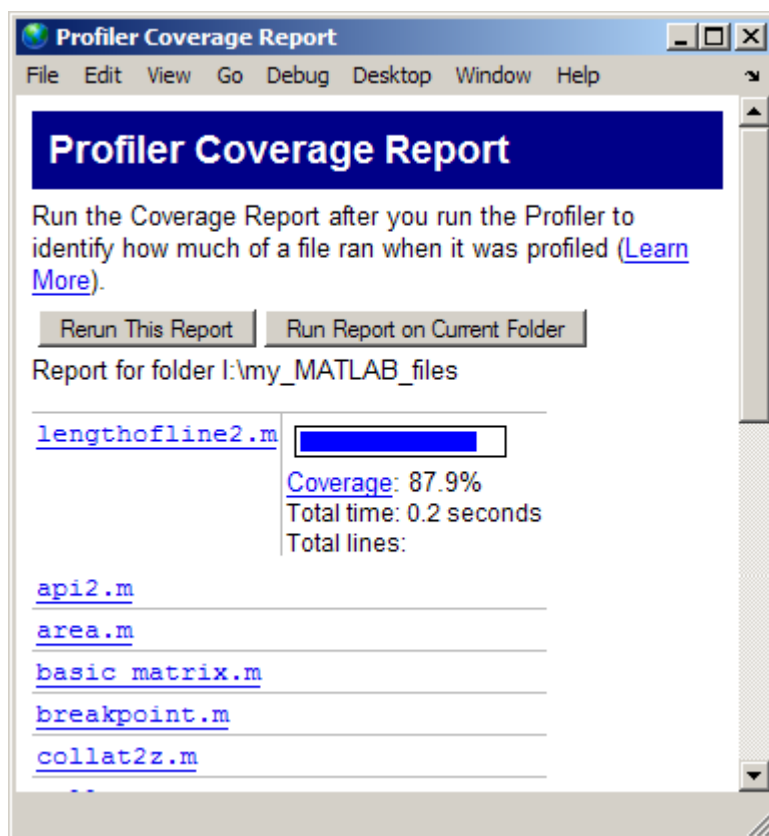
- Wait for it to stop profiling before proceeding to step 4.
- Click **Stop Profiling**.

The **Stop Profiling** button is available only when the Profiler is running.

- 4** Select **Desktop > Current Folder** and navigate to the folder containing the file for which you ran the Profiler.
- 5** On the Current Folder browser toolbar, click the **Actions** button , and then select **Reports > Coverage Report**.

The Profiler Coverage Report appears, providing a summary of coverage for the file you profiled. In the image that follows, the profiled file is `lengthofline2.m`.

- 6** Click the **Coverage** link to see the Profile Detail Report for the file.



Using the MATLAB Code Analyzer Report

In this section...

“Running the Code Analyzer Report” on page 9-22

“Changing Code Based on Code Analyzer Messages” on page 9-24

“Other Ways to Access Code Analyzer Messages” on page 9-25


Running the Code Analyzer Report

The Code Analyzer Report displays potential errors and problems, as well as opportunities for improvement in your code through messages. For example, a common message indicates that a variable `foo` might be unused.

To run the Code Analyzer Report:

- 1 In the Current Folder browser, navigate to the folder that contains the files you want to check. To use the example shown in this documentation, `lengthoffline.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. Then, make that folder the current MATLAB folder. This example saves the file in `C:\my_MATLAB_files`.
- 3 In the Current Folder browser toolbar, click the **Actions** button , and then select **Reports > Code Analyzer Report**.

The report displays in the MATLAB Web Browser, showing those files identified as having potential problems or opportunities for improvement.

Web Browser - Code Analyzer Report

File Edit View Go Debug Desktop Window Help

Code Analyzer Report

This report displays potential errors and problems, as well as opportunities to improve your MATLAB programs ([Learn More](#)).

[Rerun This Report](#) [Run Report on Current Folder](#)

Report for folder C:\my_MATLAB_files

lengthofline 18 messages	<p>22: The value assigned to variable 'nothandle' might be unused.</p> <p>23: NUMEL(x) is usually faster than PROD(SIZE(x)).</p> <p>24: The variable 'notline' appears to change size on every loop iteration. Consider preallocating for speed.</p> <p>24: Use STRCMP1(str1,str2) instead of using UPPER/LOWER in a call to STRCMP.</p> <p>28: NUMEL(x) is usually faster than PROD(SIZE(x)).</p> <p>34: The variable 'data' appears to change size on every loop iteration. Consider preallocating for speed.</p> <p>34: Use dynamic fieldnames with</p>
---	--

Line number and message

- 4 For each message in the report, review the suggestion and your code. Click the line number to open the file in the Editor at that line, and change the file based on the message. Use the following general advice:

- If you are unsure what a message means or what to change in the code, click the link in the message if one appears. For details, see “Avoid Mistakes While Editing Code” on page 8-84.
 - If the message does not contain a link, and you are unsure what a message means or what to do, search for related topics in the Help browser. For examples of messages and what to do about them, including specific changes to make for the example, `lengthoffline.m`, see “Changing Code Based on Code Analyzer Messages” on page 9-24.
 - The messages do not provide perfect information about every situation and in some cases, you might not want to change anything based on the message. For details, see “Understand the Limitations of Code Analysis” on page 8-106.
 - If there are certain messages or types of messages you do not want to see, you can suppress them. For details, see “Adjust Code Analyzer Message Indicators and Messages” on page 8-100.
- 5 After modifying it, save the 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 Text Files” on page 6-53.
 - 6 Run and debug the file or files again to be sure that you have not introduced any inadvertent errors.
 - 7 If the report is displaying, click **Rerun This Report** to update the report based on the changes you made to the file. Ensure that the messages are gone, based on the changes you made to the files.

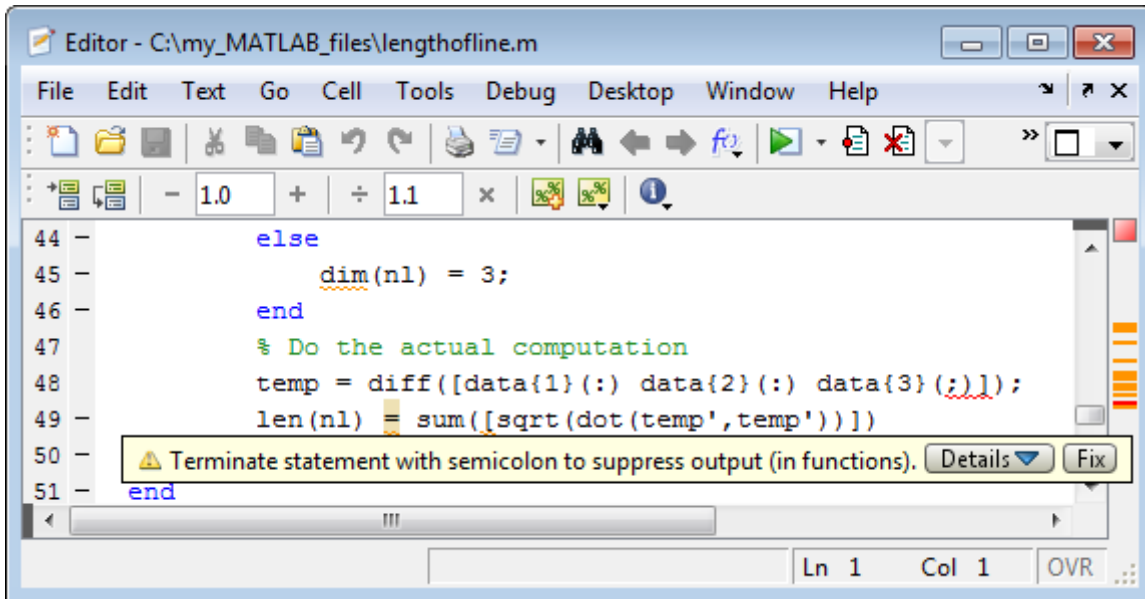
Changing Code Based on Code Analyzer Messages

For information on how to correct the potential problems presented in Code Analyzer messages, use the following resources:

- Open the file in the Editor and click the **Details** button in the tooltip, as shown in the image following this list. An extended message opens. However, not all messages have extended messages.

- Use the Help browser **Search** and **Index** panes to find documentation about terms presented in the messages.

The following image shows a tooltip with a **Details** button. The orange *line* under the equals (=) sign indicates a tooltip displays if you hover over the equals sign. The orange *highlighting* indicates that an automatic fix is available.



Other techniques to help you identify problems in and improve your code are in these topics:

Other Ways to Access Code Analyzer Messages

You can get Code Analyzer messages using any of the following methods. Each provides the same messages, but in a different format:

- Access the Code Analyzer Report for a file from the Editor **Tools** menu or from the Profiler detail report.
- Run the `checkcode` function, which analyzes the specified file and displays messages in the Command Window.

- Run the `mlintrpt` function, which runs `checkcode` and displays the messages in the Web Browser.
- Use automatic code checking while you work on a file in the Editor. See “Automatically Check Code in the Editor — Code Analyzer” on page 8-91.

Profiling for Improving Performance

In this section...

“What Is Profiling?” on page 9-27

“Profiling Process and Guidelines” on page 9-28

“Using the Profiler” on page 9-29

“Profile Summary Report” on page 9-36

“Profile Detail Report” on page 9-38

“The profile Function” on page 9-46

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. Then, look for ways to minimize their use and thus improve performance. It is often helpful to decide whether the number of times the code calls a particular function is reasonable. Because programs often have several layers, your code might 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 profiling spends most of its time on calls to a few built-in functions, you have probably optimized the code as much as you can.

Note When using the Parallel Computing Toolbox™ software, you can use the parallel profiler to profile parallel jobs. See “Profiling Parallel Code” for details.

Profiling Process and Guidelines

Here is a general process you can follow to use the Profiler to improve performance in your code. This section includes the following topics:

- Using Profiling as a Debugging Tool
- “Using Profiling to Understand an Unfamiliar File” on page 9-29

Tip 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 are called most frequently. See “Profile Summary Report” on page 9-36 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 9-38 for more information.

Consider keeping a copy of your first detail report as a basis for comparison. After you change the function file, run the Profiler again and compare the reports.

- 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 called only 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 identical 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 code.

For example, if a particular section of a file did not run, you can look at the detail reports to see what lines did run. The detail report 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 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**. Using **Ctrl+C** can be useful when a file is taking much more time to run than expected.

Using Profiling to Understand an Unfamiliar File

For a lengthy MATLAB code file that you did not create, or with which you are unfamiliar, use the Profiler to see how the file actually works. Use the Profiler detail reports to see the lines called.

If there is an existing GUI tool (or file) like 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 a file is spending its time. This section covers:

- “Opening the Profiler” on page 9-30
- “Running the Profiler” on page 9-30
- “Profiling a Graphical User Interface” on page 9-35
- “Profiling Statements from the Command Window” on page 9-35
- “Changing Fonts for the Profiler” on page 9-35

For information about the reports generated by the Profiler, see “Profile Summary Report” on page 9-36 and “Profile Detail Report” on page 9-38.

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 then select **Profile Code**.
- Type `profile viewer` in the Command Window:

Running the Profiler

To profile a MATLAB code file or a line of code:

- 1 If your system uses Intel® multi-core chips, consider restricting the active number of CPUs to one.

See one of the following for details:

- “Intel Multi-Core Processors — Setting for Most Accurate Profiling on Windows Systems” on page 9-32
- “Intel Multi-Core Processors — Setting for Most Accurate Profiling on Linux Systems” on page 9-33

2 In the Command Window, type `profile viewer`.

3 Do one of the following in the Profiler:

- For a statement you have not profiled in the current MATLAB session:

In the **Run this code** field, type the statement you want to run.

For example, you can run the Lotka-Volterra demo, which is provided with MATLAB demos (`lotkademo`):

```
[t,y] = ode23('lotka',[0 2],[20;20])
```

- For a statement you previously profiled in the current MATLAB session:

1 Select the statement from the list box—MATLAB automatically starts profiling the code.

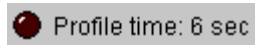
2 Skip to step 5.

4 Click **Start Profiling**.

While the Profiler is running, the **Profile time** indicator is green and the number of seconds it reports increases. The **Profile time** indicator appears at the top right of the Profiler window.



When the Profiler finishes, the **Profile time** indicator becomes dark red and shows the length of time the Profiler ran. The statements you profiled display as having been executed in the Command Window.



This time 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 very different from what you expected (for example, hundreds of seconds for a simple statement), you might have had profiling on longer than you realize. 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 9-50.

- 5 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 9-36.
- 6 Reset the number of active CPUs to the original setting if you restricted the number in step 1.

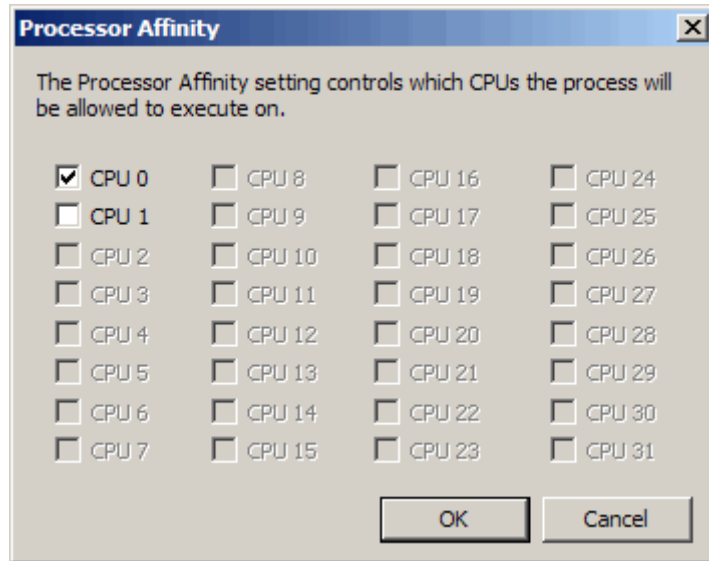
Intel Multi-Core Processors – Setting for Most Accurate Profiling on Windows Systems. If your system uses Intel multi-core chips, and you plan to profile using CPU time, set the number of active CPUs to one before you start profiling. This results in the most accurate and efficient profiling.

- 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 like 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 so is to change the Profiler timer setting to `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. Rerun the preceding steps, and restore the original selections in the Processor Affinity dialog box in step 3.

Intel Multi-Core Processors – Setting for Most Accurate Profiling on Linux Systems. If your system uses Intel multi-core 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.

For example, to set the processor affinity to one you can use the Linux `taskset` command, as follows:

- 1** Get the process ID (PID) of the currently running MATLAB instance:

```
ps -C MATLAB
PID  TTY  TIME      CMD
8745 pts/1 00:00:50  MATLAB
```

The PID in this example is 8745.

- 2** Call the Linux `taskset` command to get the current number of active CPUs for the MATLAB process:

```
taskset -pc 8745
pid 8745's current affinity list: 0-3
```

The `-p` option specifies that `taskset` operate on an existing PID, instead of creating a new task. The `-c` option lists the processor numbers.

- 3** Call the Linux `taskset` command again — this time to set the processor affinity for the MATLAB process to one CPU (that is, CPU #0):

```
taskset -pc 0 8745
pid 8745's current affinity list: 0-3
pid 8745's  new affinity list: 0
```

For more information on the syntax of `taskset`, execute `man taskset` from a Linux terminal.

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 to `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 its original setting when you finish profiling. Rerun the preceding steps, and then restore the original number of CPUs returned in step 2.

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:

- 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 **Stop Profiling**, step 1, until after you start the graphical interface.)
- 3** Use the graphical interface. When you finish, 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:

- 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:

- 1** Select **File > Preferences > Fonts** to open the Font Preferences dialog box.
- 2** 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**. Be aware 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 like one of the following, depending on whether MATLAB can determine the profiling overhead:

- Self time is the time spent in a function excluding:
 - The time spent in its child functions
 - Most of the overhead resulting from the process of profiling

In the present run, self time excludes 0.240 secs 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.

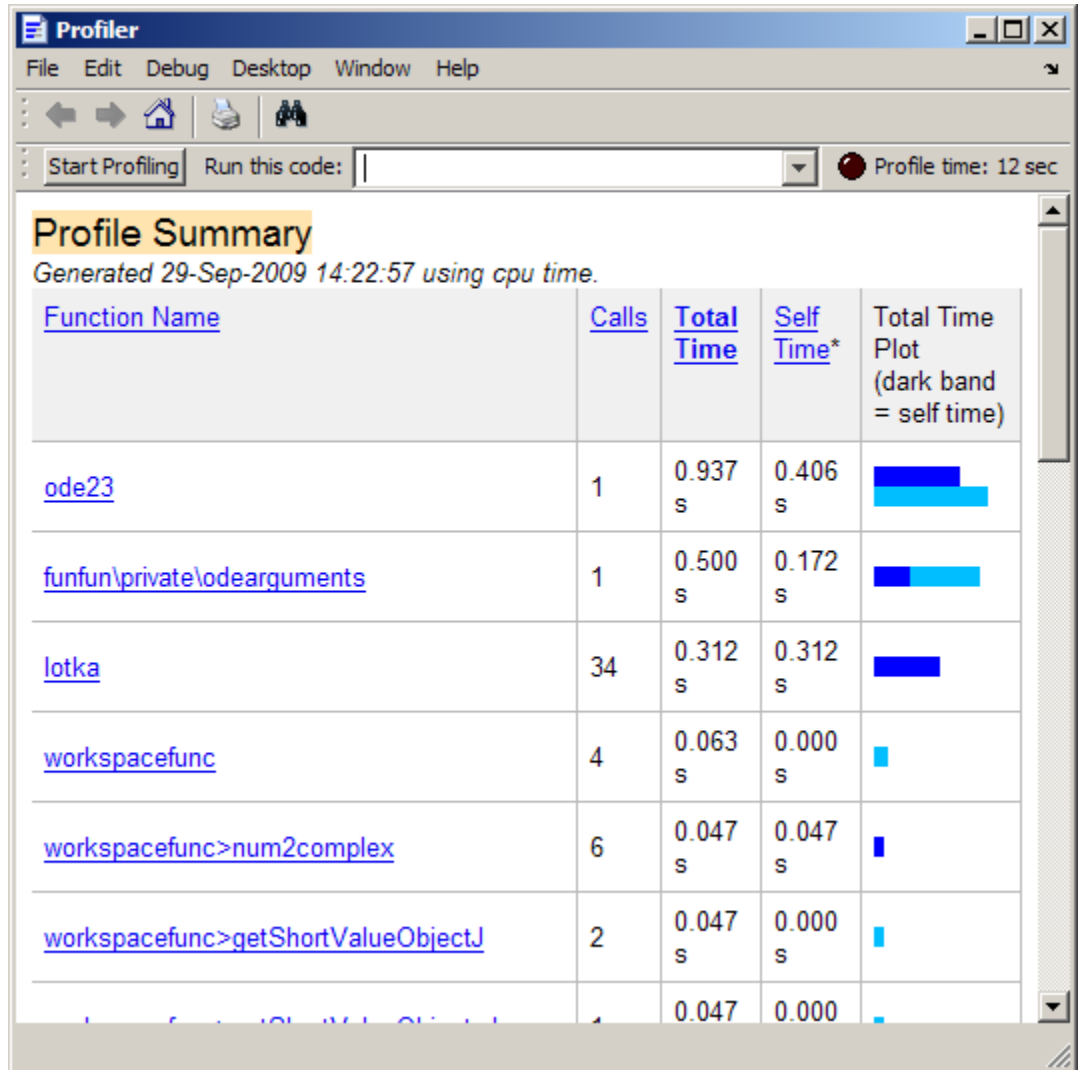
To sort the functions by this time value, click the **Self Time** link at the top of the column.

- **Total Time Plot** — Graphic display showing self time compared to total time.

The following is an image of the summary report for the Lotka-Volterra model used in “Example: Using the profile Function” on page 9-47.


In the summary report, you can:

- Print it, by clicking the Print button .
- Get more detailed information about a particular function by clicking its name in the **Function Name** column. See “Profile Detail Report” on page 9-38 for more information.
- Sort by a given column by clicking the name of the column. For example, click [Function Name](#) to sort by the names of the functions included in the summary report.



Profile Detail Report

The Profile Detail report shows profiling results for a selected function that was called during profiling. A Profile Detail report has seven sections. The topics that follow describe each section. 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 topics provide details about opening and using a Profile Detail Report:

- “Opening the Profile Detail Report” on page 9-39
- “Controlling the Contents of the Detail Report Display” on page 9-39
- “Profile Detail Report Header” on page 9-40
- “Parent Functions” on page 9-41
- “Busy Lines” on page 9-41
- “Child Functions” on page 9-42
- “Code Analyzer Results” on page 9-43
- “File Coverage” on page 9-44
- “Function Listing” on page 9-45

Opening the Profile Detail Report

To open the Profile Detail Report:

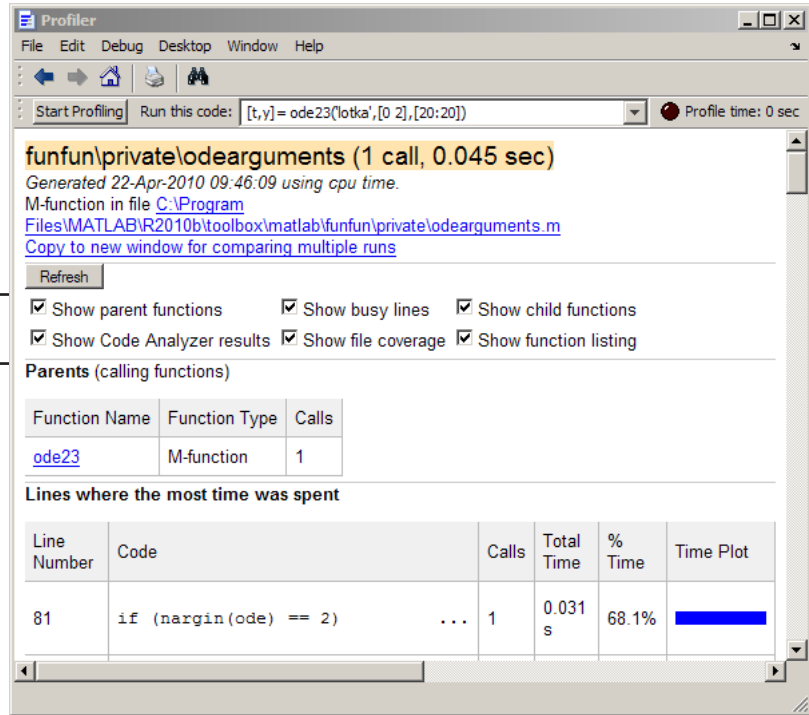
- 1 Create a Profile Summary report, as described in “Using the Profiler” on page 9-29.
- 2 Click a function name listed in the Profile Summary report.

Controlling the Contents of the Detail Report Display

To specify which sections the Profile Detail Report includes:

- 1 Select report options from the set of check boxes at the top of the report.
- 2 Click the **Refresh** button.

Report Options



Profile Detail Report Header

The detail report header includes:

- The name of the function profiled
- The number of times the profiled function was called in the parent function
- The amount of time the profiled function used
- A link that opens the function in your default text editor
- A link that copies the report to a separate window

Creating a copy of the report is helpful when you change the file, run the Profiler for the updated file, and compare the Profile Detail reports for the two runs. Do not change files provided with products from MathWorks, that is, files in the `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. Click the name of a parent function to open a Detail Report for that parent function.

Show parent functions Show busy lines Show child functions
 Show Code Analyzer results Show file coverage Show function listing

Parents (calling functions)

Function Name	Function Type	Calls
ode23	M-function	1







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.

Refresh




Show parent functions Show busy lines Show child functions
 Show Code Analyzer results Show file coverage Show function listing

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
110	<code>f0 = feval(ode,t0,y0,args{:});...</code>	1	0.482 s	58.5%	
81	<code>if (nargin(ode) == 2) ...</code>	1	0.110 s	13.3%	
148	<code>rtol = odeget(options,'RelTol'...</code>	1	0.063 s	7.6%	
184	<code>htry = abs(odeget(options,'Ini...</code>	1	0.032 s	3.9%	
105	<code>if any(tdir*diff(tspan) <=...</code>	1	0.032 s	3.9%	
All other lines			0.107 s	12.9%	
Totals			0.825 s	100%	

Child Functions

To include the **Children** section of the detail report, select the **Show child functions** check box. This section of the report lists all the functions called by the profiled function. If the called function is a MATLAB code 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	0.466 s	56.5%	
odeget	M-function	5	0.032 s	3.9%	
double_superiorfloat	M-function	1	0 s	0%	
Self time (built-ins, overhead, etc.)			0.327 s	39.6%	
Totals			0.825 s	100%	

Code Analyzer Results

To include the **Code Analyzer results** section in the detail report display, select the **Show results** check box. This section of the report provides information about problems and potential improvements for the function. For more information, see “Using the MATLAB Code Analyzer Report” on page 9-22.

Show parent functions Show busy lines Show child functions
 Show Code Analyzer results Show file coverage Show function listing

M-Lint results

Line number	Message
52	EXIST with two input arguments is generally faster and clearer than with one input argument.
80	EXIST with two input arguments is generally faster and clearer than with one input argument.
90	The value assigned to variable 'ME' might be unused.

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.

Refresh

Show parent functions Show busy lines Show child functions
 Show Code Analyzer results Show file coverage Show function listing

Coverage results
[\[Show coverage for parent directory \]](#)

Total lines in function	190
Non-code lines (comments, blank lines)	57
Code lines (lines that can run)	133
Code lines that did run	53
Code lines that did not run	80
Coverage (did run/can run)	39.85 %

Function Listing

To include the **Function listing** section in the detail report display, select the **Show function listing** check box. If the file is a MATLAB code file, the Profile Detail report includes three columns:

- The first column lists the execution time for each line.
- The second column lists the number of times the line was called
- The third column specifies the source code for the function.

In the function listing, the color of the text indicates the following:

- Green — Comment lines
- Black — Lines of code that executed
- Gray — Lines of code that did not execute

By default, the Profile Detail report highlights lines of code with the longest execution time. The darker the highlighting, the longer the line of code took to execute.

To change the lines of code highlighted based on other criteria, use the drop-down menu in this section of the detail report. The color of the highlighting changes, depending on the drop-down option you choose. You can choose to highlight lines of code called the most, lines of code that were (or were not) executed, or lines called out by the Code Analyzer. Or, you can turn off highlighting by selecting none.

The following image shows that lines highlight in blue when you select coverage from the drop-down menu.

Refresh

Show parent functions Show busy lines Show child functions
 Show Code Analyzer results Show file coverage Show function listing

Function listing
Color highlight code according to coverage

time	calls	line
		1 func ode15i (span, nts pan , next, t0, tfinal, tdir, y0, f0, args, odeFcn, ..
		2 s, threshold, rtol, normcontrol, normy, hmax, htry, htspan, ..
		3 pe] = ...
		4 odearguments(FcnHandlesUsed, solver, ode, tspan, y0, options, extras)
		5 %ODEARGUMENTS Helper function that processes arguments for all ODE solvers.
		6 %
		7 % See also ODE113, ODE15I, ODE15S, ODE23, ODE23S, ODE23T, ODE23TB, ODE45.
		8
		9 % Mike Karr, Jacek Kierzenka
		10 % Copyright 1984-2008 The MathWorks, Inc.
		11 % \$Revision: 1.12.4.12 \$ \$Date: 2009/03/16 22:17:47 \$
		12
		1 13 if strcmp(solver, 'ode15i')
		14 FcnHandlesUsed = true; % no MATLAB v. 5 legacy for ODE15I
		15 end

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, use the `profile` function to specify that 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” on page 9-47
- “Accessing profile Function Results” on page 9-48

- “Saving profile Function Reports” on page 9-50
- “Using the profile Function to Change the Time Type Used by the Profiler” on page 9-50

Example: Using the profile Function

This example demonstrates how to run `profile`:

- 1** To start `profile`, type the following in the Command Window:

```
profile on
```

- 2** Execute a MATLAB code file. This example runs the Lotka-Volterra predator-prey population model. For more information about this model, type `lotkadem`, 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 files you run. It will add these new statistics to those statistics 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 the following 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 a MATLAB code file. This example runs the Lotka-Volterra predator-prey population model. For more information about this model, type `lotkademmo`, 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 a MATLAB function (M-function), MATLAB 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)
```

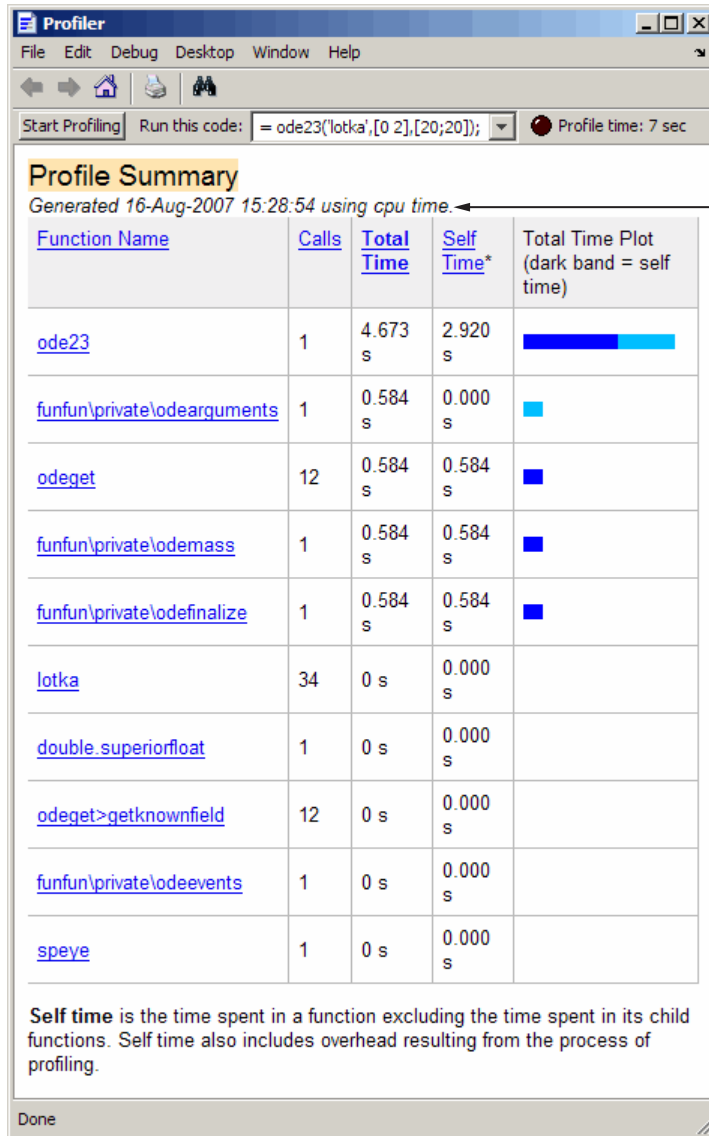
By default, `profsave` puts these HTML files in a subfolder of the current folder named `profile_results`. You can specify another folder name as an optional second argument to `profsave`.

```
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.



Generated 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
```

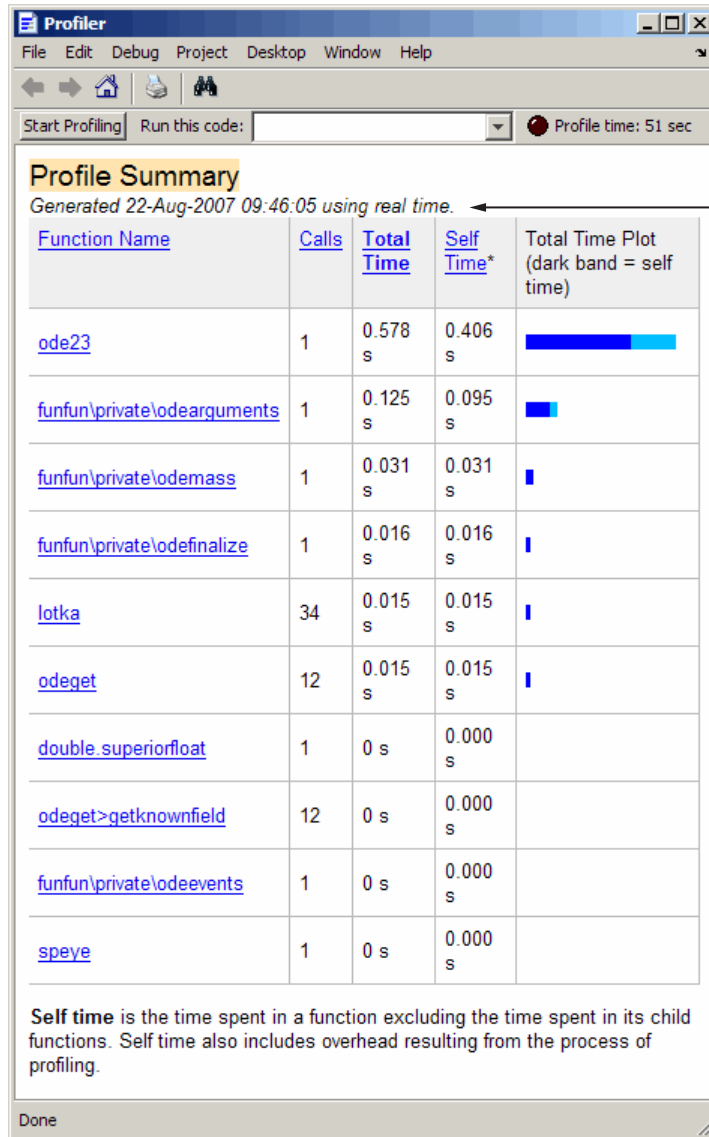
3 Run the file that you want to profile. This example runs the Lotka-Volterra predator-prey population model.

```
[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 it is using real time, as shown in the following image.



Generated using real time

- 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
```

Publishing MATLAB Code

MATLAB software lets you mark up MATLAB code and publish it to various output file formats.

- “Overview of Publishing MATLAB Code” on page 10-2
- “Mark Up MATLAB Comments for Publishing” on page 10-17
- “Mark Up MATLAB Code for Publishing” on page 10-62
- “Specify Output Preferences for Publishing” on page 10-66
- “Summary of Options for Presenting Your Code to Others” on page 10-109

Overview of Publishing MATLAB Code

In this section...

“What Is Meant by Publishing MATLAB Code?” on page 10-2

“Using Code Cells” on page 10-2

“Process for Publishing MATLAB Code” on page 10-3

“Example of Published MATLAB Code” on page 10-4

“Adding the Markup for the Example” on page 10-10

What Is Meant by Publishing MATLAB Code?

MATLAB software enables you to publish your MATLAB code quickly, so you can describe and share your code with others, even if they do not have MATLAB software. 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.

You can include the following within the file you want to publish:

- Commentary on the code, including bulleted and numbered lists, bold and monospace font, preformatted text, LaTeX equations, and so on
- MATLAB code
- Results of running the code, including output to the Command Window and figures created or modified by the code

If you have an active Internet connection, you can watch the Publishing MATLAB Code from the Editor video demo for an overview of the major publishing features using cells with text markup.

Using Code Cells

After you write and debug MATLAB code, if you insert code cells and commentary using text markup features, you can publish the code as a document. A code cell is a section of MATLAB code (see “What Are Code Cells?” on page 8-58). For the purposes of publishing, a code cell can be either of the following, or both:

- A section of the code that you want to present as a titled subsection within the output
- 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)

Any code cell features that you use for evaluating and improving your code, as described in “Evaluate Subsections of Files Using Code Cells” on page 8-58, you also can use for publishing purposes. However, to display comments in the output document, those comments must appear at the start of a code cell, before any executable code. This can require you to change code cells that you inserted for evaluating subsections of code. If you do so, be aware that this changes the cells for evaluation purposes, as well.

“Example of Published MATLAB Code” on page 10-4 shows how the code cells and formatted comments appear when you publish MATLAB code.

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, you can do both.

Note Cell mode is supported for use with files containing MATLAB code only. It is not intended for use with other text files.

Process for Publishing MATLAB Code

The overall process to publish a MATLAB code file using code cell features in the Editor is as follows:

- 1 Open your file in the Editor.
- 2 Select **Cell > Insert Text Markup** as described in “Mark Up MATLAB Comments for Publishing” on page 10-17.

This enables you to specify how MATLAB comments appear in the output. For example, you can specify that comments appear as bold or monospaced text.

- 3 To publish the file, do one of the following, as described in “Specify Output Preferences for Publishing” on page 10-66:

- To publish the file with default publishing properties, select **File > Publish *file name***. When you use this method, MATLAB publishes the file to HTML in an `/html` subfolder of the folder that contains the file you are publishing. However, if you previously specified custom property values, as described in the next list item, publishing uses the last configuration you specified. The output file formats and folder can be different from the default.
- 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 output.

Example of Published MATLAB Code

This example demonstrates how MATLAB code appears when published. It shows how the file appears before and after text markup is added to code cells to achieve the desired results. This section contains the following topics:

- “Sample File Before Adding Markup” on page 10-4
- “Published Sample File Before Adding Markup” on page 10-5
- “Published Sample File After Adding Markup” on page 10-7

For detailed information on inserting text markup, see “Mark Up MATLAB Comments for Publishing” on page 10-17.

Sample File Before Adding Markup

```
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

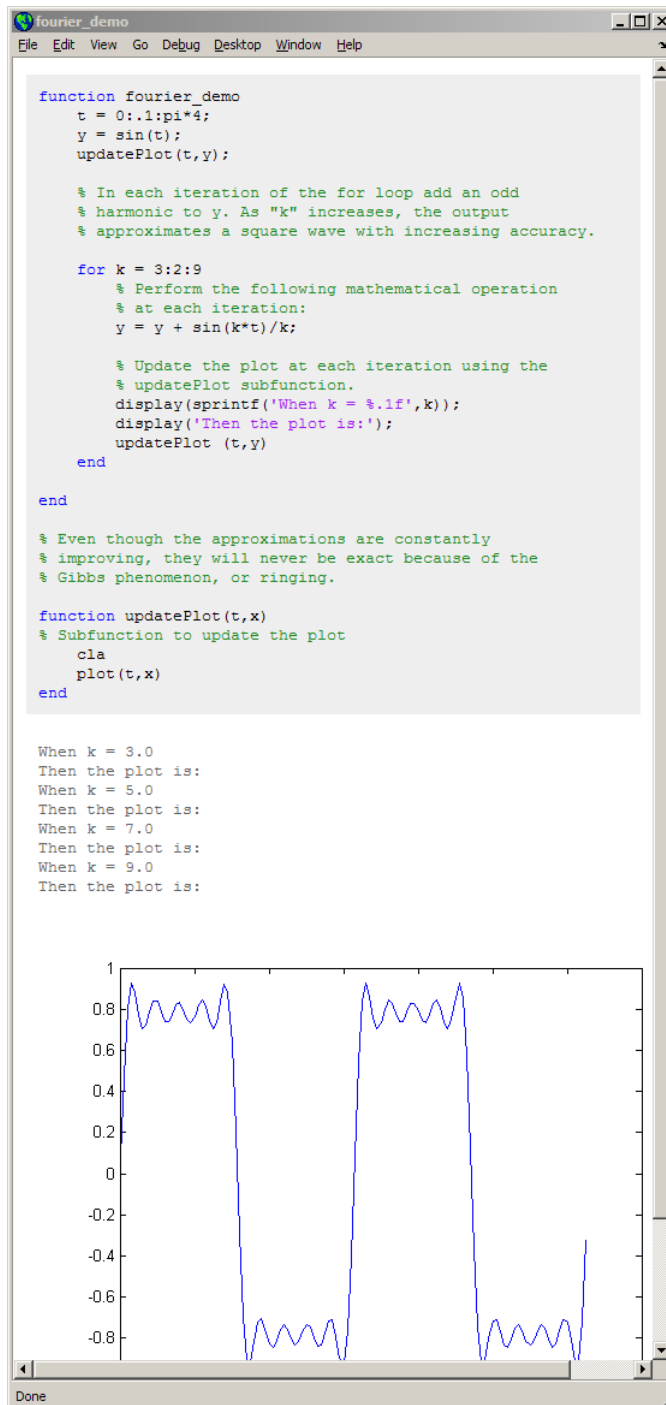
end

% 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 File Before Adding Markup

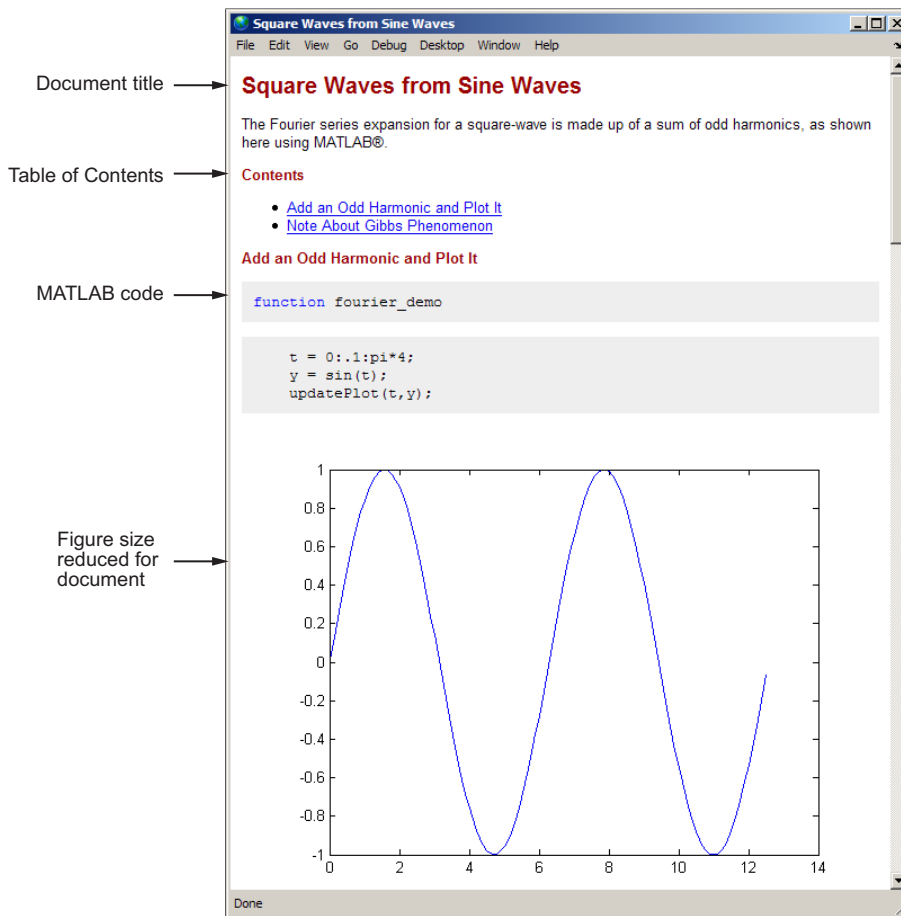
Before you add text markup and cell breaks, publishing `fourier_demo.m` includes the last plot generated by the for loop, but otherwise, has little effect. For example, if you select **File > Publish fourier_demo.m**, the output, as shown in the following figure, are of limited use.



Published Sample File After Adding Markup

If you add comments for clarity, apply text markup, and insert code cell breaks, as described in “Adding the Markup for the Example” on page 10-10, publishing the code transforms the output as shown in the following three figures:

- The first figure shows the top of the output.
- The second figure shows the middle of the output.
- The third figure shows the bottom of the output.



The letter *k* is italic →

Equation in TeX format →

Includes each iteration of the for loop →

The image shows a MATLAB window titled "Square Waves from Sine Waves". The window contains the following text and code:

```

File Edit View Go Debug Desktop Window Help
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;

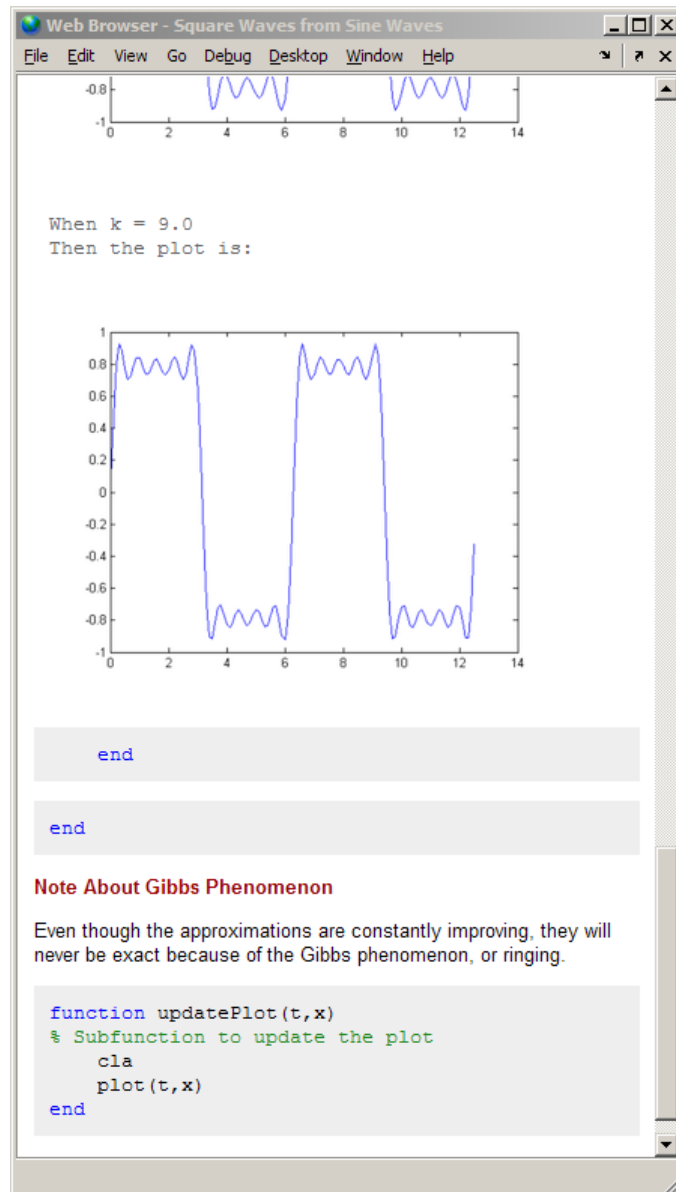
display(sprintf('When k = %.1f',k));
display('Then the plot is:');
updatePlot (t,y)

When k = 3.0
Then the plot is:

[Plot of a square wave approximation with k=3.0]

When k = 5.0
Then the plot is:
    
```

The plot shows a blue line representing a square wave approximation. The x-axis ranges from 0 to 14, and the y-axis ranges from -1 to 1. The plot shows two full cycles of the square wave, with the first cycle from t=0 to t=6 and the second cycle from t=6 to t=12. The plot is a smooth curve that approximates the square wave, with some overshoot and oscillations near the discontinuities.



Adding the Markup for the Example

The following steps apply text markup to the `fourier_demo.m` file. When published to HTML, the output appears as shown in “Published Sample File After Adding Markup” on page 10-7.

For detailed information about each **Cell** menu option, see “Mark Up MATLAB Comments for Publishing” on page 10-17.

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:

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 code 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)” appears as a registered trademark symbol when you publish it.

- 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 cell 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.

When you move from one cell to the next in the file, the highlighting in the file indicates the code cell containing the cursor.

```

1  %% Square Waves from Sine Waves
2  % The Fourier series expansion for a square-wave
3  % made up of a sum of odd harmonics, as shown
4  % using MATLAB(R).
5
6  %% Add an Odd Harmonic and Plot It.
7  function fourier_demo
8      t = 0:.1:pi*4;
9      y = sin(t);
10     updatePlot(t,y);
11
12     % In each iteration of the for loop add an
13     % harmonic to y. As "k" increases, the out


```

- 5 To display the commented text as explanatory text, rather than MATLAB code, insert a cell break before the explanation. That is:
- a Place the cursor at line 12.
 - b Select **Cell > Insert Cell Break**.
- The code 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 appearing enclosed in quotation marks, the letter now appears as `_k_`.

- d** To see the effect, click Publish .

7 MATLAB publishes output generated by code immediately after the end of the cell that contains the code. Therefore, the current cell would cause MATLAB to publish the phrase `When k = n Then the plot is:` four times in succession. In addition, only the final plot generated by the `for` loop would be published.

To include every plot generated by the `for` loop, 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.

```

15 % approximates a square wave with increasing accuracy
16
17 for k = 3:2:9
18     %%
19     % Perform the following mathematical operation
20     % at each iteration:
21     y = y + sin(k*t)/k;
22
23     display(sprintf('When k = %.1f', k));
24     display('Then the plot is:');
25     updatePlot (t, y)
26 end
27
28 end
29
30 % Even though the approximations are constantly

```

- c To see the effect, click Publish .
- 8 Display equations with symbols and Greek characters (such as pi) using the LaTeX format. In this example, to output a comment containing a polished presentation of the equation, $y = y + \sin(k*t)/k$, use text markup as follows:
- Position the cursor at the end of the comment on line 20, % at each iteration.
 - Select **Cell > Insert Text Markup > LaTeX Equation**.

MATLAB inserts the following lines; the second line is a sample equation with text markup applied:

```

%
% $$e^{\pi i} + 1 = 0$$
%

```



The Editor highlights the sample equation, which is the text between the set of two dollar signs (\$\$).

- c Replace the sample equation with the following LaTeX equation:

$$y = y + \frac{\sin(k*t)}{k}$$

The three lines that display the LaTeX equation now appear as follows in the file:

```
%
% $$y = y + \frac{\sin(k*t)}{k} $$
%
```

- d To see the effect, click Publish .
- 9 Reduce the size of the figures in the output 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 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 > Save File and Publish** `fourier_demo`.

When you publish the code to HTML, it appears in the MATLAB Web Browser, as shown in “Published Sample File After Adding Markup” 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 file, `fourier_demo.m`.

See “MATLAB Code After Text Markup” on page 10-15 for the resulting code.

MATLAB Code After Text Markup

After adding text markup, the `fourier_demo.m` file appears as follows. When you publish the file to HTML, it appears as shown in “Published Sample File After Adding Markup” 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 marked up file in the Editor, instead of following the steps in “Adding the Markup for the Example” on page 10-10 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 `fourier_demo.m` in a folder for which you have write permission.

Mark Up MATLAB Comments for Publishing

In this section...

“Publishing Overview” on page 10-17

“Document Titles and Introductory Text” on page 10-18

“Preformatted Text” on page 10-25

“Syntax Highlighted Sample Code” on page 10-27

“Bulleted or Numbered Lists” on page 10-29

“External Graphics” on page 10-32

“HTML Markup” on page 10-35

“LaTeX Markup” on page 10-37

“Inline LaTeX Math Equations” on page 10-40

“LaTeX Display Math” on page 10-41

“Force a Snapshot of Output” on page 10-43

“Bold, Italic, and Monospaced Text” on page 10-44

“Trademark Symbols” on page 10-47

“Hyperlinks” on page 10-48

“Cleaning Up Text Markup Before Publishing MATLAB Files” on page 10-55

“Summary of Markup for Publishing MATLAB Files” on page 10-58

Note Many examples in this section show the effects of publishing to HTML. For information on how to publish to HTML, see “Specify Output Preferences for Publishing” on page 10-66.

Publishing Overview

This section describes how to mark up comments in your MATLAB files so that when you publish the code, it appears polished, rather than as a text file of code. You can single-source your MATLAB code with documentation that describes what the code is doing.

You can mark up MATLAB comments in either of the following ways to specify the appearance of the file when you publish it:

- Use **Cell > Insert Text Markup** menu options to format the comments.

This method automatically inserts the text markup for you.

- Type the markup directly in the comments.

The markup symbols you type are the same as the text markup that results when you use the equivalent menu item. See “Summary of Markup for Publishing MATLAB Files” on page 10-58 for details.

You can use text markup as you create a file, to mark up an existing file, or a combination of the two. When you use the **Cell** menu options, the Editor might insert more comment lines and other markup that you want.

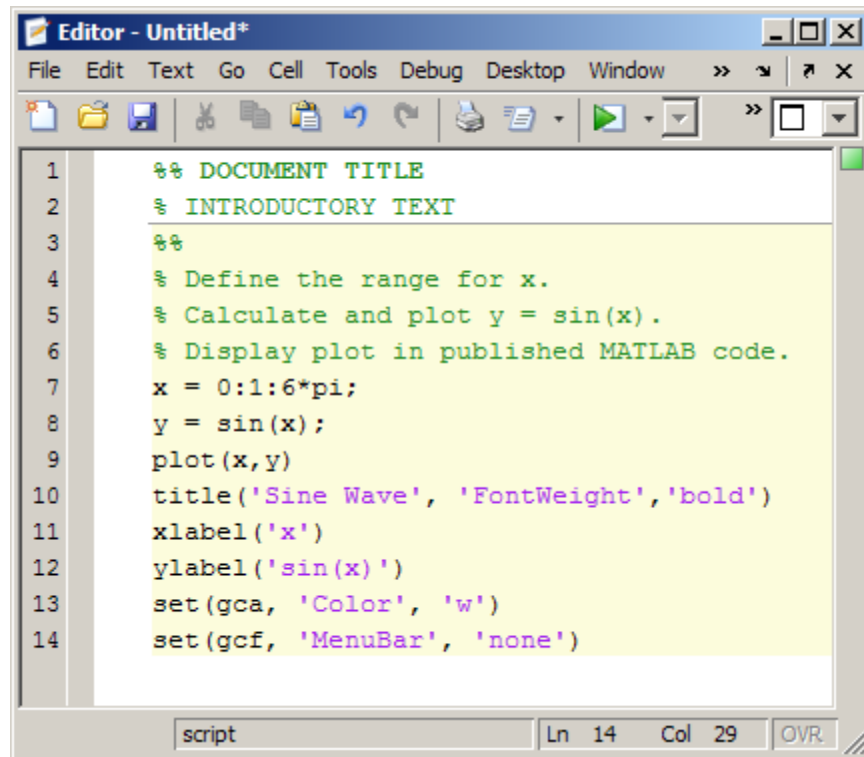
Several examples in the sections that follow use this file, `sine_wave.m`:

```
% Define the range for x.  
% Calculate and plot y = sin(x).  
% Display plot in published MATLAB code.  
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')
```

Document Titles and Introductory Text

To specify a document title and introductory text for a MATLAB file, follow these steps:

- 1** In the Editor, position the cursor anywhere in the 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.



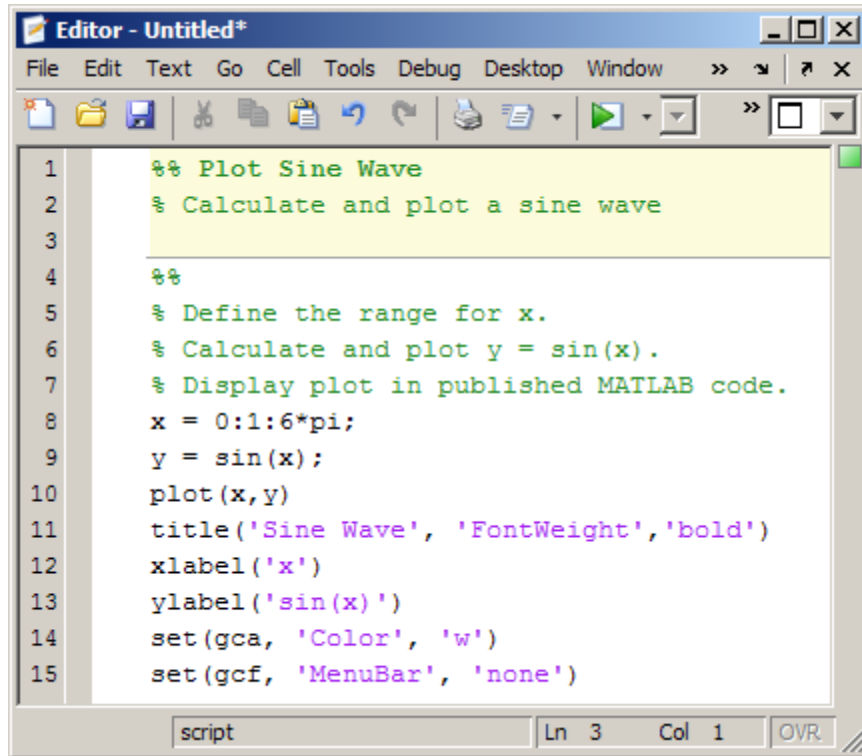
The screenshot shows the MATLAB Editor window titled "Editor - Untitled*". The window contains a script with the following code:

```
1 %% DOCUMENT TITLE
2 % INTRODUCTORY TEXT
3 %%
4 % Define the range for x.
5 % Calculate and plot y = sin(x).
6 % Display plot in published MATLAB code.
7 x = 0:1:6*pi;
8 y = sin(x);
9 plot(x,y)
10 title('Sine Wave', 'FontWeight','bold')
11 xlabel('x')
12 ylabel('sin(x)')
13 set(gca, 'Color', 'w')
14 set(gcf, 'MenuBar', 'none')
```

The status bar at the bottom of the window shows "script", "Ln 14", "Col 29", and "OVR".

- 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 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 resulting file appears as follows.



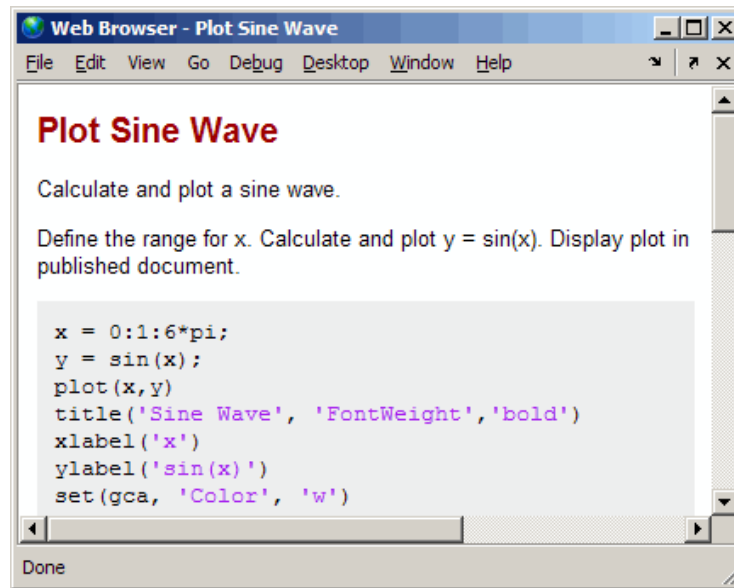
The screenshot shows the MATLAB Editor window titled "Editor - Untitled*". The window has a menu bar with "File", "Edit", "Text", "Go", "Cell", "Tools", "Debug", "Desktop", and "Window". Below the menu bar is a toolbar with various icons for file operations and execution. The main editing area contains the following code:

```
1 %% Plot Sine Wave
2 % Calculate and plot a sine wave
3
4 %%
5 % Define the range for x.
6 % Calculate and plot y = sin(x).
7 % Display plot in published MATLAB code.
8 x = 0:1:6*pi;
9 y = sin(x);
10 plot(x,y)
11 title('Sine Wave', 'FontWeight','bold')
12 xlabel('x')
13 ylabel('sin(x)')
14 set(gca, 'Color', 'w')
15 set(gcf, 'MenuBar', 'none')
```

The status bar at the bottom of the editor shows "script", "Ln 3", "Col 1", and "OVR".

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 file.

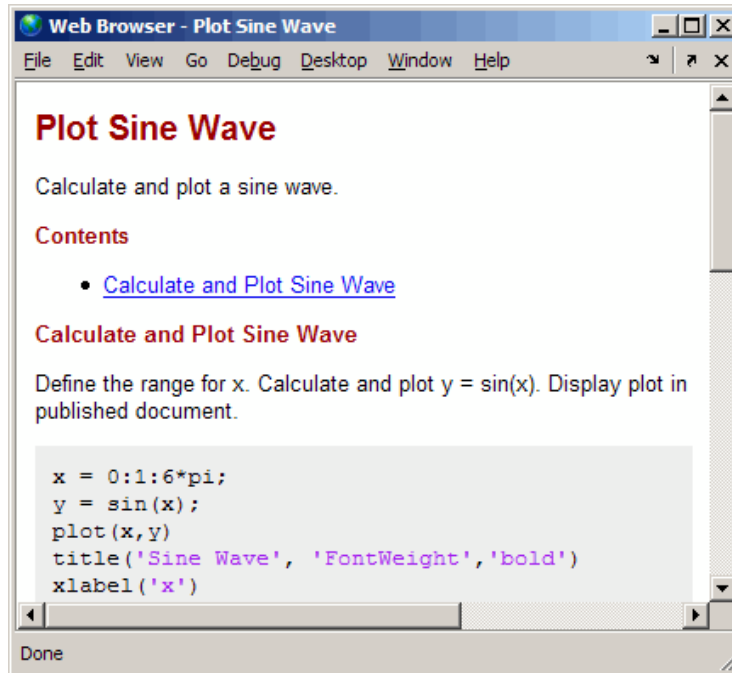
When you publish the file, the document title is displayed as a top-level heading (h1 in HTML), using a large size, bold font. The introductory text appears polished. The following figure shows the file published to HTML and presented in the MATLAB Web Browser.



Title for the New Section that the Editor Inserts with the Document Title

When you follow the steps in the previous section, “Document Titles and Introductory Text” on page 10-18, the first cell, demarcated in the Editor with the horizontal rule followed by a line with double percent signs (%%), does not appear when you publish the file. It is not evident because the section does not have a title.

To provide a title for the section, insert text after the double percent signs on line 4—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 “New Section Titles” on page 10-22.

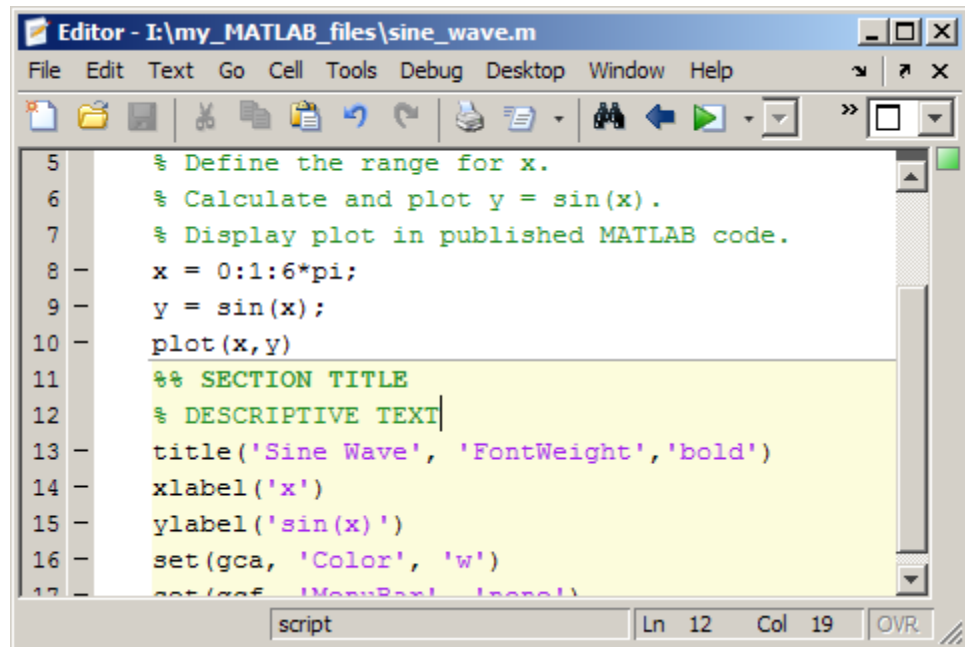
Note You can add any comments in the lines immediately following the title. However, if you want the title to appear as the overall document title, you cannot add any other text before the next cell (a line starting with %%).

New Section Titles

To insert a new section title and descriptive text within a MATLAB file, follow these steps:

- 1 Position the cursor where you want to insert a new cell—before the title function shown in the example, for instance.

- 2 Select **Cell > Insert Text Markup > Section Title with Cell Break**. The file appears as follows.



The screenshot shows the MATLAB Editor window for a file named 'sine_wave.m'. The code is as follows:

```

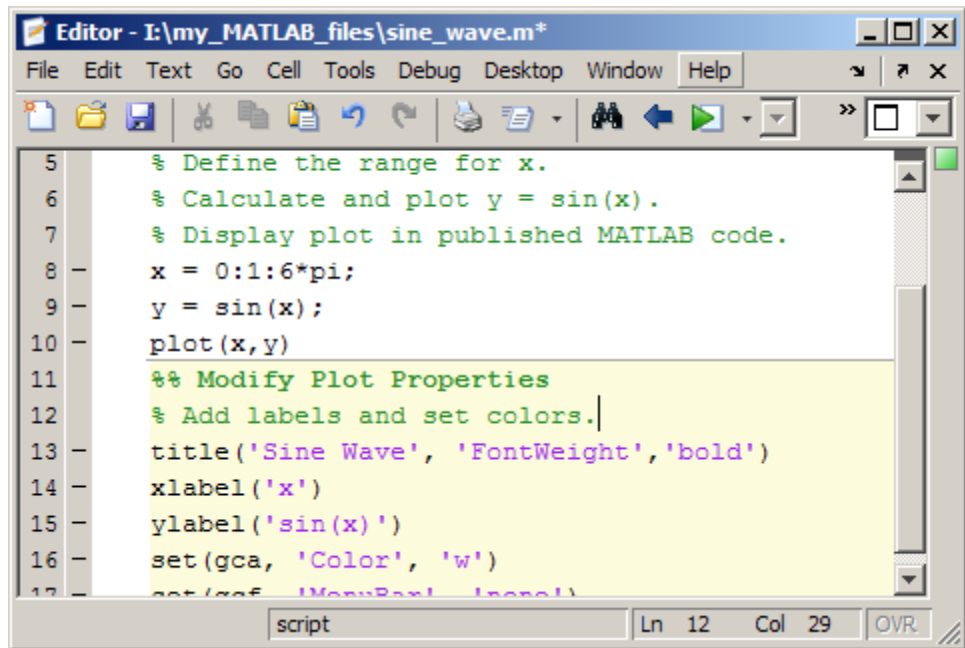
5      % Define the range for x.
6      % Calculate and plot y = sin(x).
7      % Display plot in published MATLAB code.
8      x = 0:1:6*pi;
9      y = sin(x);
10     plot(x,y)
11     %% SECTION TITLE
12     % DESCRIPTIVE TEXT
13     title('Sine Wave', 'FontWeight', 'bold')
14     xlabel('x')
15     ylabel('sin(x)')
16     set(gca, 'Color', 'w')
17     set(gcf, 'MenuBar', 'none')

```

The text from line 11 to 12 is highlighted in yellow. The status bar at the bottom indicates 'script', 'Ln 12', 'Col 19', and 'OVR'.

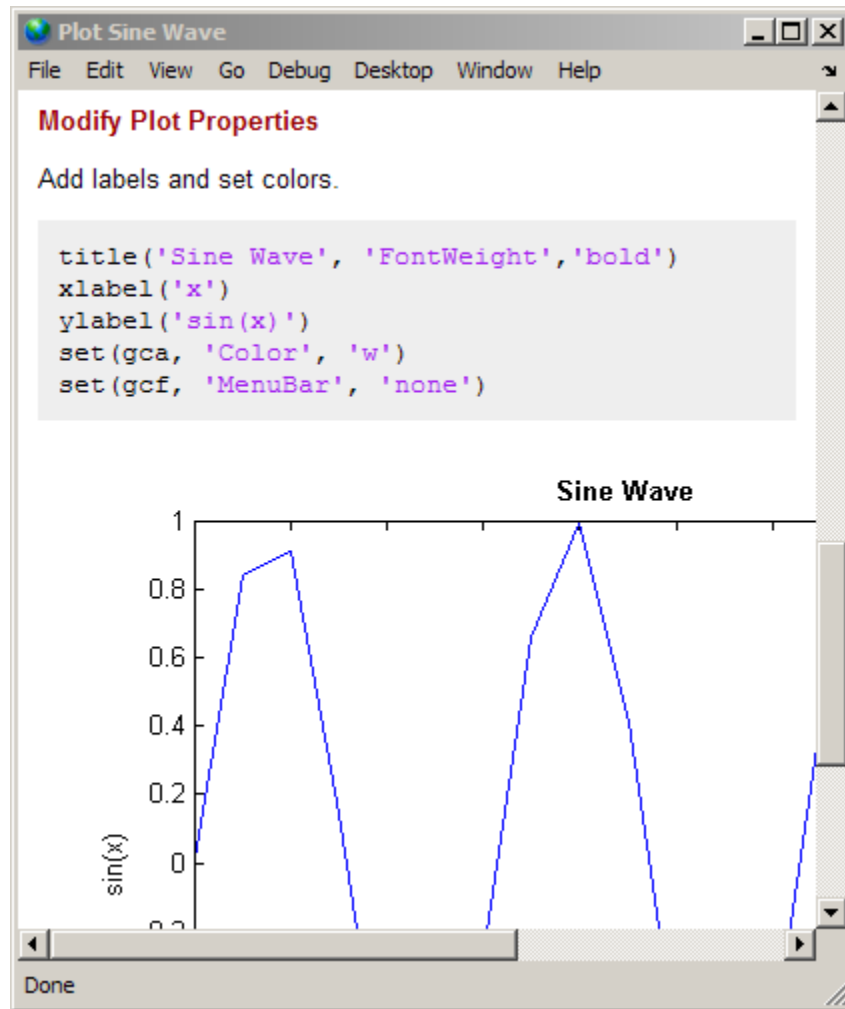
- 3 Replace **SECTION TITLE** with your title—**Modify Plot Properties**, for example.
- 4 Replace **DESCRIPTIVE TEXT** with text that describes the cell—Add labels and set colors., for example.

If you specify the example text suggested in the previous list and “Title for the New Section that the Editor Inserts with the Document Title” on page 10-21, then the resulting file appears as follows.



```
5 % Define the range for x.
6 % Calculate and plot y = sin(x).
7 % Display plot in published MATLAB code.
8 x = 0:1:6*pi;
9 y = sin(x);
10 plot(x,y)
11 %% Modify Plot Properties
12 % Add labels and set colors.
13 title('Sine Wave', 'FontWeight', 'bold')
14 xlabel('x')
15 ylabel('sin(x)')
16 set(gca, 'Color', 'w')
17 set(gcf, 'MenuBar', 'none')
```

When you publish the file to HTML, the section title appears as a heading, using a medium size, bold font. Comments appear as polished text. The following figure shows the results when you publish the updated `sine_wave.m` file to HTML output. Notice that the `Modify Plot Properties` heading is appears as an h2.



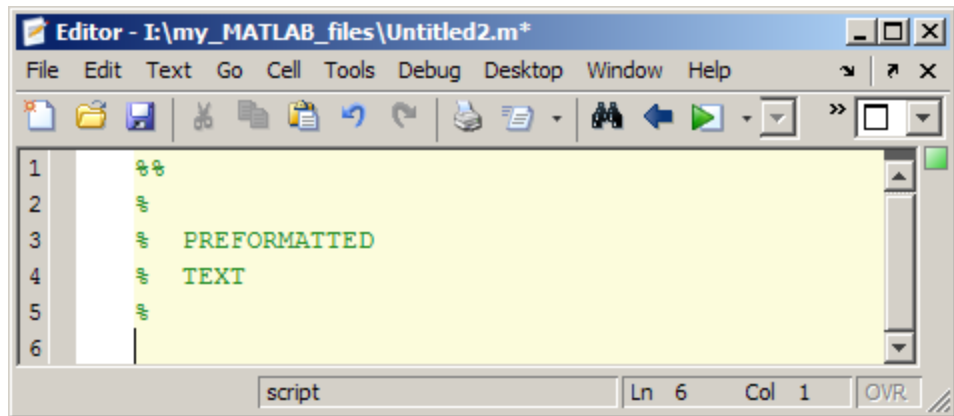
Preformatted Text

MATLAB software enables you to specify preformatted text in a MATLAB file. Preformatted text appears in monospace font, maintains white space, and does not wrap long lines.

To insert preformatted text, follow these steps:

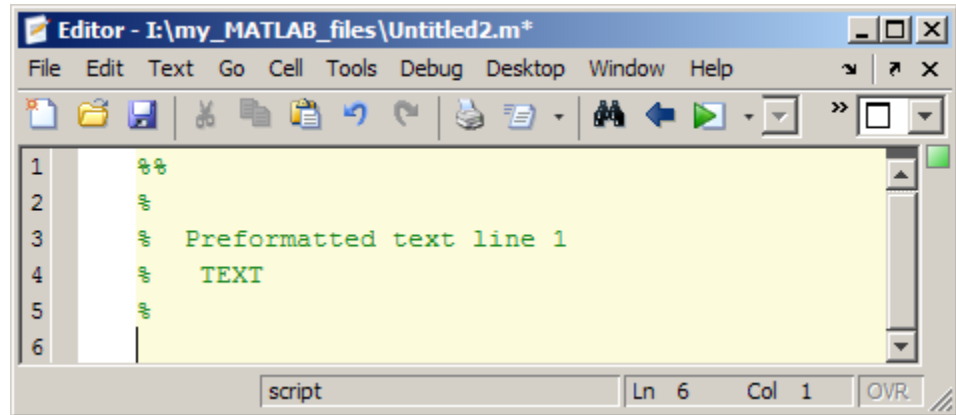
- 1 Position the cursor within the file where you want to insert preformatted text.
- 2 Select **Cell > Insert Text Markup > Preformatted Text**.

Five lines of text are inserted.

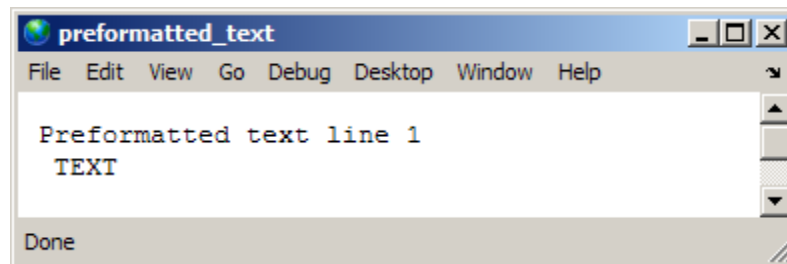


- 3 Being careful not to delete the two blank spaces before the word PREFORMATTED, replace the words PREFORMATTED and TEXT with your text. Your text can include tabs, spaces, and additional comment lines. For example:
 - a Replace PREFORMATTED with Preformatted text line 1.
 - b Insert a tab before TEXT on line 4.

The resulting comments appear as follows.



If you save the file to `preformatted_text.m` and then publish the file to HTML, the output appears as shown in the following figure.



Syntax Highlighted Sample Code

Executable code automatically appears with syntax highlighting in published documents. *Sample code* is code that appears within comments. You can specify that you want sample code to be syntax highlighted.

To include syntax highlighted sample code:

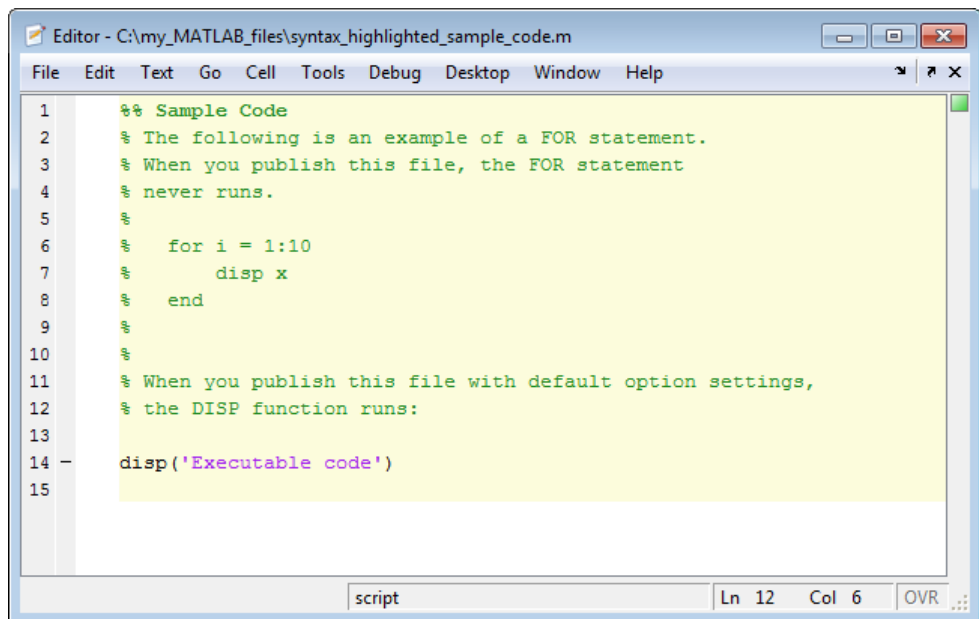
- 1** In the Editor, position the cursor where you want to present sample code.
- 2** Select **Cell > Insert Text Markup > Syntax Highlighted Sample Code**.

MATLAB inserts the following cell break (%%) and commented code:

```
%%  
%  
%   for i = 1:10  
%       disp x  
%   end  
%
```

- 3** Being careful not to delete the three spaces between the comment character and the first character of code in the comment block, replace the for loop code that MATLAB inserted with your sample code and add any additional comments.

For example, suppose you publish the following file to HTML:

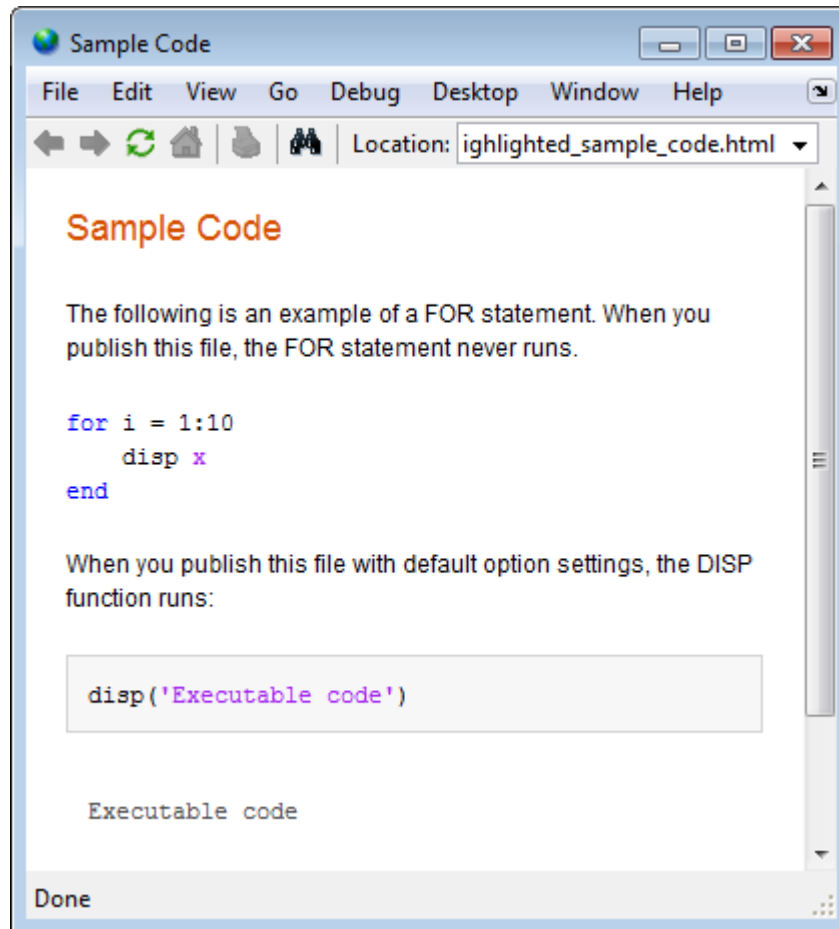


The screenshot shows the MATLAB Editor window titled "Editor - C:\my_MATLAB_files\syntax_highlighted_sample_code.m". The window contains the following code:

```
1   %% Sample Code  
2   % The following is an example of a FOR statement.  
3   % When you publish this file, the FOR statement  
4   % never runs.  
5   %  
6   %   for i = 1:10  
7   %       disp x  
8   %   end  
9   %  
10  %  
11  % When you publish this file with default option settings,  
12  % the DISP function runs:  
13  %  
14  - disp('Executable code')  
15
```

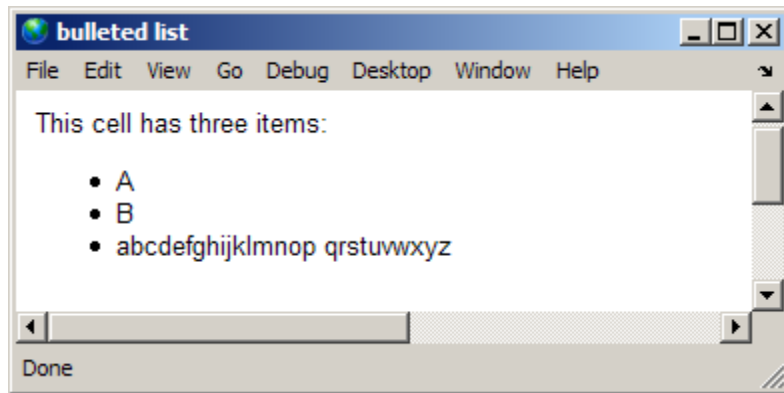
The status bar at the bottom of the window shows "script", "Ln 12", "Col 6", and "OVR".

The output appears as shown in this figure.



Bulleted or Numbered Lists

The following steps describe how to specify text markup for a bulleted or numbered list, so that it appears as shown when you publish it.

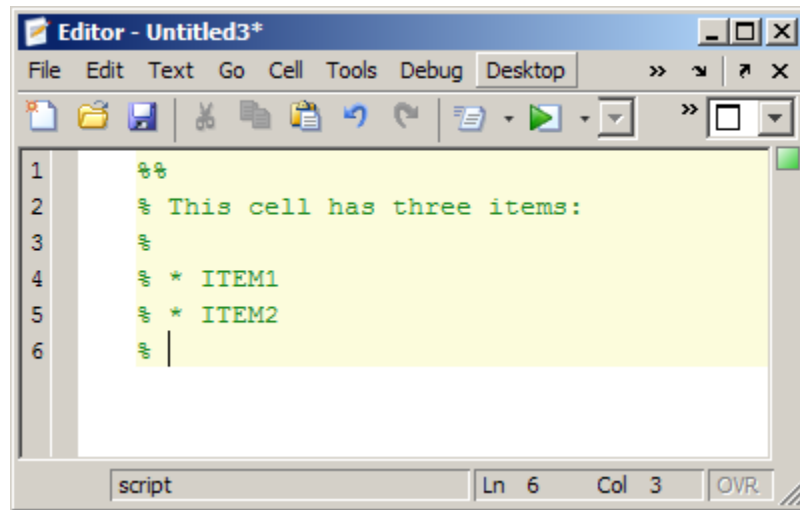


- 1 Position the cursor at the end of the line that precedes the location where you want to add a list. For example, if your file contains the following lines, position the cursor after the colon:

```
%%  
% 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 comments to the file. The following figure shows the result when you insert a bulleted list.

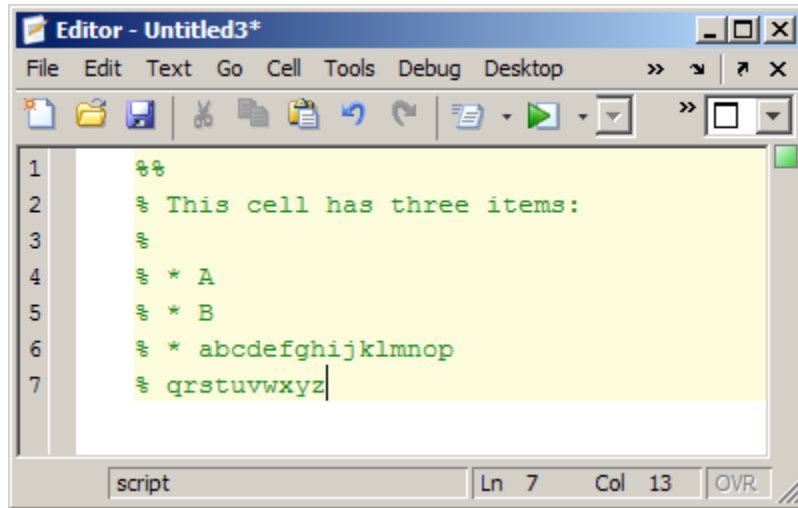
The image shows a screenshot of the MATLAB Editor window titled "Editor - Untitled3*". The window has a menu bar with "File", "Edit", "Text", "Go", "Cell", "Tools", "Debug", and "Desktop". Below the menu bar is a toolbar with various icons for file operations, editing, and execution. The main editing area contains a code cell with a yellow background. The code in the cell is as follows:

```
1 %%  
2 % This cell has three items:  
3 %  
4 % * ITEM1  
5 % * ITEM2  
6 %
```

The status bar at the bottom of the window shows "script", "Ln 6", "Col 3", and "OVR".

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 breaks over two comment lines in the source, yet maintains the appearance of a list when published (as shown at the beginning of this section).

External Graphics

You can insert text markup to publish an image that the MATLAB code did not generate. This is shown in the following example. (By default, the publishing includes images generated by the MATLAB code.)

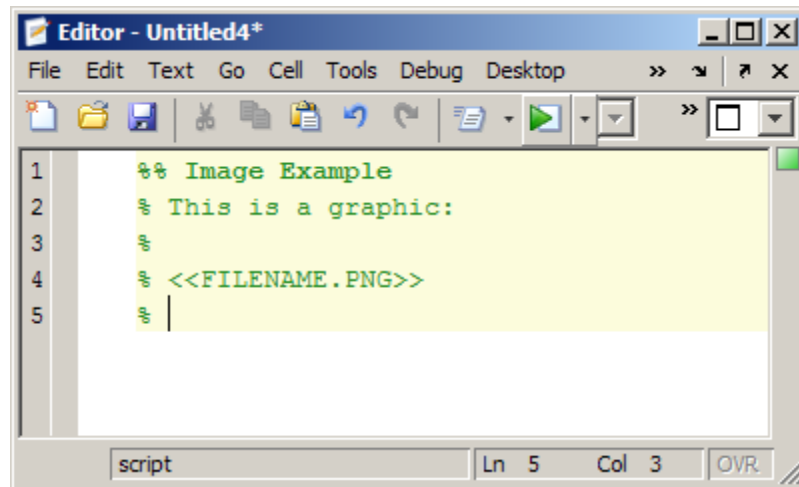
- 1 Position the cursor where you want to add a graphic. For example, if your file contains the following lines, position the cursor after the colon:

```

%% 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 file.

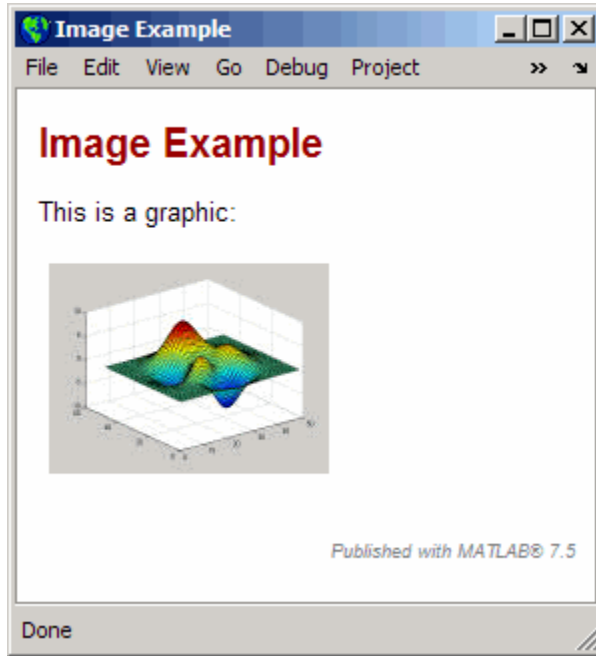
For example, if you want to include the graphic, `surfpeaks.png`, and it is in the folder into which MATLAB publishes the MATLAB code file, then replace FILENAME.PNG with `surfpeaks.png`. See “Creating the surfpeaks.png Image” on page 10-35.

By default, MATLAB publishes the file to an `/html` subfolder of the folder containing the file. You can change this folder, referred to as the output folder, by changing the publish configuration settings, as described in “Specify Output Preferences for Publishing” on page 10-66.

If the graphic is not in the folder to which you publish the file, then you must specify the location of the graphic file as a relative path from the location of the file that results from publishing. The following table summarizes how to specify such graphics, assuming the folder containing the MATLAB code is `I:\my_MATLAB_files`.

Folder Where publish Saves the Output	Image File Location	How to Specify the Image in the MATLAB Comment
<code>I:\my_MATLAB_files/html</code>	<code>I:\my_MATLAB_files/html</code>	<code>% <<surfpeaks.jpg>></code>
<code>I:\my_MATLAB_files/doc</code>	<code>I:\my_MATLAB_files/images</code>	<code>% <<../images/surfpeaks.jpg>></code>

When you publish the file to HTML, the output appears as shown in the following figure.



Valid Image Types for Output File Formats

The type of images you can include when you publish depends on the output type of that document as indicated in the table that follows. For greatest compatibility, MathWorks recommends using the default image format for each output type.

Output File Format	Types of Images You Can Include
doc	Any format that your installed version of Microsoft Office supports.
html	Any format publishes successfully. Ensure that the tools you use to view and process the output files can display the output format you specify.

Output File Format	Types of Images You Can Include
latex	Any format publishes successfully. Ensure that the tools you use to view and process the output files can display the output format you specify.
pdf	bmp and jpg.
ppt	Any format that your installed version of Microsoft Office supports.
xml	Any format publishes successfully. Ensure that the tools you use to view and process the output files can display the output format you specify.

Creating the surfpeaks.png Image

To create the `surfpeaks.png` image used in the preceding example, follow these steps:

- 1 Create an `html` subfolder in the folder where the 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 shown in the previous example.

HTML Markup

You can use the **Cell** menu to insert HTML code into your MATLAB 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.

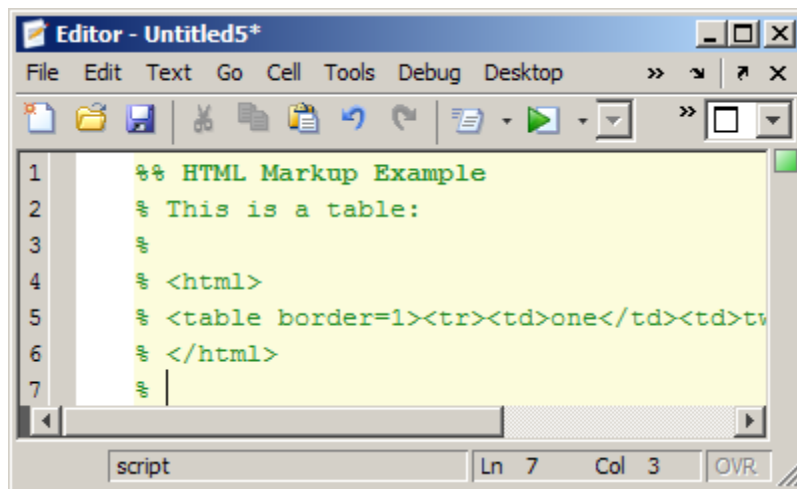
Note When you insert text markup for HTML code, the HTML code publishes only when the specified output file format is HTML. For example, if you add HTML markup, but then specify LaTeX as the output file format, publishing excludes the text enclosed within the HTML markup. See “Specify Output Preferences for Publishing” on page 10-66 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 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 file to HTML and leave the inserted HTML code as is, MATLAB creates a single-row table with two columns. The table contains the values one and two, as shown in the following figure.



LaTeX Markup

Use the **Cell** menu to insert LaTeX code into your file for publishing. Then, use the inserted code as a guideline for inserting other LaTeX code.

Note When you insert text markup for LaTeX code, that code publishes only when the specified output file format is LaTeX. For example, if you add LaTeX markup, but specify HTML as the output file format, publishing excludes the code enclosed within the LaTeX markup. See “Specify Output Preferences for Publishing” on page 10-66 for information on specifying the output file format.

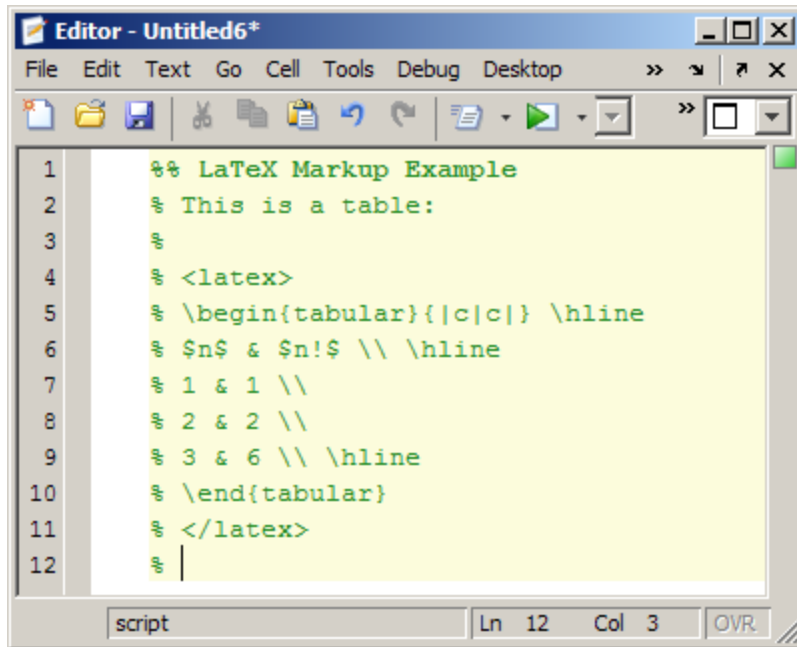
To insert the text markup for LaTeX code, follow these steps:

- 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 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.



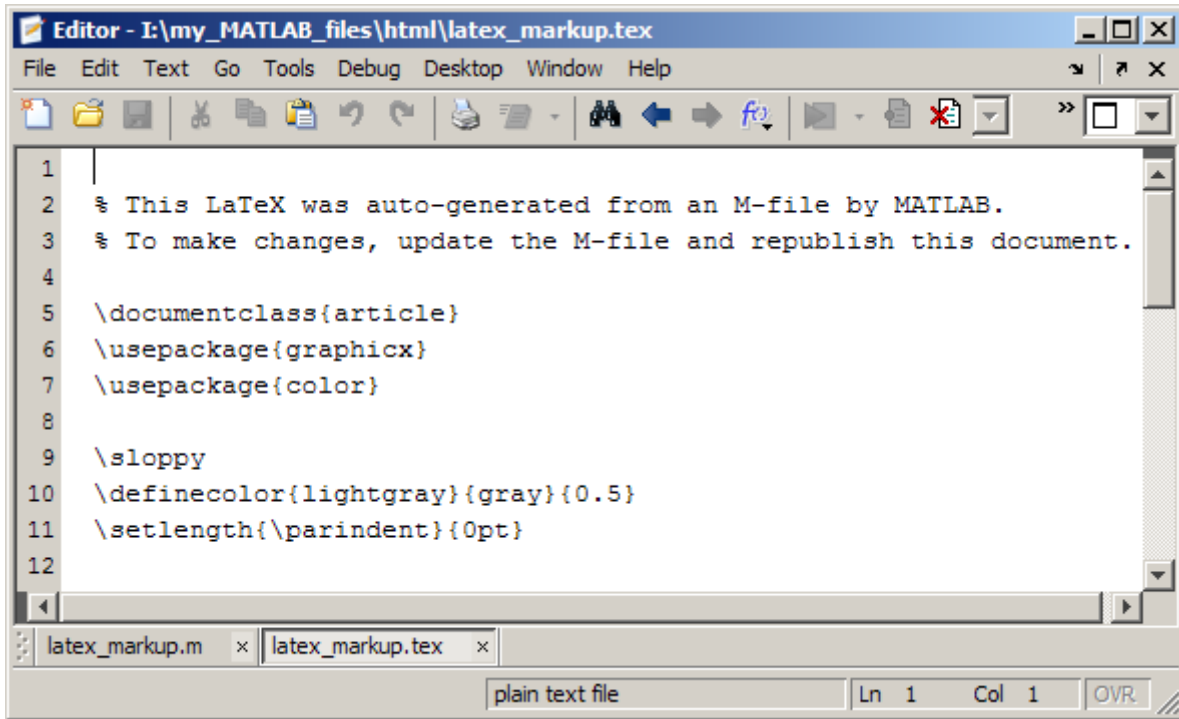
The screenshot shows the MATLAB Editor window titled "Editor - Untitled6*". The menu bar includes File, Edit, Text, Go, Cell, Tools, Debug, and Desktop. The toolbar contains various icons for file operations and editing. The main text area shows the following code:

```
1 %% LaTeX Markup Example
2 % This is a table:
3 %
4 % <latex>
5 % \begin{tabular}{|c|c|} \hline
6 % $n$ & $n!$ \\ \hline
7 % 1 & 1 \\
8 % 2 & 2 \\
9 % 3 & 6 \\ \hline
10 % \end{tabular}
11 % </latex>
12 %
```

The status bar at the bottom indicates the file name "script", the current line "Ln 12", the current column "Col 3", and the view mode "OVR".

- 3 Edit the inserted LaTeX code to specify the LaTeX code that you want to use.

Suppose you publish the file to LaTeX, and leave the inserted markup text as is. In that case, the Editor opens a new file with the LaTeX code, as shown in the following figure. (See “Creating a Publish Configuration for a MATLAB File” on page 10-68 for information on specifying LaTeX as the output format.)



The screenshot shows a MATLAB Editor window titled "Editor - I:\my_MATLAB_files\html\latex_markup.tex". The window contains the following LaTeX code:

```
1 |
2 | % This LaTeX was auto-generated from an M-file by MATLAB.
3 | % To make changes, update the M-file and republish this document.
4 |
5 | \documentclass{article}
6 | \usepackage{graphicx}
7 | \usepackage{color}
8 |
9 | \sloppy
10 | \definecolor{lightgray}{gray}{0.5}
11 | \setlength{\parindent}{0pt}
12 |
```

The status bar at the bottom indicates "plain text file", "Ln 1 Col 1", and "OVR".

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

Inline LaTeX Math Equations

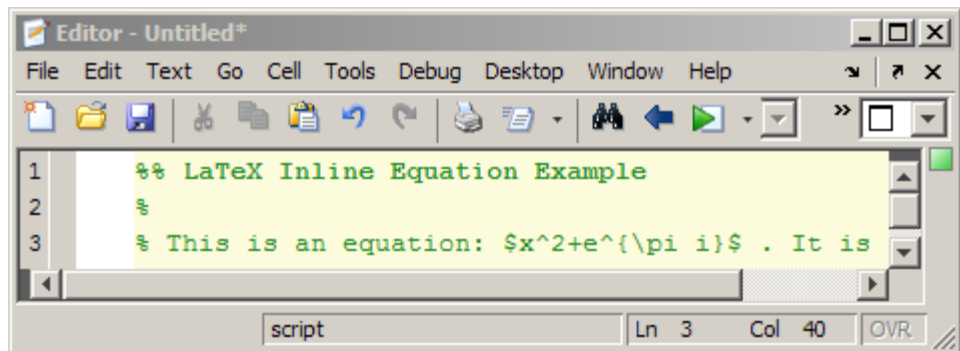
Use the **Cell** menu to include an inline LaTeX math equation in MATLAB code that you intend to publish. Then, use the inserted equation as a guideline for inserting other LaTeX math equations.

Note Publishing supports all output types for LaTeX math equations, except Microsoft PowerPoint. For PowerPoint output, see “LaTeX Markup” on page 10-37.

- 1 Position the cursor within the line where you want to add an equation. For example, if your file contains the following lines, position the cursor between the colon and the period in the third line:

```
%% LaTeX Inline Equation Example
%
% This is an equation:. It is in line
% with the text.
```

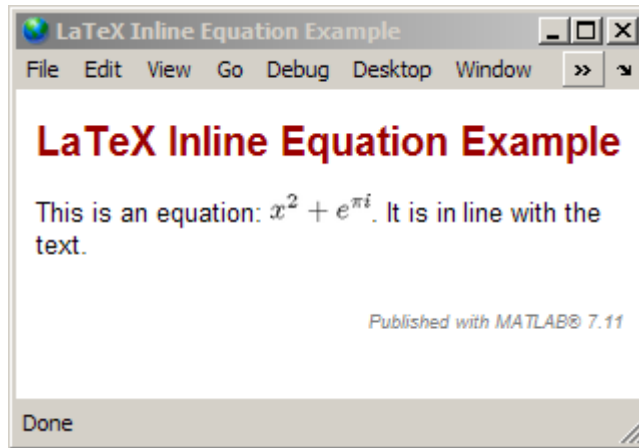
- 2 Select **Cell > Insert Text Markup > LaTeX Inline Math** to insert the markup for the sample equation shown here.



- 3 Replace $x^2+e^{\pi i}$ with the LaTeX math symbols that you want, being careful not to delete the dollar sign characters (\$).

For a list of symbols you can display, and the character sequence to create them, see the MATLAB String property.

If you publish the file to HTML, it appears as shown here. (For this example, the only change to the inserted sample code is the deletion of the space between the inserted markup and the period.)



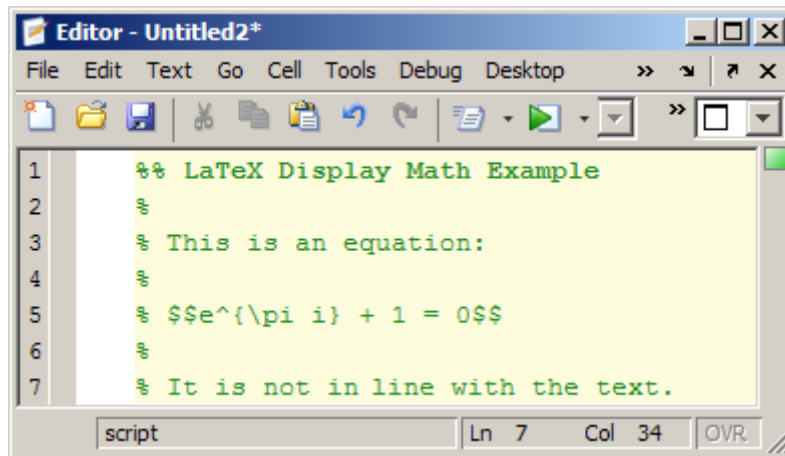
LaTeX Display Math

Use the **Cell** menu to insert LaTeX symbols in blocks offset from the main comment text. Then, use the inserted code as a guideline for inserting other LaTeX code. A block of LaTeX symbols is sometimes referred to as *display math*.

- 1 Position the cursor before the line where you want to add the display math. For example, if your file contains the following lines, position the cursor after the colon on the end of the third line:

```
%% LaTeX Display Math Example
%
% This is an equation:
% It is not in line with the text.
```

- 2 Select **Cell > Insert Text Markup > LaTeX Display Math** to insert the sample equation markup.



The screenshot shows the MATLAB Editor window titled "Editor - Untitled2*". The menu bar includes File, Edit, Text, Go, Cell, Tools, Debug, and Desktop. The toolbar contains icons for file operations and execution. The script content is as follows:

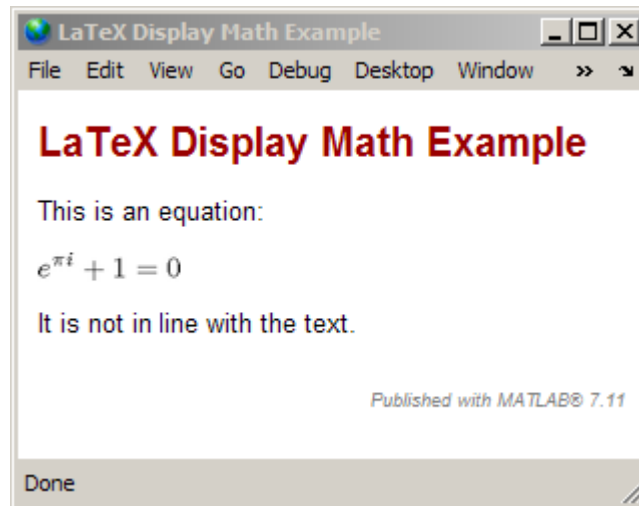
```
1 %% LaTeX Display Math Example
2 %
3 % This is an equation:
4 %
5 % $$e^{\pi i} + 1 = 0$$
6 %
7 % It is not in line with the text.
```

The status bar at the bottom indicates the file name "script", the current cursor position "Ln 7 Col 34", and the view mode "OVR".

- 3 Replace the inserted sample markup $e^{\pi i} + 1 = 0$ with the LaTeX math symbols that you want to use, being careful not to delete the dollar sign characters (\$).

For a list of symbols that you can display and the character sequence to create them, see the MATLAB String property.

If you leave the inserted sample text markup as is, and publish to HTML, it appears as shown here.



Force a Snapshot of Output

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 publish them all, 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 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
```

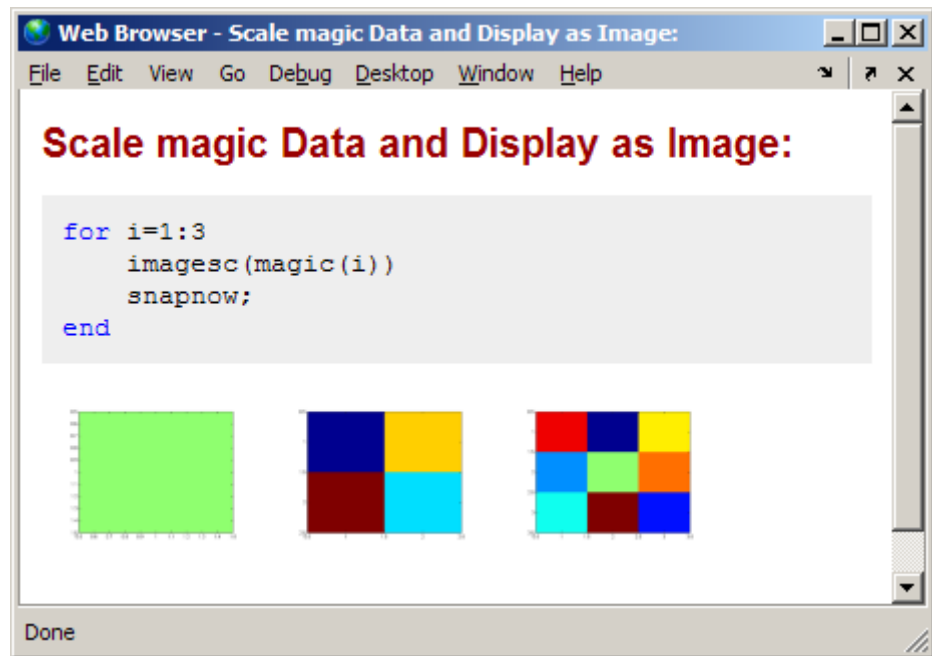
- 2 Select **Cell > Insert Text Markup > Force Snapshot** to insert the `snapshot` function:

```
%% Scale magic Data and Display as Image:

for i=1:3
    imagesc(magic(i))
    snapshot
```

```
    snapnow;  
end
```

- 3 If you publish the file to HTML, it resembles the following figure. The images in your HTML will be larger than shown in the figure. To resize images generated by MATLAB code, use the **Max image width** and **Max image height Publish settings**, as described in “Specify Output Preferences for Publishing” on page 10-66.



Bold, Italic, and Monospaced Text

You can mark selected strings in the MATLAB comments so that they display in bold, italic, or monospaced text when you publish the file. The following sections provide instructions.

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:

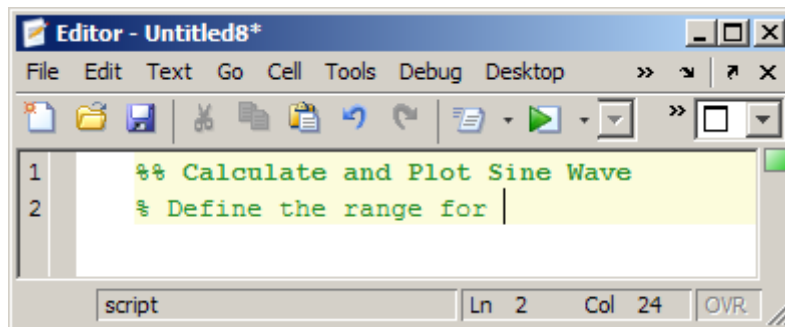
- 1 Select **Cell > Insert Text Markup**, and then select **Bold Text**, **Italic Text**, or **Monospaced Text**.
- 2 Replace the inserted text with the text that you want.

When the Editor inserts sample text, the inserted text appears as follows:

```
% *BOLD TEXT*  
% _ITALIC TEXT_  
% |MONOSPACED TEXT|
```

Example of Font Formats

Suppose your file appears as follows.

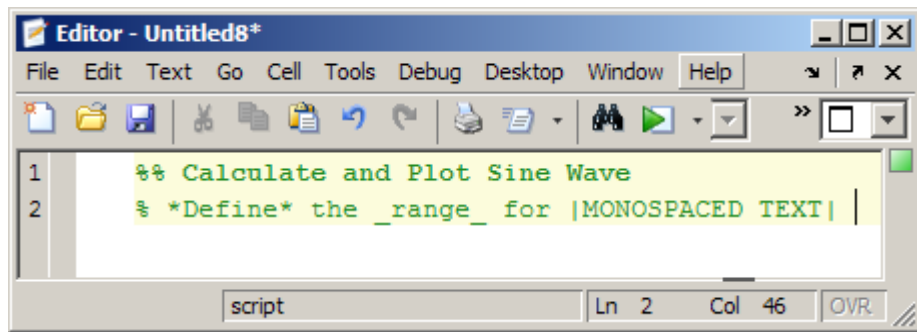


Mark up the comments as follows:

- 1 Select the word **Define**, and then select **Cell > Insert Text Markup > Bold Text**.

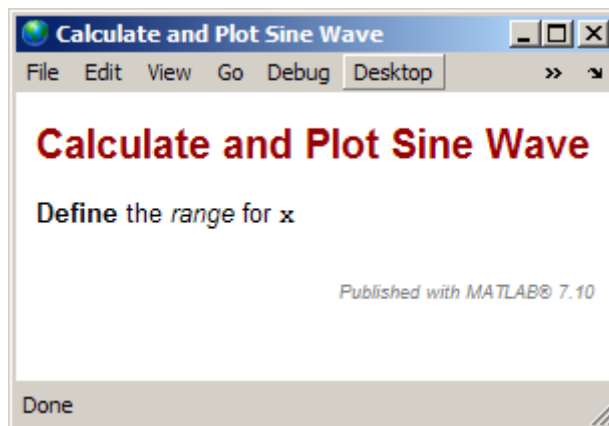
- 2 Select the word range, and then select **Cell > Insert Text Markup > Italic Text**.
- 3 Position the cursor after the word for, insert a space, and then select **Cell > Insert Text Markup > Monospaced Text**.

The file appears as follows.



- 4 Replace |MONOSPACED TEXT| with |x|.

If you publish the file to HTML, the output appears as shown in the following figure.

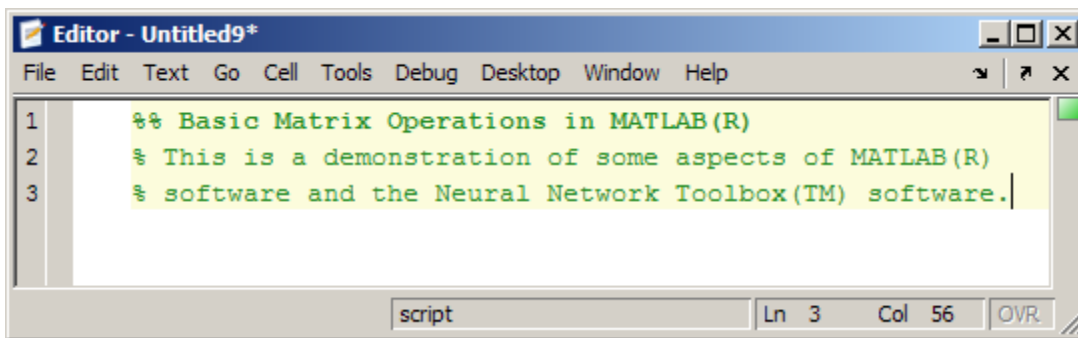


Trademark Symbols

If the comments in your MATLAB file include trademarked terms, you can include text to produce a trademark symbol (™) or registered trademark symbol (®) in the output.

- To produce the trademark symbol, enter (TM) in a MATLAB comment.
- To produce the registered trademark symbol, enter (R) in a MATLAB comment.

For example, suppose you enter lines in a file as shown in the following image.



If you publish the file to HTML, it appears as follows in the MATLAB Web Browser.



Hyperlinks

You can insert dynamic or static hyperlinks within a MATLAB comment, and then publish the MATLAB file to HTML or XML. You can also publish static hyperlinked text to Microsoft Word.

When you specify a *dynamic* hyperlink, MATLAB evaluates the hyperlinked code when someone clicks it in the output. This practice is useful when, for example, you want to point the reader to MATLAB code or documentation or you want the link to run code. When you specify a *static* hyperlink to a Web location, you specify a complete URL within the code. This is useful when you want to point the reader to a location on the Web. For both static and dynamic hyperlinks, the output contains active hyperlinks.

You can include or exclude the URL from a static hyperlink. Consider including the URL, for example, when you anticipate that readers of your output file might view it in printed form, and therefore need the URL. Consider excluding the URL, when you are confident that readers will view your output online and therefore be able to click the hyperlink.

This section includes the following topics:

- “Static Hyperlinks and Publishing URLs” on page 10-48
- “Static Hyperlinks Without Publishing URLs” on page 10-49
- “Dynamic Hyperlinks” on page 10-50
- “Effect of Copying Hyperlinked Text from the MATLAB Command Window” on page 10-54

Static Hyperlinks and Publishing URLs

You can insert a URL, such as `www.mathworks.com`, that you know at the time you write the file 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 and you have the following comment within the file:

```
%%  
% For more information, see our Web site:
```

Position your cursor after the colon (:).

2 Select **Cell > Insert Text Markup > Hyperlinked Text**.

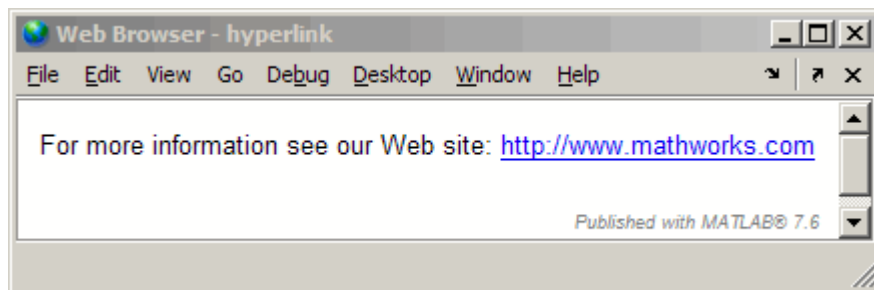
The Editor inserts the following:

```
<http://www.mathworks.com MathWorks>
```

3 Replace `www.mathworks.com` with the URL you want to use.

4 Delete the string, `MathWorks`.

When you publish the file to HTML, the results resemble the following figure (except the URL in this image is still `http://www.mathworks.com`).



Static Hyperlinks 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 hyperlink to the MathWorks Web site and you have the following lines within your file:

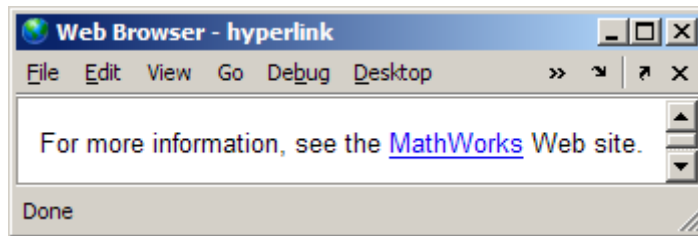
```
%%
% For more information, see the MathWorks Web site.
```

Select the text you want to replace with a hyperlink. For example, select "MathWorks"

- 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 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.

Dynamic Hyperlinks

You can insert a hyperlink which MATLAB evaluates at the time a reader clicks that link. You implement these links using `matlab:` syntax.

You cannot insert dynamic links in Microsoft Word files.

Note A reader must have MATLAB running for dynamic links to work.

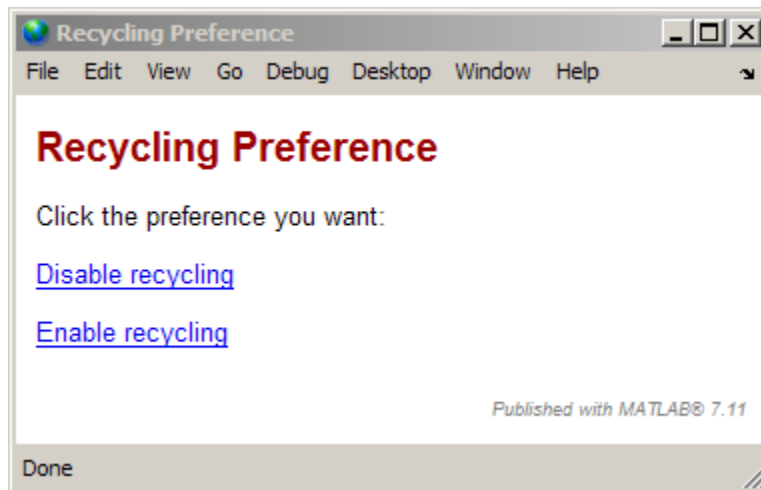
The following sections provide examples:

- “Dynamic Link To Run Code” on page 10-51
- “Dynamic Link to a File” on page 10-51
- “Dynamic Link to a MATLAB Function Reference Page” on page 10-53

Dynamic Link To Run Code. You can specify a dynamic hyperlink to run code when a user clicks the hyperlink. For example, the following `matlab:` syntax creates hyperlinks in the output, which when clicked either enable or disable recycling:

```
%% Recycling Preference
% Click the preference you want:
%
% <matlab:recycle('off') Disable recycling>
% <matlab:recycle('on') Enable recycling>
```

When you publish the file to HTML, the results resemble the following figure:



When you click one of the hyperlinks, MATLAB sets the `recycle` command accordingly. After clicking a hyperlink, run `recycle` in the Command Window to confirm the setting is as you expect.

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 file:

```
%%  
% See the code  
% for the publish function.
```

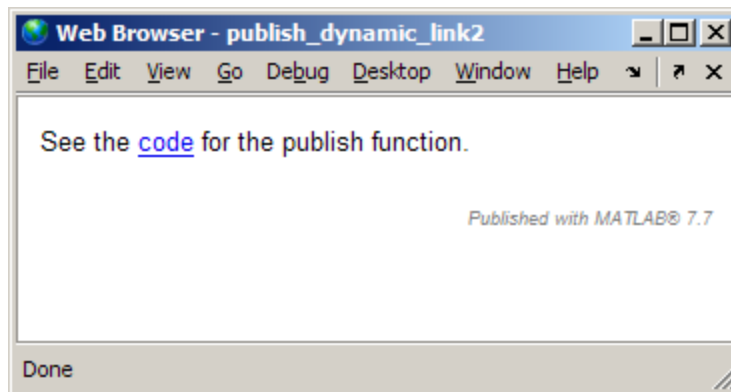
2 Replace the word `code` with the following markup:

```
<matlab:web(fullfile(matlabroot,'toolbox','matlab','codetools','publish.m')) code>
```

The resulting 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 you click 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 file resolves to the reader's installation folder for MATLAB.

The screenshot shows a web browser window with the address bar containing the file path: `file:///C:/Program Files/MATLAB/R2009a/toolbox/matlab/codetools/p...`. The browser's menu bar includes File, Edit, View, Go, Debug, Desktop, Window, and Help. The main content area displays the MATLAB documentation for the `publish` function, including its signature `function outputAbsoluteFilename = publish(file,options)` and detailed comments explaining its usage and supported output formats like 'html', 'doc', and 'ppt'.

```
function outputAbsoluteFilename = publish(file,options)
%PUBLISH Create a document from an M-file.
% PUBLISH(FILE) evaluates the M-file one cell at a time in the
% base workspace. It saves the code, comments, and results to an
% with the same name. The HTML-file is stored, along with other
% output files, in an "html" subdirectory within the script's dire
%
% PUBLISH(FILE,FORMAT) saves the results to the specified format.
% can be one of the following:
%
% 'html' - HTML.
% 'doc' - Microsoft Word (requires Microsoft Word).
% 'ppt' - Microsoft PowerPoint (requires Microsoft PowerPoint)
% 'html' - An HTML file that can be transformed with XSLT...
```

Dynamic Link to a MATLAB Function Reference Page. You can specify a link to a MATLAB function reference page using `matlab:` syntax. For example, suppose your readers 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 file:

```
%%
% See the help for the publish function.
```

- 2 Replace the word `publish` with the following markup:

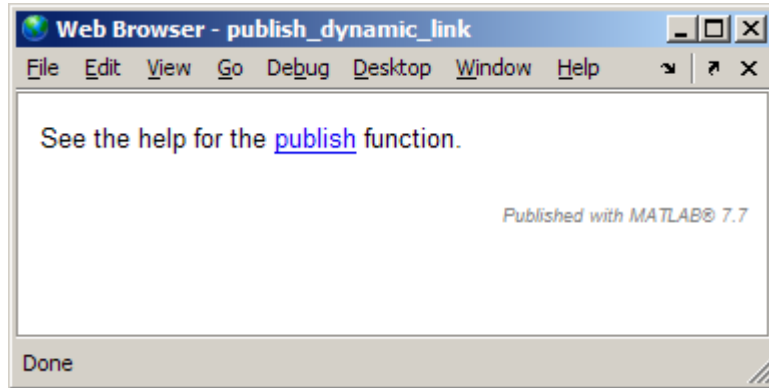
```
<matlab:doc('publish') publish>
```

The resulting 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 you click the `publish` hyperlink, 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 Copying Hyperlinked Text from the MATLAB Command Window

If you copy statements that display hyperlinked text from the MATLAB Command Window to a file and then publish that file, results might not be as you expect. When you do so, the output 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](http://www.mathworks.com)

However, if you include the preceding `disp` statement in a file that you publish, the HTML tag and the included text appears in the output, 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:

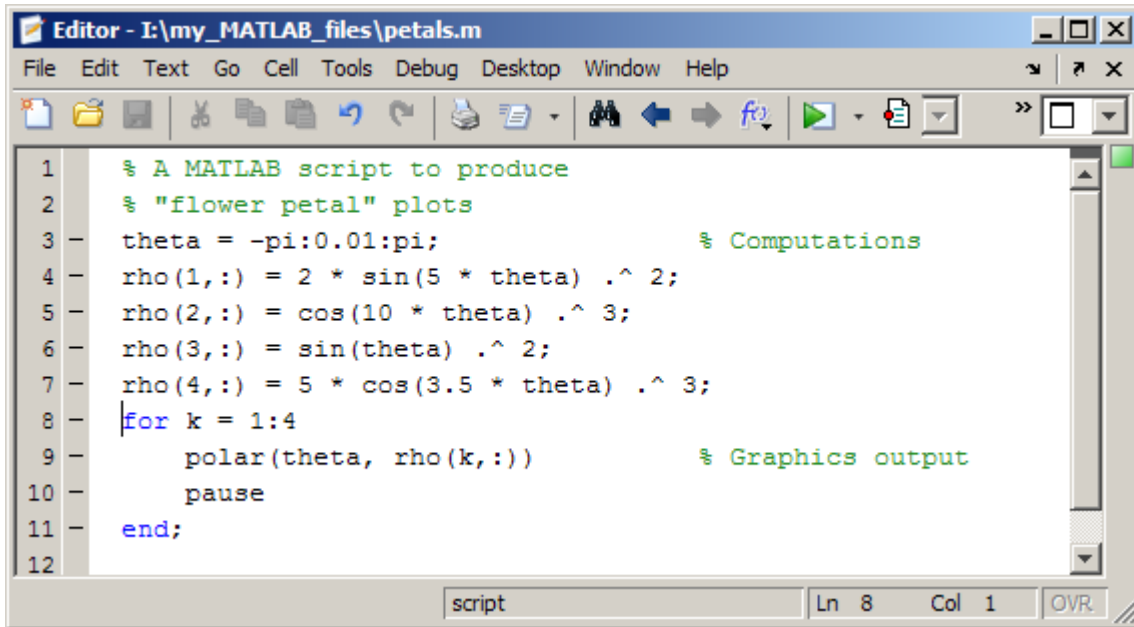
- “Static Hyperlinks and Publishing URLs” on page 10-48
- “Static Hyperlinks Without Publishing URLs” on page 10-49
- “Dynamic Hyperlinks” on page 10-50

Cleaning Up Text Markup Before Publishing MATLAB Files

When you insert text markup into an existing file using the **Cell** menu options, sometimes 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 might be unintended consequences, however. See “Preformatted Text” on page 10-25 for details.

The following example shows how you can use **Cell** menu options with an existing file.

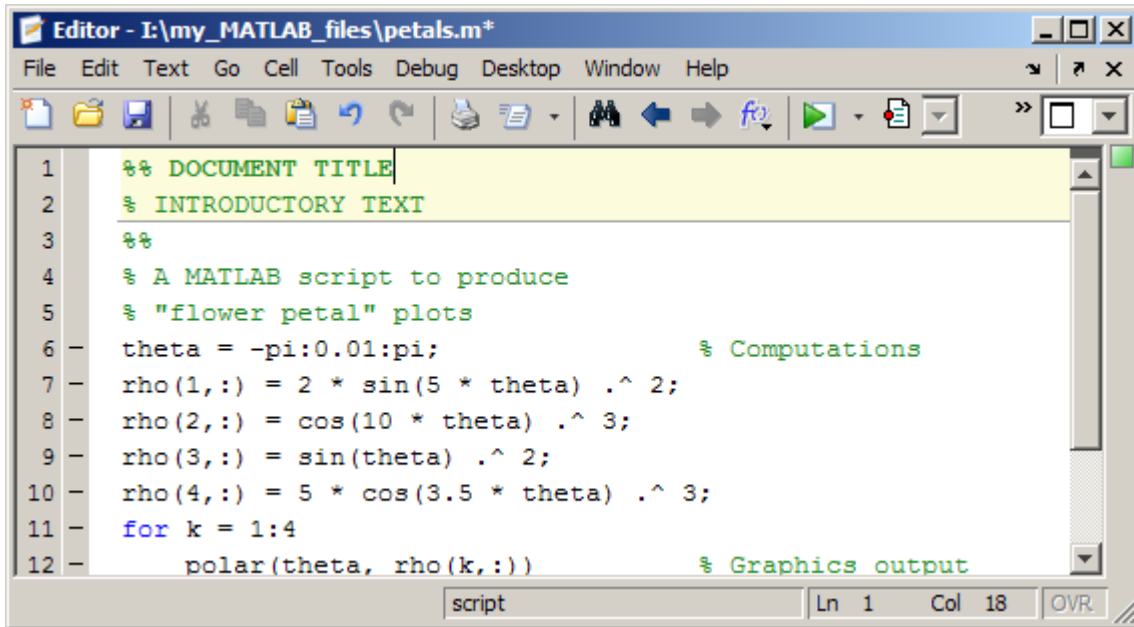
Suppose a file currently appears in the Editor as shown in the following image.



```
Editor - I:\my_MATLAB_files\petals.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1 % A MATLAB script to produce
2 % "flower petal" plots
3 - theta = -pi:0.01:pi; % Computations
4 - rho(1,:) = 2 * sin(5 * theta) .^ 2;
5 - rho(2,:) = cos(10 * theta) .^ 3;
6 - rho(3,:) = sin(theta) .^ 2;
7 - rho(4,:) = 5 * cos(3.5 * theta) .^ 3;
8 - for k = 1:4
9 -     polar(theta, rho(k,:)) % Graphics output
10 -     pause
11 - end;
12
```

script Ln 8 Col 1 OVR

If you position the cursor anywhere within the file and select **Cell > Insert Text Markup > Document Title and Introduction**, the file looks like the following.

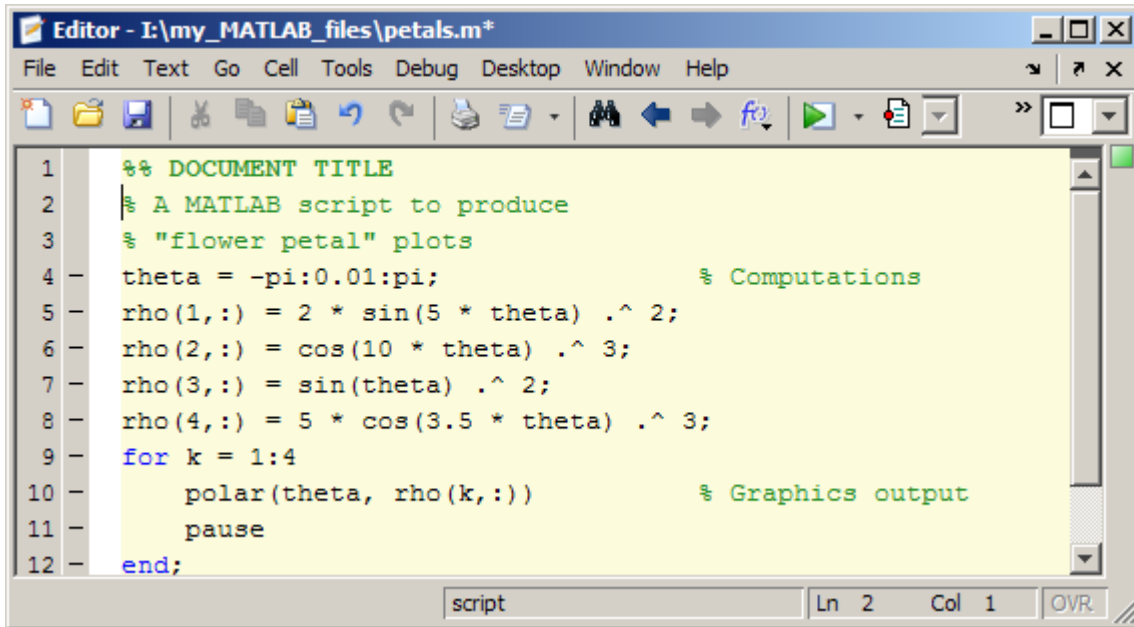


The screenshot shows the MATLAB Editor window titled "Editor - I:\my_MATLAB_files\petals.m*". The window contains the following code:

```
1 %% DOCUMENT TITLE
2 % INTRODUCTORY TEXT
3 %%
4 % A MATLAB script to produce
5 % "flower petal" plots
6 - theta = -pi:0.01:pi; % Computations
7 - rho(1,:) = 2 * sin(5 * theta) .^ 2;
8 - rho(2,:) = cos(10 * theta) .^ 3;
9 - rho(3,:) = sin(theta) .^ 2;
10 - rho(4,:) = 5 * cos(3.5 * theta) .^ 3;
11 - for k = 1:4
12 -     polar(theta, rho(k,:)) % Graphics output
```

The status bar at the bottom indicates "script", "Ln 1", "Col 18", and "OVR".

The file already contains a comment with introductory text, so you can delete the % INTRODUCTORY TEXT line and the double percent sign (%%) line. When you do so, the code appears as follows.



```
Editor - I:\my_MATLAB_files\petals.m*
File Edit Text Go Cell Tools Debug Desktop Window Help
1 %% DOCUMENT TITLE
2 | A MATLAB script to produce
3 | "flower petal" plots
4 - theta = -pi:0.01:pi; % Computations
5 - rho(1,:) = 2 * sin(5 * theta) .^ 2;
6 - rho(2,:) = cos(10 * theta) .^ 3;
7 - rho(3,:) = sin(theta) .^ 2;
8 - rho(4,:) = 5 * cos(3.5 * theta) .^ 3;
9 - for k = 1:4
10 -     polar(theta, rho(k,:)) % Graphics output
11 -     pause
12 - end;
```

Summary of Markup for Publishing MATLAB Files

The following table provides a summary of the text markup that you can type into a file to achieve the same results as using the **Cell > Insert Text Markup** menu options. These tables are useful if you are not using the MATLAB Editor, or if you do not want to use menus to apply markup. For a description of the **Cell** menu options, see “Mark Up MATLAB Comments for Publishing” on page 10-17.

Note The blank comment lines preceding and following some items, such as bulleted lists, are required.

Result in Output	Example of Corresponding File Markup
Document title and introduction	<pre>%% DOCUMENT TITLE % INTRODUCTORY TEXT</pre>
Section title and description	<pre>%% SECTION TITLE % DESCRIPTIVE TEXT</pre>
Section title without cell break	<pre>%% SECTION TITLE % DESCRIPTIVE TEXT</pre>
Bold text	<pre>% *BOLD TEXT*</pre>
Italic text	<pre>% _ITALIC TEXT_</pre>
Monospaced text	<pre>% MONOSPACED TEXT </pre>
Hyperlinked text	<pre>% <http://www.mathworks.com MathWorks></pre>
Trademark symbol	<pre>% TEXT(TM)</pre>
Registered trademark symbol	<pre>% TEXT(R)</pre>
Code to force a snapshot of the current output	<pre>snapnow;</pre>
Image	<pre>% % <<FILENAME.PNG>> %</pre>
Bulleted list	<pre>% % * ITEM % * ITEM %</pre>

Result in Output	Example of Corresponding File Markup
Numbered list	<pre data-bbox="563 326 704 447">% % # ITEM1 % # ITEM2 %</pre>
HTML markup	<pre data-bbox="563 494 972 708">% % <html> % <table border=1><tr> % <td>one</td> % <td>two</td></tr></table> % </html> %</pre>
LaTeX display math	<pre data-bbox="563 755 912 789">% \$\$e^{\pi i} + 1 = 0\$\$</pre>
LaTeX inline math	<pre data-bbox="563 847 823 881">% \$x^2+e^{\pi i}\$</pre>
LaTeX markup	<pre data-bbox="563 951 972 1229">% % <latex> % \begin{tabular}{ r r} % \hline \$n\$&\$n!\$\\ % \\hline 1&1\\ 2&2\\ 3&6\\ % \\hline % \end{tabular} % </latex> %</pre>

Result in Output	Example of Corresponding File Markup
Preformatted text block	<pre>% % This is a block of preformatted text. % The first line of commented text must % begin with two blank spaces. %</pre>
Syntax Highlighted Sample Code	<pre>%% % Three spaces must appear % between comment character % and first character of % code. For instance: % % for i = 1:10 % disp x % end %</pre>

Mark Up MATLAB Code for Publishing

In this section...

“Overview of Marking Up MATLAB Code for Publishing” on page 10-62

“Specifying the Display of Code Output” on page 10-62

“Example of Marking Up Code” on page 10-62

Overview of Marking Up MATLAB Code for Publishing

This section describes ways to control how output that MATLAB generates when it publishes executable MATLAB code. For example, you can direct MATLAB to include the last, or all plots generated by a `for` loop. You can interweave comments, code, and code output throughout your output document to draw your readers’ attention to certain areas of interest.

Specifying the Display of Code Output

The method you use to specify how output displays in the document is the same method you use to specify document titles and section headers; namely the 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 the entire file is a cell, inserting a cell break results in the file containing two cells. The first cells is the one above the cell break, and the second is the one below the cell break. The examples in the remaining topics demonstrate how you can use this behavior to control the output produced by MATLAB code.

Example of Marking Up Code

This section provides an example to demonstrate how a MATLAB file appears when published. It demonstrates how the published example file appears before and after cell breaks are added.

Sample MATLAB File Before Inserting Mark Up in Code

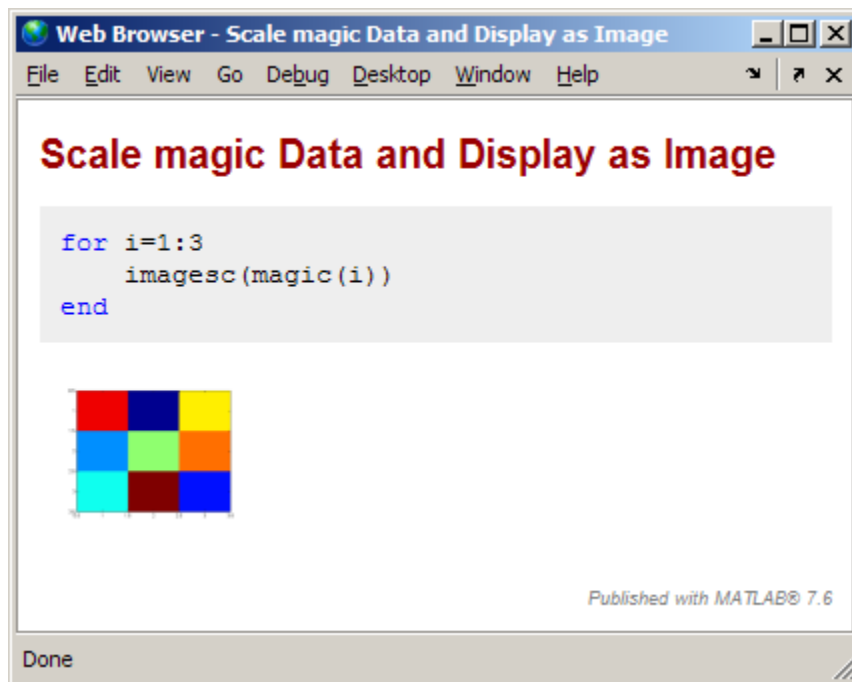
Suppose your file contains the following code:

```
%% 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 code using factory default settings. For information on setting publishing properties for images, see “Specify Output Preferences for Publishing” on page 10-66.

Notice that the output displays the plot after the end of the `for` loop and that only the last plot generated by the code displays.



Sample MATLAB File After Inserting Cell Breaks in Code

By placing cell breaks within a loop, you can display the output that the MATLAB code generates when iterating a loop.

To include the plot generated by each iteration of the loop in the output file, 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 code using factory default settings. For information on setting publishing properties for images, see “Specify Output Preferences for Publishing” on page 10-66.

Notice that the output file displays the plot within the `for` loop code. You can also use text markup for similar results with figures. See “Mark Up MATLAB Code for Publishing” on page 10-62 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

Specify Output Preferences for Publishing

In this section...

“About Publishing Configurations” on page 10-66

“Creating a Publish Configuration for a MATLAB File” on page 10-68

“Running an Existing Publish Configuration” on page 10-95

“Creating Multiple Publish Configurations for a File” on page 10-96

“About the publish_configurations.m File” on page 10-107

“Finding Publish Configurations” on page 10-108

“Removing Publish Configurations” on page 10-108

“Reassociating and Renaming Publish Configurations” on page 10-108

About Publishing Configurations

Once you have marked up MATLAB code for publishing, as described in “Mark Up MATLAB Comments for Publishing” on page 10-17 and “Mark Up MATLAB Code for Publishing” on page 10-62 you are ready to publish it. The easiest method for publishing a MATLAB code file is to use the factory default publishing configuration. This method is appropriate if your code requires no input arguments and you want to publish to HTML. However, if your code requires input arguments, or if you want to specify preferences for publishing (such as the output folder, output format, image format, and so on), specify a custom configuration.

Publishing MATLAB Files Using No Input Arguments and Factory Default Settings

To publish a MATLAB script or MATLAB function 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 code is evaluated and included in the output file, and so on.

If the file is not in a folder on the search path or in the current folder, a dialog box opens and presents you with options that allow you to publish the file. Either change the current folder to the folder containing the file, or 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 MATLAB Files Using Input Arguments or Custom Settings

Using a publish configuration, you can specify custom settings, including input arguments for a MATLAB function file in the Editor. You can associate multiple publish configurations with a 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 `collatzplot_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 before publishing a file, whether it takes input arguments or not.

Note Publish configurations use the base MATLAB workspace. Therefore, a value that you assign to a variable in a publish configuration overwrites the value for that variable (assuming that it currently exists) in the base workspace.

Function Alternative to Publishing

From the Command Window, execute the `publish` function to run the file and publish the results. See the `publish` function reference page for options you can set.

Creating a Publish Configuration for a MATLAB File

- “Specifying File Input Using a Publish Configuration” on page 10-68
- “Specifying Publish Configuration Settings” on page 10-72
- “Specifying Values for the Publish Settings Property Table” on page 10-76
- “Creating a Template for Typical Publish Settings” on page 10-93

Specifying File Input Using a Publish Configuration

Follow these steps to create a publish configuration for a MATLAB code file in the Editor. The example in this section shows how to create and use a publish configuration to specify input arguments to a MATLAB function 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 like the `sine_wave.m` file, after it has been marked up as described in “Mark Up MATLAB Comments for Publishing” on page 10-17, but it is slightly altered to make it a MATLAB function 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)|.

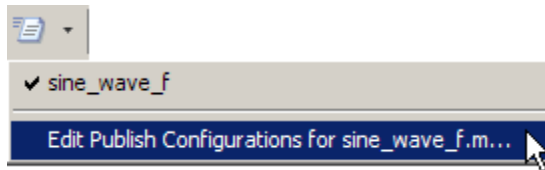
function sine_wave_f(x)

y = sin(x);
plot(x,y)
```

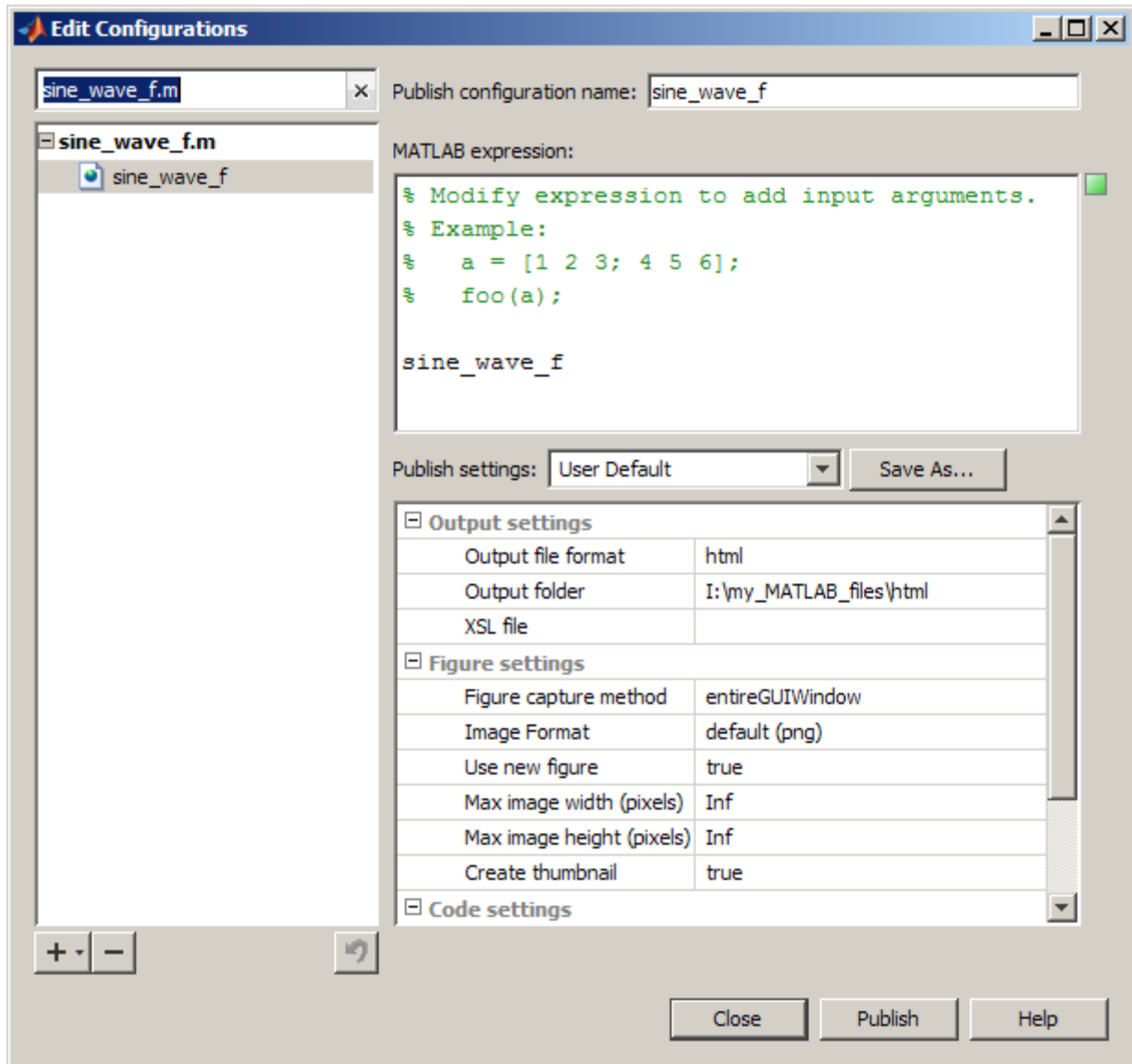
```
%% Modify Plot Properties

title('Sine Wave', 'FontWeight','bold')
xlabel('x')
ylabel('sin(x)')
set(gca, 'Color', 'w')
set(gcf, 'MenuBar', 'none')
```

- 2 Click the down arrow on the **Publish** button  on the Editor toolbar, and select **Edit Publish Configuration for *file name***, where file name in this example is `sine_wave_f.m`.



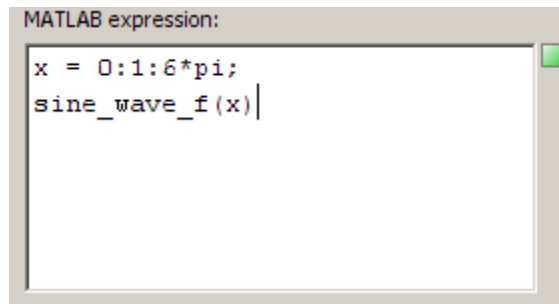
The Edit Configurations dialog box opens, with the default publish configuration template for `sine_wave_f.m`, as shown in the following figure.



- 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 a 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 file. In this example, delete the commented statements and replace them as shown in the following figure.



```
MATLAB expression:
x = 0:1:6*pi;
sine_wave_f(x) |
```

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 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 10-76 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:

- An output file, `sine_wave_f.html`
- A thumbnail file for the last image generated by the MATLAB code, `sine_wave_f.png`
- Image files created by the executable MATLAB code, `sine_wave_f_##.png`
- To create another publish configuration for the same file, click the plus button , and then select **Publish Configuration**.


See “Creating Multiple Publish Configurations for a File” on page 10-96 for details.

- To close the Edit Configurations dialog box, click **Close**. MATLAB saves the configuration and its association with the file.

After creating a configuration, you can view the MATLAB expression and use the configuration to publish the file without opening the Edit Configurations dialog box. See “Running an Existing Publish Configuration” on page 10-95 for details.

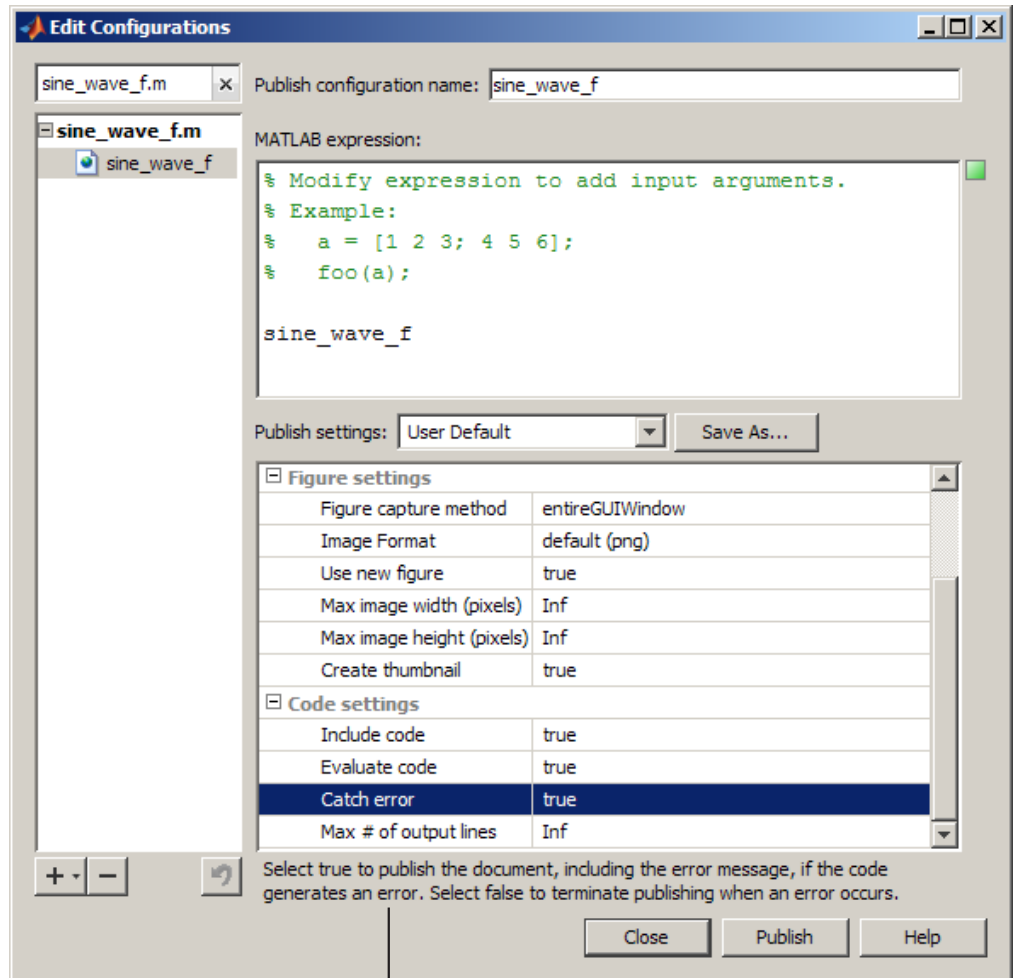
Specifying Publish Configuration Settings

This section describes how to specify new publish settings for a configuration. Publish settings enable you to specify the folder to which an output file is saved, how images generated by the code are captured and included in the output, and so on.

- 1** If the Edit 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 a MATLAB File” on page 10-68.

- 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.



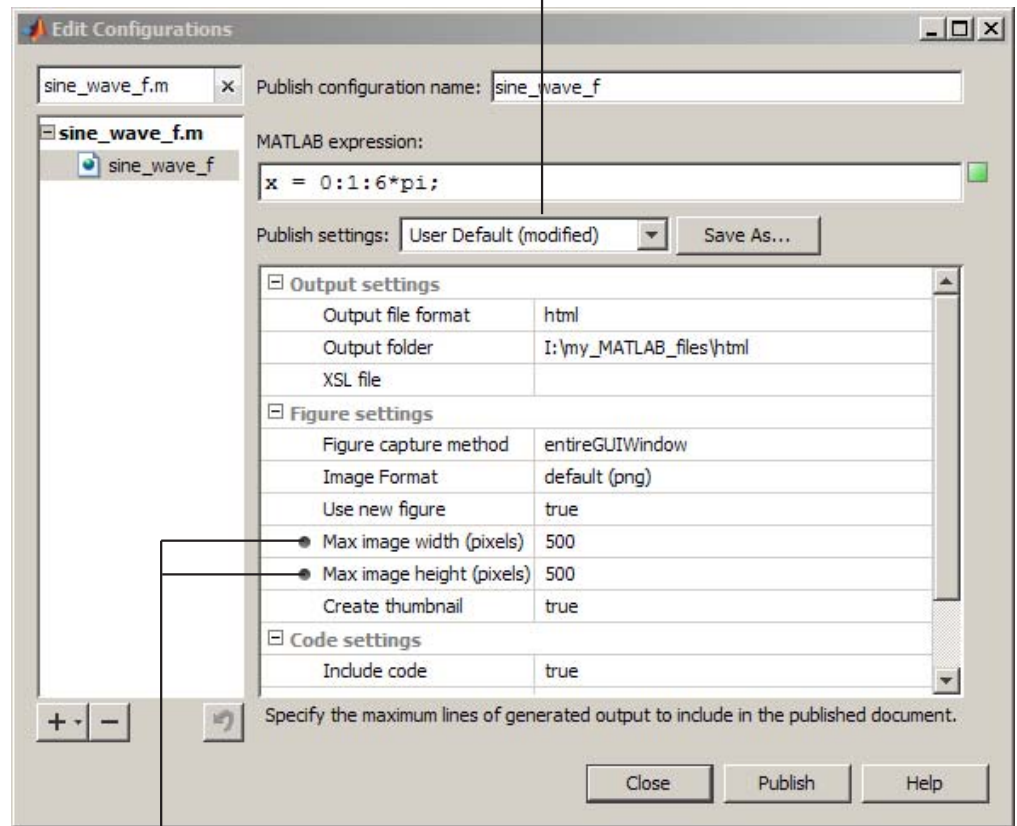
Description of Catch error property

- 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 10-76 for information about the various properties you can set.

Indicates that one or more settings differ from the saved User Default property settings



Indicates that these properties' values differ from the settings saved in the User Default publish settings.

- 5 Click **Publish** to preview the publication of the 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 install with MATLAB:

- **Factory Default**

The MATLAB installation includes this set of publishing properties for you to get started with publishing. It enables you to publish a MATLAB file to HTML quickly and view the results. You can use it to test the effect of changing settings. 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 MATLAB installation includes this set of publishing properties in anticipation that you will have a set of publishing properties that are common to most or all of your publishing configurations. Initially, **User Default** settings are identical to the **Factory Default** settings. 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. Select it from the **Publish settings** drop-down list, and then click **Overwrite**. However, you cannot overwrite the **Factory Default** publish settings.

Note When you overwrite a publish settings name, publish configurations that currently specify the old name do not have their publish properties updated to reflect the new settings. Instead, properties that have different values from the updated publish settings appear with a dot next to each of them.

- 8** In the Edit Configurations dialog box, do one of the following:
 - Click **Publish** to publish the 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 when you create or update a publish configuration. To access a publish configuration, open the file for which you want to 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” on page 10-76
- “Output folder” on page 10-77
- “XSL file” on page 10-77
- “Figure capture method” on page 10-77
- “Image format” on page 10-83
- “Use new figure” on page 10-83
- “Max image width” on page 10-89
- “Max image height” on page 10-89
- “Create thumbnail” on page 10-90
- “Include code” on page 10-90
- “Evaluate code” on page 10-90
- “Catch error” on page 10-92
- “Max # of output lines” on page 10-92

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 output file with the same name as the publish configuration that produced it and stores it, along with images that MATLAB generates from code, in the folder specified with the **Output folder** property.

Note MATLAB does not preserve syntax highlighting for MATLAB code published to these output types:

- LaTeX
- Microsoft Word (.doc)
- Microsoft PowerPoint (.ppt)

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 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.

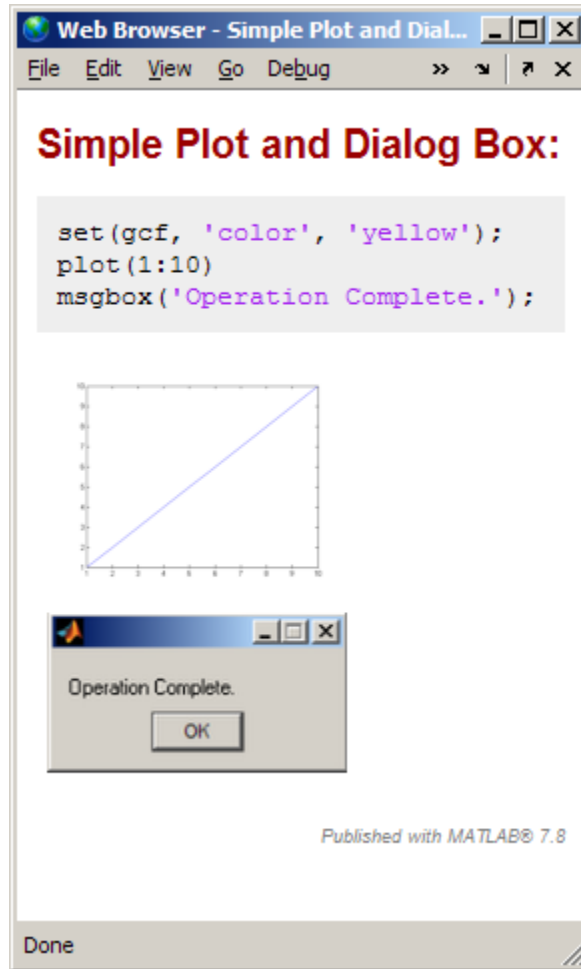
XSL file. Type the full path of the Extensible Stylesheet Language (XSL) file, that you want to use when you specify the **Output file format** as **HTML**, **XML**, or **LaTeX**. If you leave this field blank, MATLAB uses a default stylesheet that installs with the MATLAB software.

Figure capture method. Specify a figure capture method to indicate how you want figures and dialog boxes that the MATLAB code creates to appear when published.

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.)

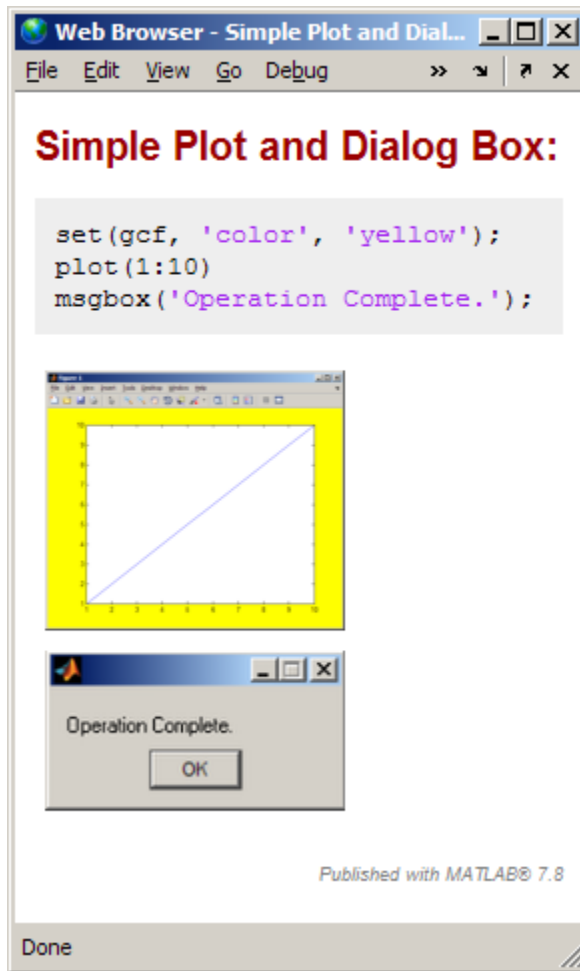
- To output the Figure data and complete dialogs boxes that the MATLAB code creates, set the **Figure capture method** to **entireGUIWindow**.

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 output.

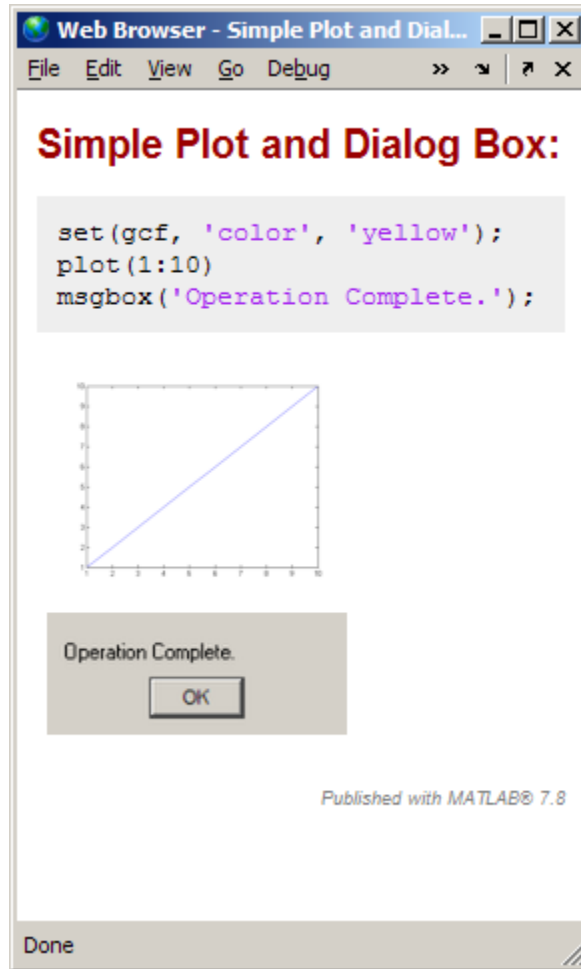


- To output a tutorial on using MATLAB, set the **Figure capture method** to **entireFigureWindow**.

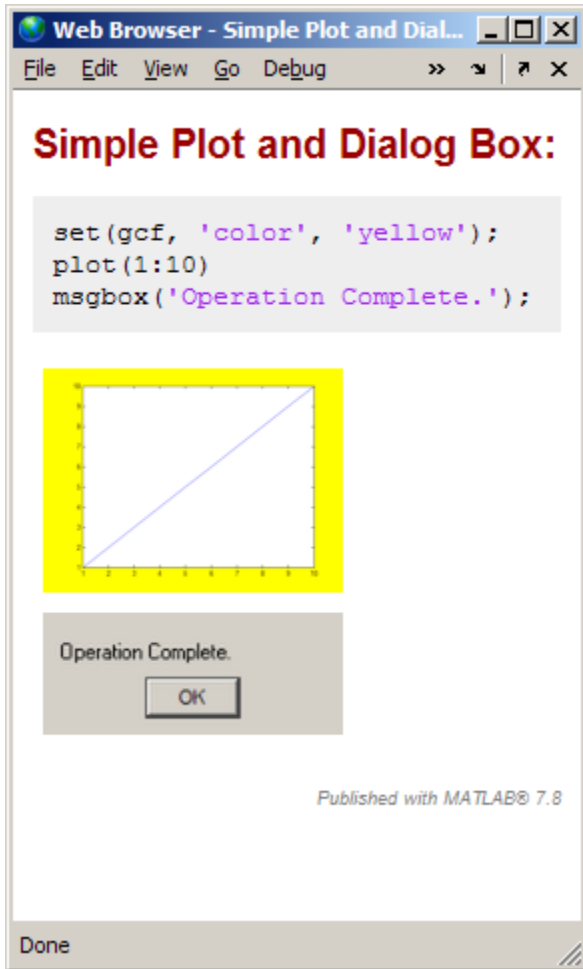
Notice that this method preserves the background color for figures and includes the window decorations for both figures and dialog boxes in the output.



- To output figures with the plot background set to white and dialog boxes without window decorations, set the **Figure capture method** to **print**.



- To output 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 the results listed in the preceding list and table, the `HandleVisibility` of your figures or dialog boxes might be atypical. For more information, see “`HandleVisibility Property`”.

Image format. Select the file type for images produced when publishing MATLAB files. The image file types available in the drop-down list depend on the **Figure capture method** you specify. You can choose any type available in the drop-down list, but for greatest compatibility select the default.

Use new figure. Set to **true** if you want MATLAB to create a Figure window with a white background and at the default size before publishing if the MATLAB 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 output use the characteristics of the figure you opened before publishing.

Note This preference applies to executable MATLAB code that generates a figure. It does not apply to figures included using the **Cell > Insert Text Markup > Image** menu option.

The following example demonstrates how to specify new Figure window properties for output 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 a MATLAB File” on page 10-68.
- 2 Create a Figure window by saving the following code in a file and then running it:

```
function createfigure
%CREATEFIGURE

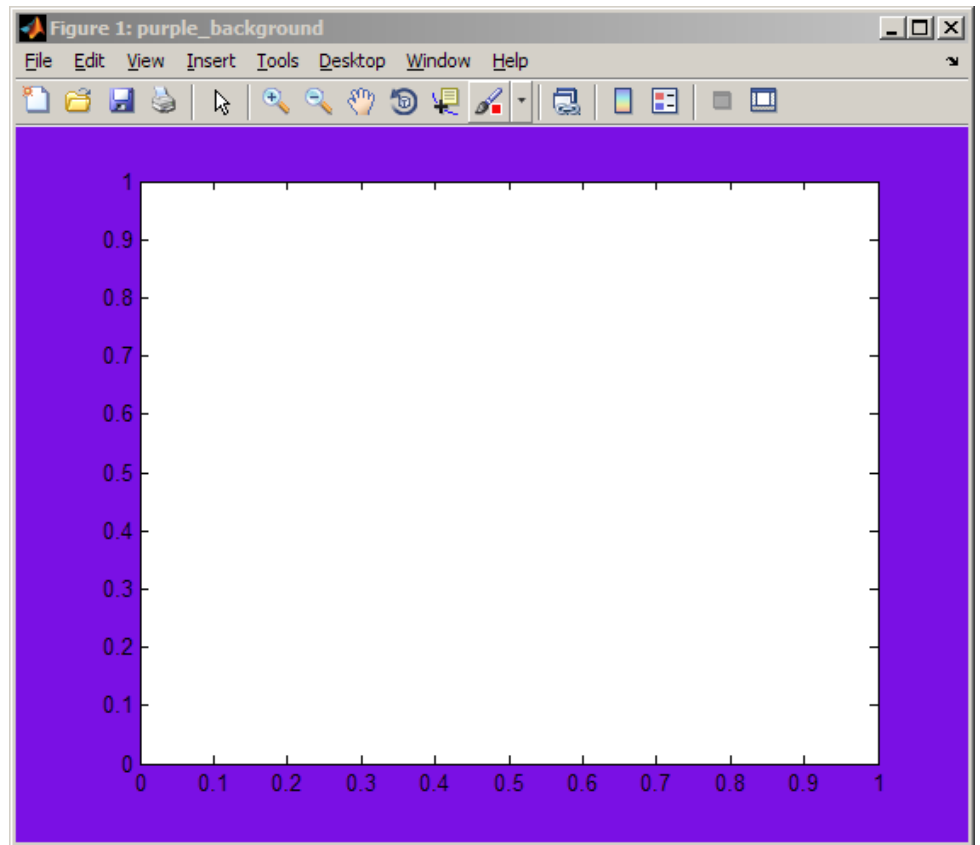
% Create figure
figure1 = figure('Name','purple_background',...
'Color',[0.4784 0.06275 0.8941]);
colormap('hsv');

% Create subplot
subplot(1,1,1,'Parent',figure1);
box('on');

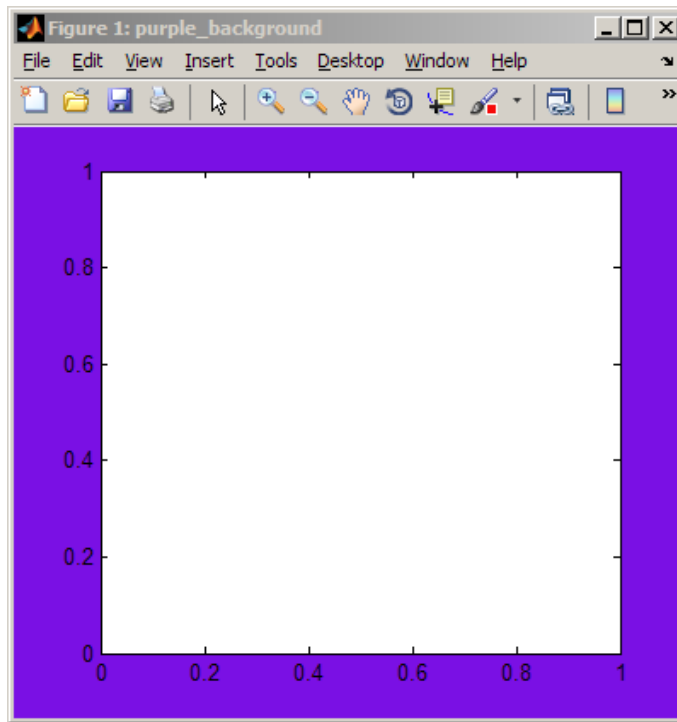
% Create xlabel
xlabel({' '});

% Create title
title({' '});
```

The following figure appears.



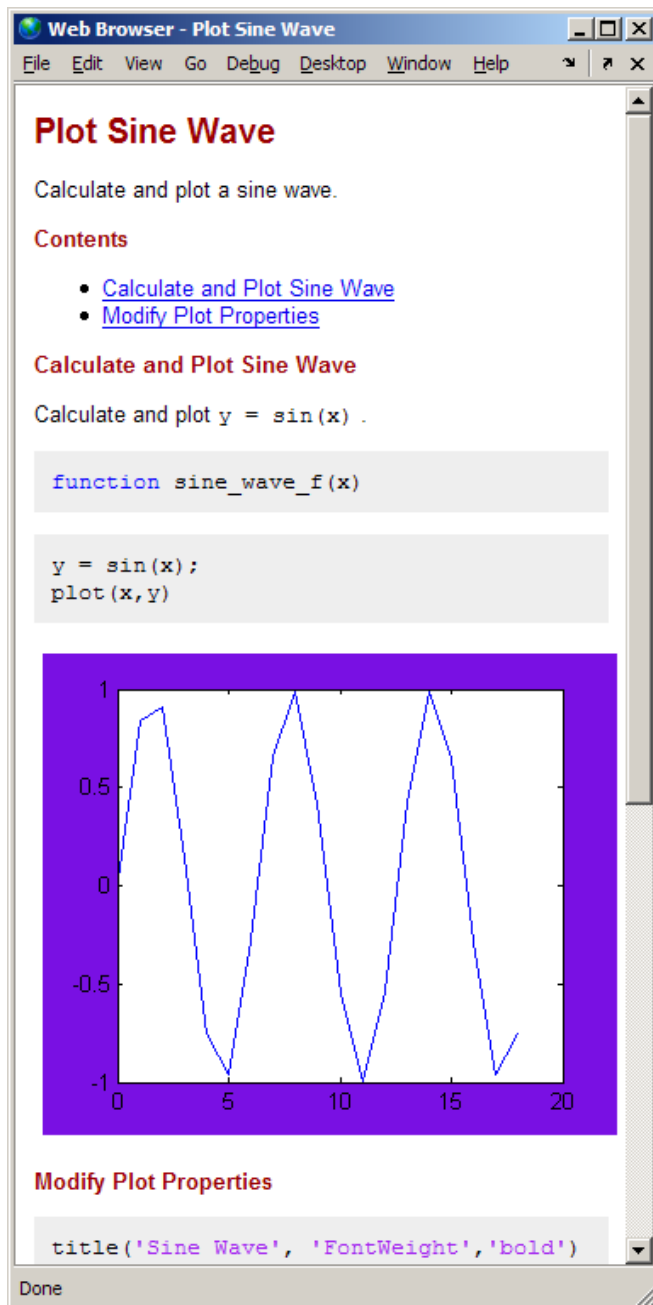
- 3 Reduce the size of the figure by dragging and dropping the edges. For example:



- 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**.

<input checked="" type="radio"/> Use new figure	false
---	-------

- 9 Click **Publish**. MATLAB publishes `sine_wave_f.m` as shown in the following figure.



The screenshot shows a web browser window with the title "Web Browser - Plot Sine Wave". The browser's address bar and menu bar are visible. The main content area displays the following:

Plot Sine Wave

Calculate and plot a sine wave.

Contents

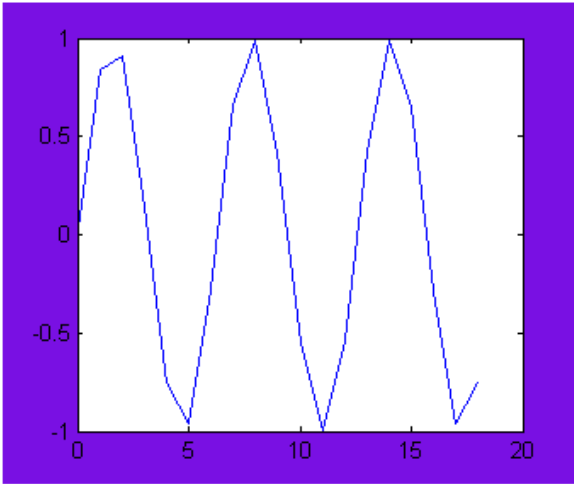
- [Calculate and Plot Sine Wave](#)
- [Modify Plot Properties](#)

Calculate and Plot Sine Wave

Calculate and plot $y = \sin(x)$.

```
function sine_wave_f(x)
```

```
y = sin(x);  
plot(x,y)
```



The plot shows a sine wave oscillating between -1 and 1. The x-axis is labeled from 0 to 20 in increments of 5. The y-axis is labeled from -1 to 1 in increments of 0.5. The plot is set against a purple background.

Modify Plot Properties

```
title('Sine Wave', 'FontWeight','bold')
```

Done

Max image width. Overwrite the current value to restrict the width of images in the output. Note the following about this property:

- It applies only to images that the code generates. It does not apply to images you include using the method described in “External Graphics” on page 10-32.
- 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 the **Max image width** and **Max image height** properties to resize the image, then MATLAB maintains the aspect ratio. It does so 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 output. Note the following about this property:

- It applies only to images that the code generates. It does not apply to images you include using the method described in “External Graphics” on page 10-32.
- 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 the **Max image width** and **Max image height** properties to resize the image, then MATLAB maintains the aspect ratio. It does so 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 file in HTML pages. If you create your own demos and include them in the Help browser **Demos** pane using 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 MATLAB code in the output. Set to **false** to have MATLAB exclude the code from all output file formats except HTML. When the output file format is HTML, MATLAB inserts the MATLAB code in the output file as an HTML comment. Therefore, when viewed in a Web browser, for example, the MATLAB code does not display.

Use the MATLAB `grabcode` function if you want to extract the MATLAB code from the output 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 output with colleagues, they can view it in a Web browser. If your colleagues want to see the MATLAB code that generated the output, they can issue the following command from the folder containing `sine_wave_f.html`:

```
grabcode('sine_wave_f.html')
```

MATLAB opens the file that created `sine_wave_f.html` in the Editor.

See “Creating a Publish Configuration for a MATLAB File” on page 10-68 for the `sine_wave_f.m` code.

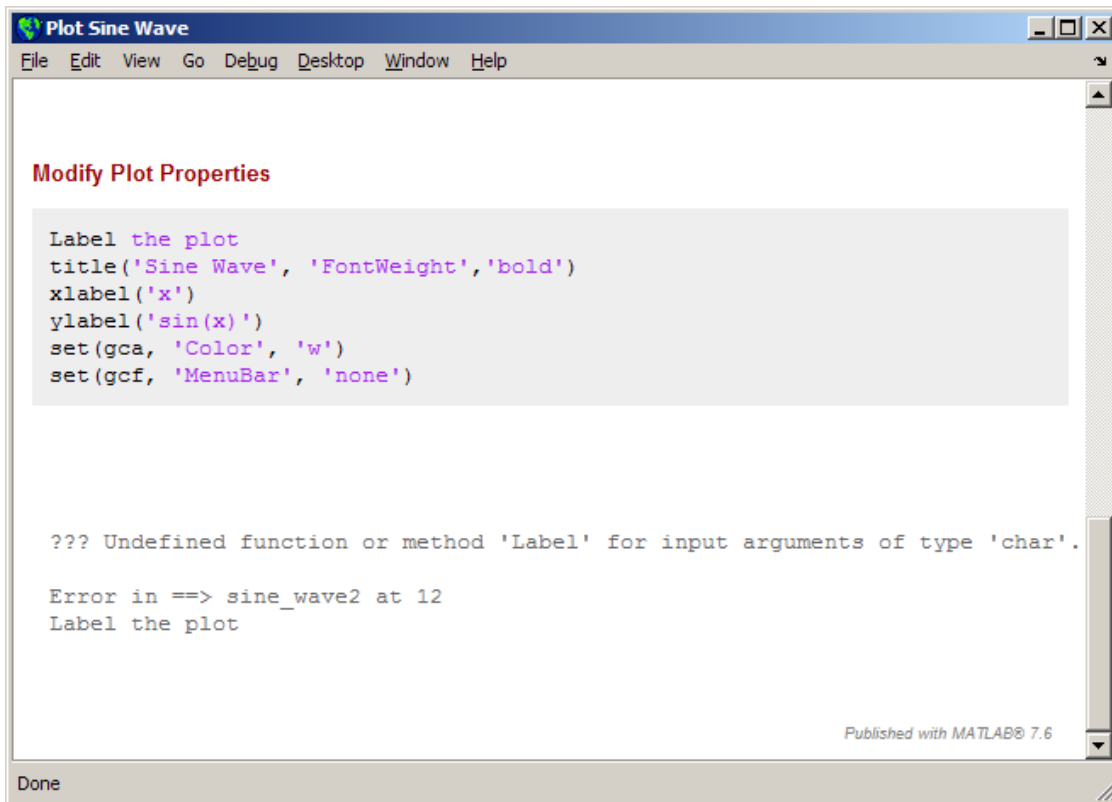
Evaluate code. Set to **true** to direct MATLAB to evaluate the MATLAB code while publishing the results and include the results in the output document. Also specify the **Max # of output lines** property to specify the maximum number of lines you want to include in the output. This property is helpful when you have code that produces much output and you only want to include a sample of it.

Set to **false**, to:

- Direct MATLAB to not evaluate the code, nor include code results when publishing a file.
- Use the `publish` function to publish the file that contains the `publish` function. Otherwise, MATLAB attempts to publish the file recursively.

Because MATLAB does not evaluate the code when you set this property to **false**, there can be invalid code in the file. Therefore, consider first running the file with this property set to **true**.

For example, suppose you include comment text, `Label the plot`, in a file, but forget to preface it with the comment character. If you publish the document to HTML, and set **Evaluate code** to **true**, the output includes the error, such as shown in the following figure.



The screenshot shows a MATLAB window titled "Plot Sine Wave" with a menu bar (File, Edit, View, Go, Debug, Desktop, Window, Help). The main area contains a code cell with the following MATLAB code:

```
Label the plot
title('Sine Wave', 'FontWeight','bold')
xlabel('x')
ylabel('sin(x)')
set(gca, 'Color', 'w')
set(gcf, 'MenuBar', 'none')
```

Below the code, an error message is displayed:

```
??? Undefined function or method 'Label' for input arguments of type 'char'.

Error in ==> sine_wave2 at 12
Label the plot
```

At the bottom right of the window, it says "Published with MATLAB® 7.6". The status bar at the bottom left says "Done".

Catch error. Set to **true** to direct MATLAB to publish and include the error message text in the output 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 output.

For example, suppose your MATLAB code includes a loop, such as the following:

```
for n = 1:100
    disp(x)
end;
```

If you publish the code, then by default, all 100 lines generated by the preceding code appears in the output. If you want to include a smaller representative sample, 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 publish configuration.

For example, suppose that you frequently publish your files using the factory installed **User Default** settings, with a few exceptions. You want to change the factory installed **User Default** settings to:

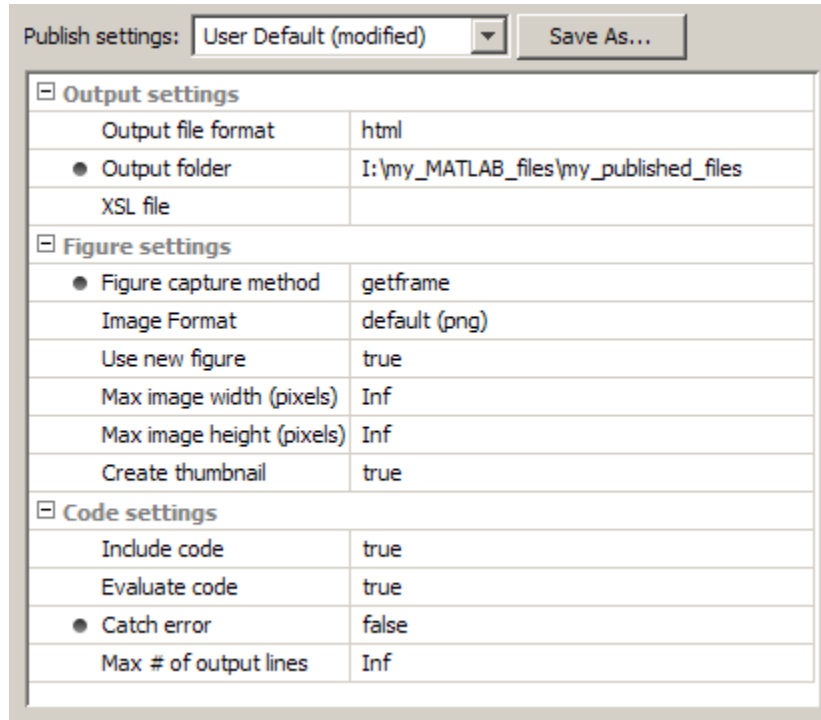
- Save the output files to `I:\my_MATLAB_files\my_published_files`
- Use the `getframe` figure capture method
- Terminate publishing if an error occurs while the MATLAB code is being evaluated

Update the **User Default** publish settings, as follows:

- 1** If the **Edit 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 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 file to a Microsoft Word document. Follow these steps:

1 In the Editor, open the 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 Configurations dialog box opens. Notice that the **Publish settings** is **User Default** and the publish settings properties table contains the values you set in the preceding list of steps.

- 3 If you want, adjust the MATLAB expression.
- 4 Change the **Output file format** from **html** to **doc**.
- 5 Click **Save As**.

The Save Publish Settings dialog box opens.


- 6 In the **Settings name** box, type a name for the new group of publish settings. For example, `WordDefault`.
- 7 Click **Save**.

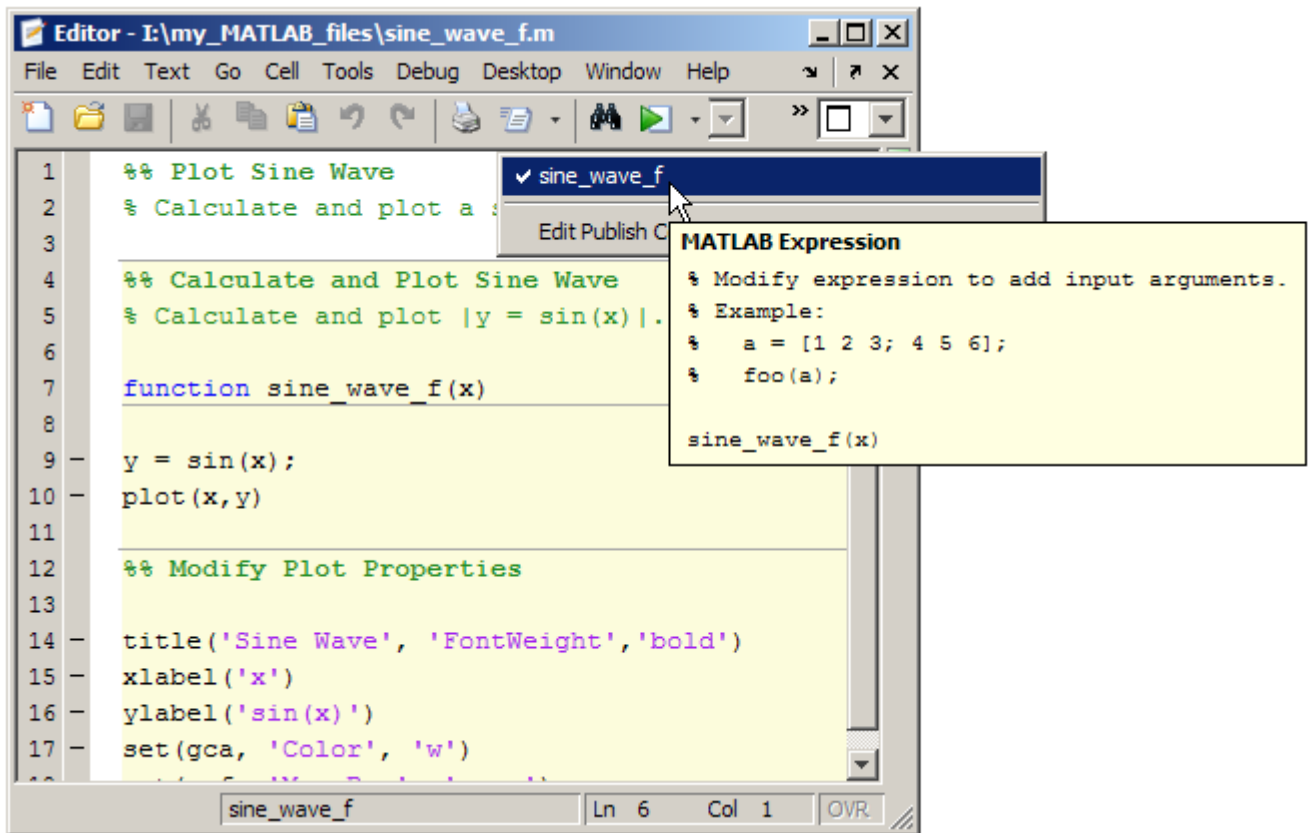
Now you can use any one of the following publish settings as the basis for new publish settings, for the next publish configuration you create:

- 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 Configurations dialog box, as follows:

- 1 In the Editor toolbar, click the down arrow on the **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 file using the named configuration.



- 2 To use the publish configuration, select a configuration name. MATLAB publishes the 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 MATLAB function before evaluating and publishing the file. (To see how to set the MATLAB expression, see “Creating a Publish Configuration for a MATLAB File” on page 10-68.)

Creating Multiple Publish Configurations for a File

You can create multiple publish configurations for a given file. You might do this to publish the 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 file. Then, any time you publish the file, you can choose 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.
 - Set publishing properties to evaluate and include the code in the output published to HTML.
 - Set publishing properties to 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 10-97
- “Steps for Publishing `sine_wave_f.m` to HTML” on page 10-100
- “Steps for Publishing `sine_wave_f.m` to Microsoft® PowerPoint®” on page 10-103
- “Steps for Publishing `sine_wave_f.m` to PDF” on page 10-105

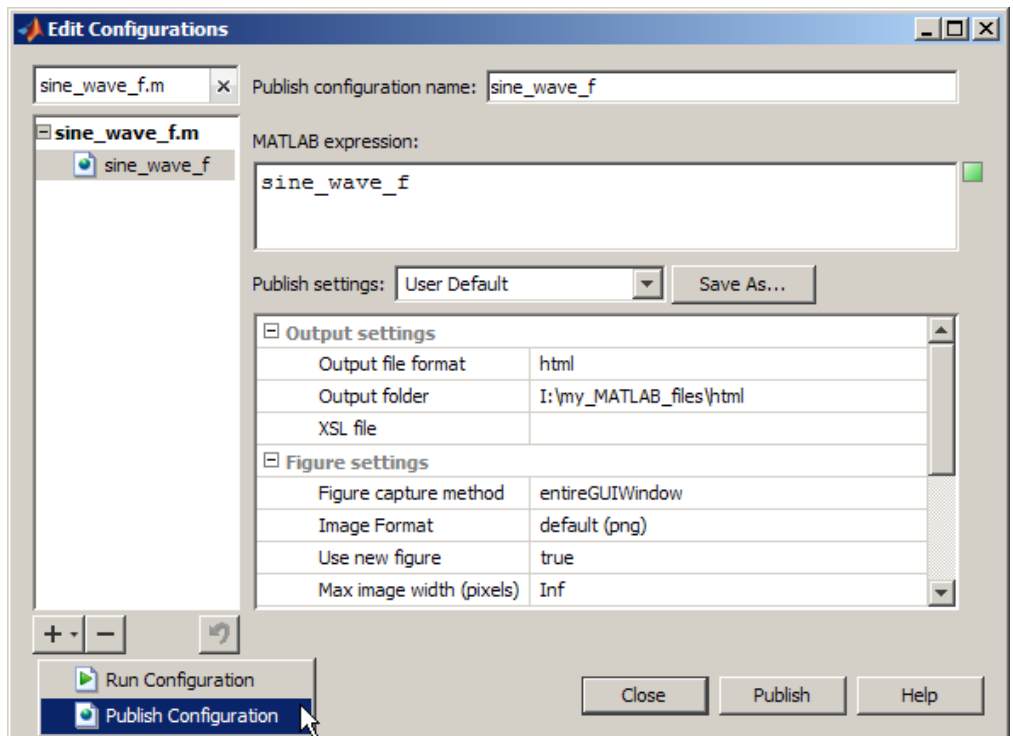
Example of Publishing `sine_wave_f.m` to Microsoft Word

The following steps provide an example of settings you might use when you want to publish a file to Word. This example uses the `sine_wave_f.m` file, the code for which appears in “Creating a Publish Configuration for a MATLAB File” on page 10-68.

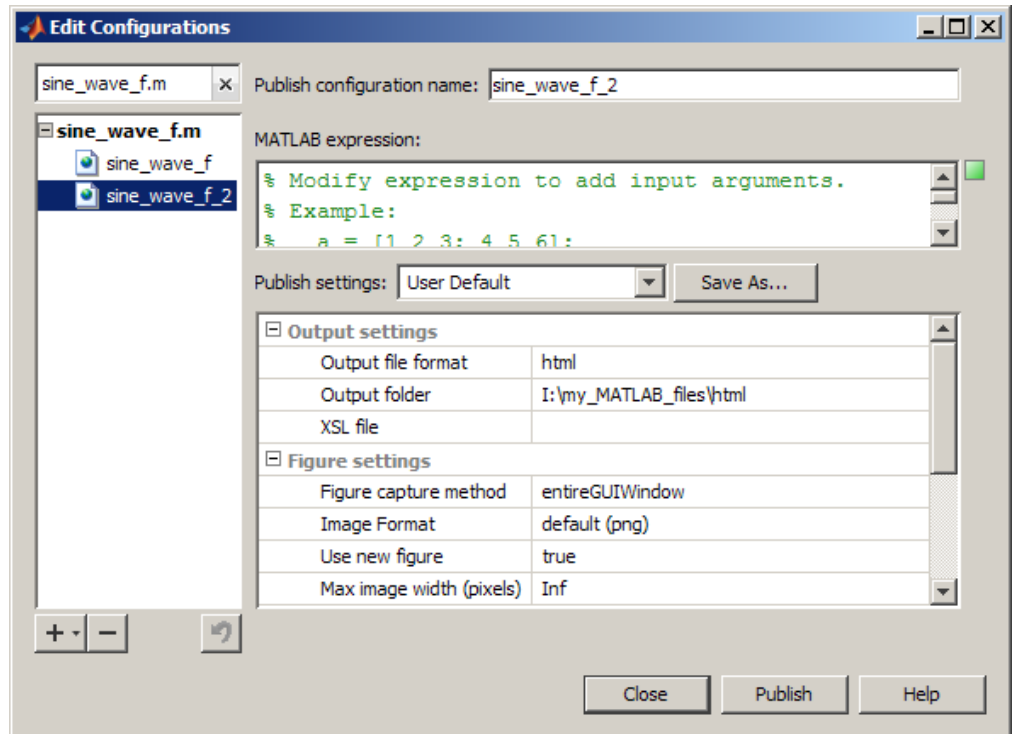
- 1 Copy `sine_wave_f.m` to your current folder. If you have write permission to your current folder, you can type the following in the Command Window to copy the file from the MATLAB root folder:

```
copyfile(fullfile(matlabroot,'help','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 files and configurations, click the down arrow next to the **Add** button \pm , 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 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 output is a Word document that includes the code, its output and any errors the code might 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 MATLAB code evaluates as expected and publishes to Word as expected, consider running the spelling and grammar checker in Word to be sure that the comments in your code 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 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 folder:

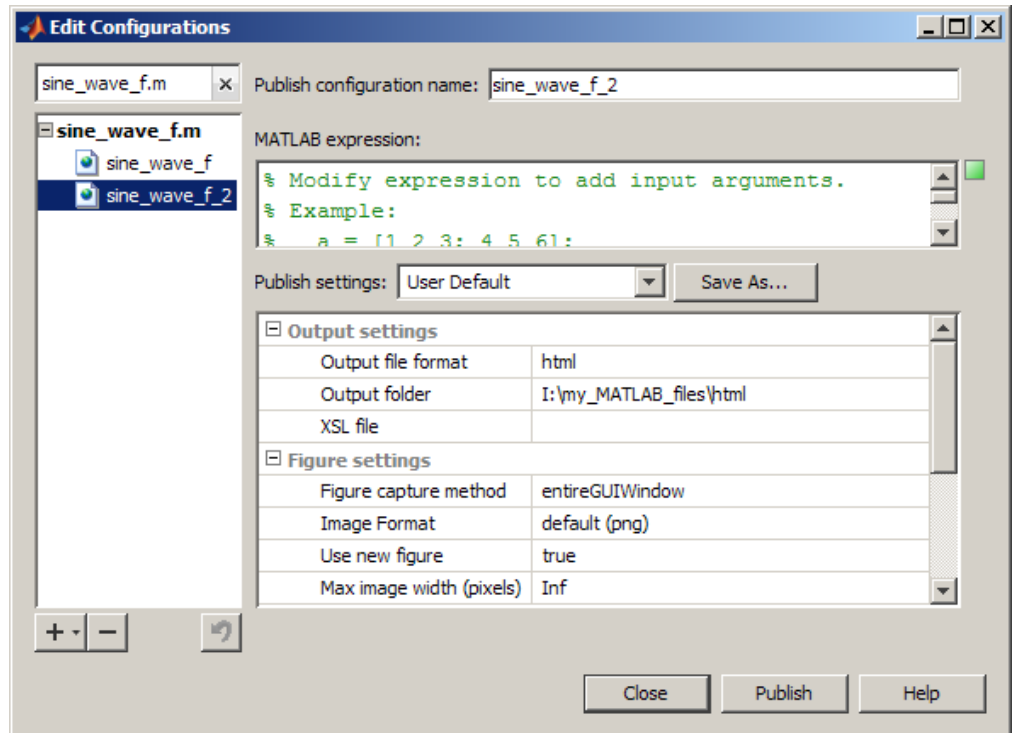
```
copyfile(fullfile(matlabroot,'help','techdoc','matlab_env',...  
'examples','sine_wave_f.m'),'.','f')
```

- 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 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 files and configurations, click the down arrow next to the **Add** button \downarrow , 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`.



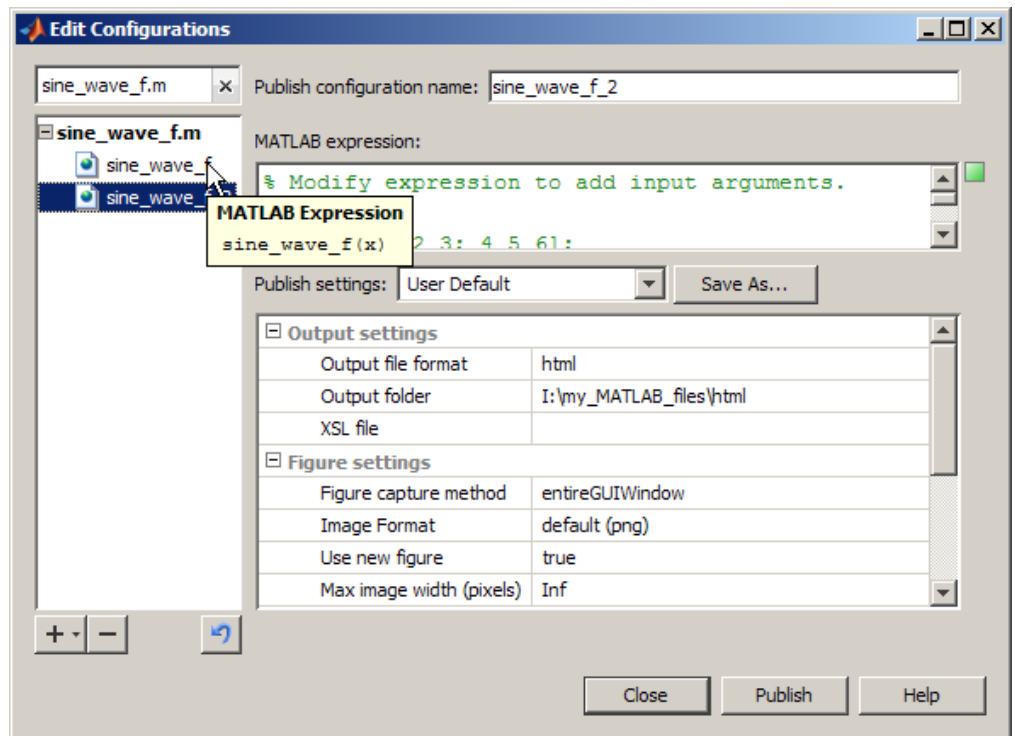
- 5 In the **Publish configuration name** field, replace `sine_wave_f_n` with `sine_wave_html`.
- 6 In the **MATLAB expression** field, replace the default expression with the following:

```

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 file publishes to an HTML document, including the code, its output and any errors the code might 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**.

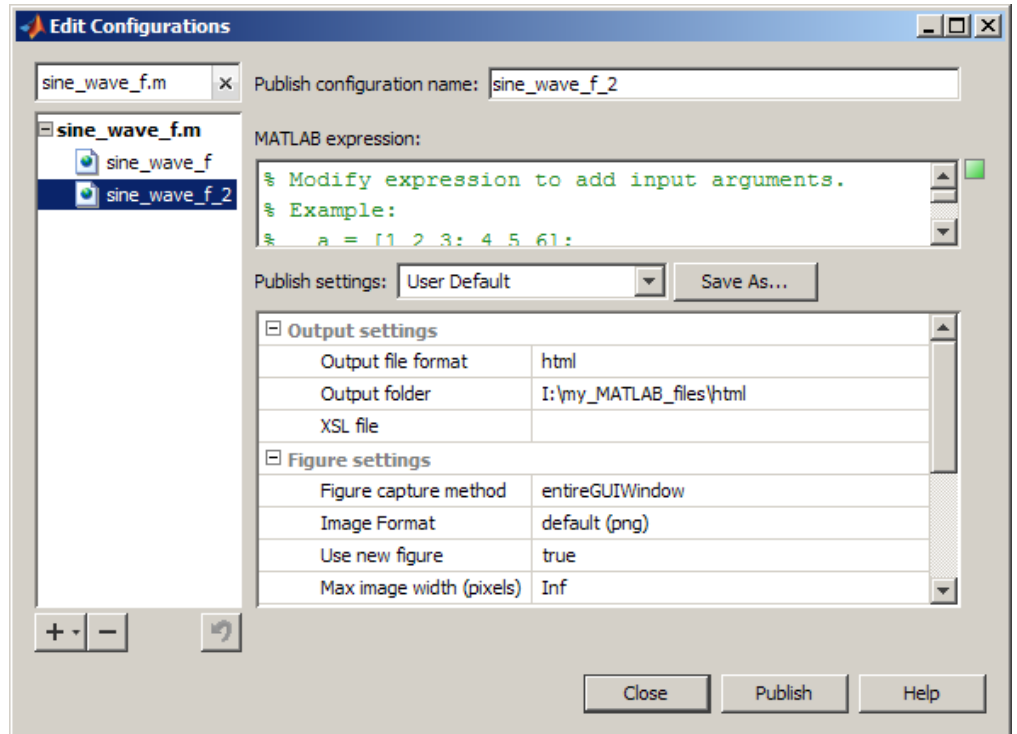
Steps for Publishing `sine_wave_f.m` to Microsoft PowerPoint

These steps provide an example of creating a configuration for `sine_wave_f.m`, that publishes the 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, type the following in the Command Window to copy the file from the MATLAB root folder:


```
copyfile(fullfile(matlabroot, 'help', '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 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_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 appears in “Creating a Publish Configuration for a MATLAB File” on page 10-68.


- 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 folder:

```
copyfile(fullfile(matlabroot, 'help', '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 files and configurations, click the down arrow next to the **Add** button , and then select **Publish Configuration**.

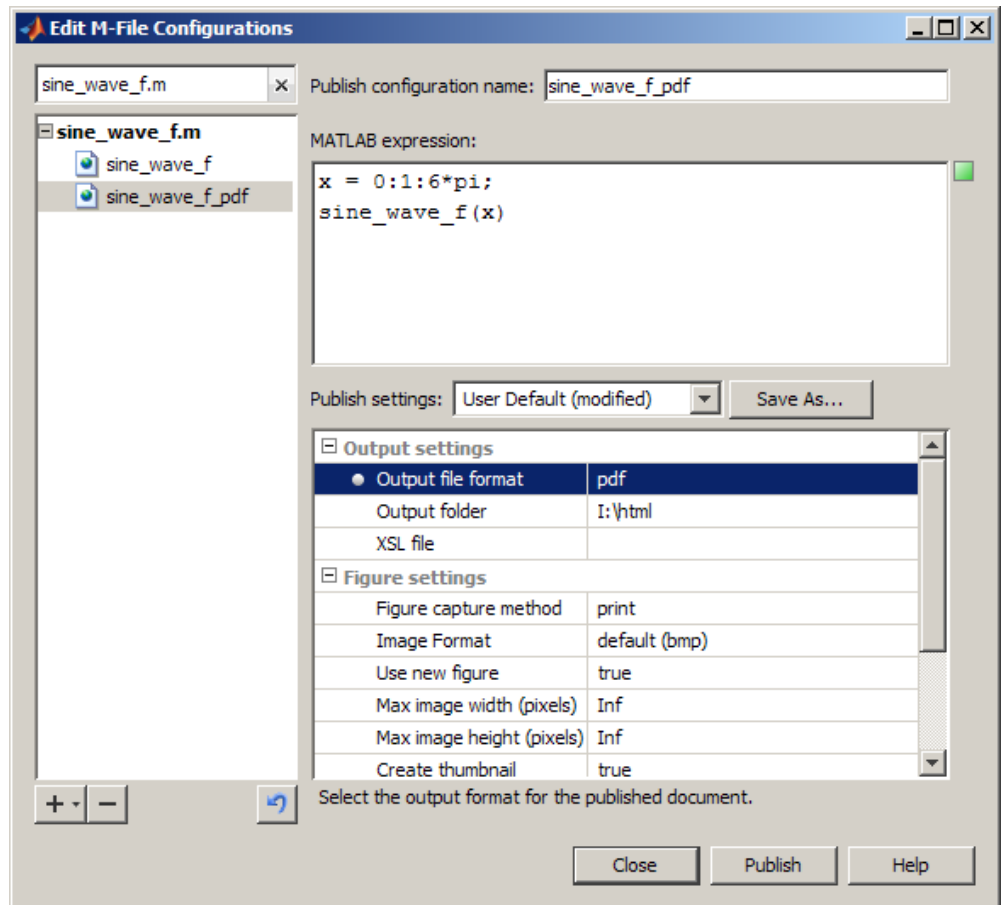
MATLAB creates a publish configuration, `sine_wave_f_n`, where the value of n depends on the number of publish configurations you have previously created for `sine_wave_f`.

- 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:

Plot Sine Wave	
Table of Contents	
Calculate and Plot Sine Wave	1
Modify Plot Properties	1
Introductory text → Calculate and plot a sine wave.	

- 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 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, only do this if you have not already created configurations on the second system. In addition, because this file might contain references to file paths, be sure that the specified files and paths exist on the second system.

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 Configurations dialog box might be overwritten later. Each time you save a configuration using the Edit 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 8-52.

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 8-53.

Removing Publish Configurations

If you no longer need a publish configuration because you do not use it or because you deleted the 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 8-55 for details.

Reassociating and Renaming Publish Configurations

Each publish configuration is associated with a specific file. If you move or rename the file, redefine the association. If you delete a file, consider deleting the associated configurations, or associating them with a different 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 8-56 for details.

Summary of Options for Presenting Your Code to Others

In addition to publishing, MATLAB provides other options for presenting your code to others. The following table summarizes the options and presents them in order of ease of implementing.

Method	Description	Output Formats	Details
Command line help	Using comments at the start of a MATLAB program, enables the display of help comments when you type <code>help function-name</code> or <code>help class-name</code> in the Command Window.	<ul style="list-style-type: none"> • ASCII text 	See “Add Help for Your Program Files”.
Publish	Using comments and basic markup, publish a document that includes comment text, MATLAB code, and code output. You can produce numerous output formats from the same code.	<ul style="list-style-type: none"> • XML • HTML • LaTeX • Microsoft Word (.doc) • Microsoft PowerPoint (ppt) • PDF 	See “Overview of Publishing MATLAB Code” on page 10-2.
Notebook	Using Microsoft Word, create electronic or printed records of MATLAB sessions, class notes, textbooks or technical reports. You must have Microsoft Word software installed.	<ul style="list-style-type: none"> • Microsoft Word (.doc) 	See “About MATLAB Notebooks” on page 11-2
Help Browser Topics	Create HTML and XML files to provide your own MATLAB help topics for viewing from the MATLAB Help browser or the Web.	<ul style="list-style-type: none"> • HTML 	See “Add Documentation to the Help Browser”

Method	Description	Output Formats	Details
Help Browser Demos	Create HTML files to provide your own demos, including videos for viewing from the MATLAB Help browser.	<ul style="list-style-type: none">• Text• HTML	See “Add Demos to the Help Browser”
MATLAB Report Generator	Using MATLAB Report Generator build complex reports. You must have MATLAB Report Generator software installed.	<ul style="list-style-type: none">• RTF• PDF• Word• HTML• XML	See <i>MATLAB Report Generator User’s Guide</i>

Creating a MATLAB Notebook to Publish to Microsoft Word

You can use the `notebook` command to create electronic or printed records of MATLAB sessions, class notes, textbooks, or technical reports to Microsoft Word. This document is known as a MATLAB Notebook. As an alternative, consider using the MATLAB `publish` command. For more information, see Chapter 10, “Publishing MATLAB Code”.

Note The `notebook` command is supported only on Windows systems that have Microsoft Word and its template directory installed on mounted drives (they cannot be on a UNC path). For supported versions of Word, see “Configuring the MATLAB notebook Software” on page 11-10.

- “About MATLAB Notebooks” on page 11-2
- “Configuring the MATLAB notebook Software” on page 11-10
- “Defining MATLAB Commands as Input Cells for a MATLAB Notebook” on page 11-12
- “Evaluating MATLAB Commands in a MATLAB Notebook” on page 11-17
- “Printing and Formatting a MATLAB Notebook” on page 11-23
- “Notebook Feature Reference” on page 11-28

About MATLAB Notebooks

In this section...
“Contents of MATLAB Notebooks” on page 11-2
“Creating or Opening a MATLAB Notebook” on page 11-2
“Entering Commands in a MATLAB Notebook” on page 11-7
“Protecting the Integrity of Your Workspace in MATLAB Notebooks” on page 11-8
“Ensuring Data Consistency in MATLAB Notebooks” on page 11-8
“Debugging and MATLAB Notebooks” on page 11-9

Contents of MATLAB Notebooks

Using the `notebook` command, you can create a Microsoft Word document, that contains text, MATLAB commands, and the output from MATLAB commands.

You can think of this document as a record of an interactive MATLAB session annotated with text, or as a document embedded with live MATLAB commands and output. This documentation refers to this Microsoft Word document as a MATLAB Notebook.

Creating or Opening a MATLAB Notebook

This section includes information on performing the following tasks:

- “Issuing the `notebook` Command from the MATLAB Desktop” on page 11-3
- “Creating a MATLAB Notebook” on page 11-4
- “Opening an Existing MATLAB Notebook” on page 11-5
- “Converting a Word Document to a MATLAB Notebook” on page 11-6

Issuing the notebook Command from the MATLAB Desktop

If you are running notebook for the first time since you installed a new version of MATLAB, follow the instructions in “Configuring the MATLAB notebook Software” on page 11-10.

Then, to create a MATLAB Notebook from within MATLAB desktop, type the following in the Command Window:

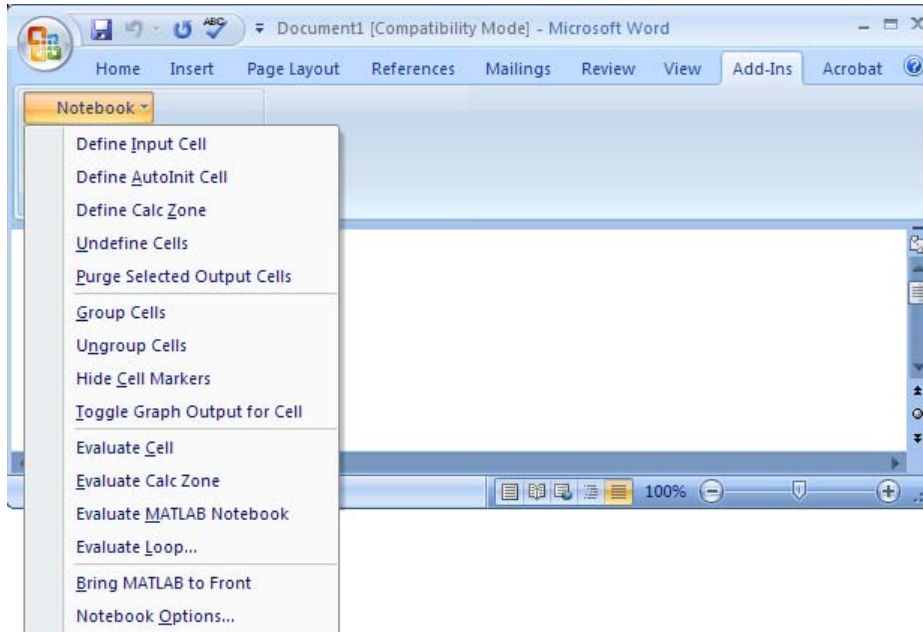
```
notebook
```

The notebook command starts Word on your system and creates a MATLAB Notebook, called Document1.

When Word is opening, if a dialog box appears asking you to enable or disable macros, choose to enable macros. The notebook command 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 the MATLAB notebook Software” on page 11-10.

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.
- In Word 2007 and later releases, notebook adds the **Notebook** menu to the Word **Add-Ins** tab, as shown in the following illustration.

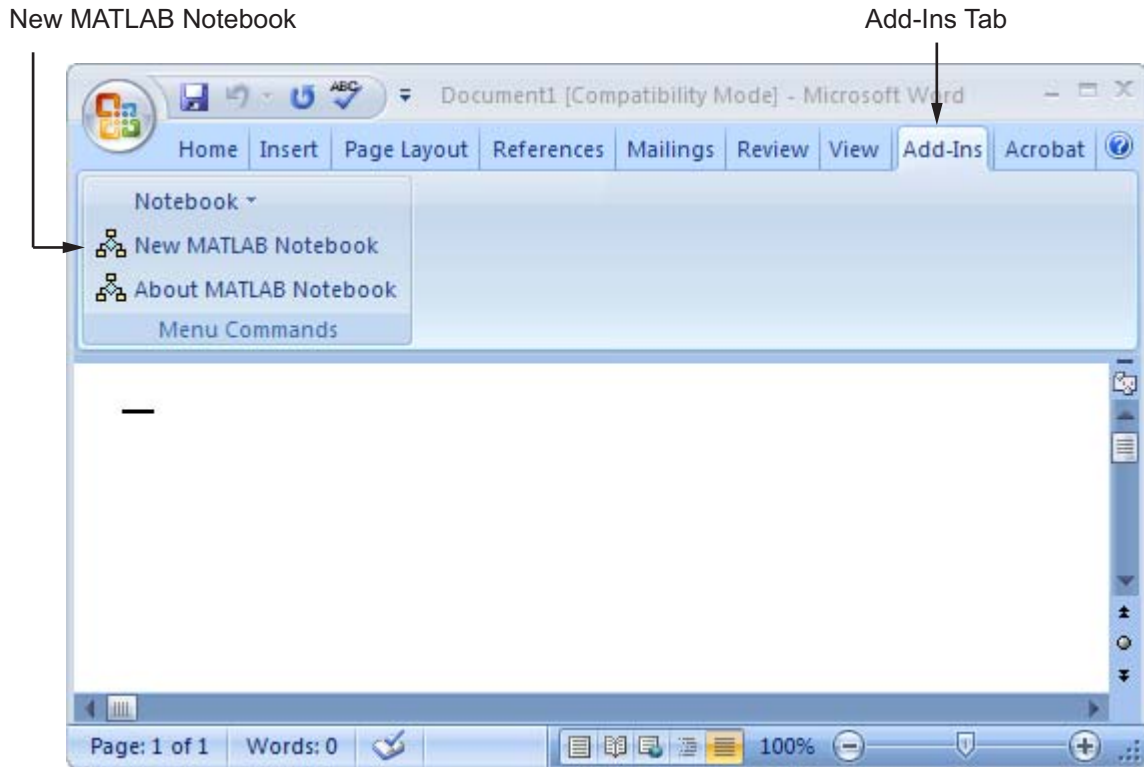


Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Creating a MATLAB Notebook

After issuing the notebook command, you can create a MATLAB Notebook as follows:

- In Word 2002, and 2003, select **File > New MATLAB Notebook**
- In Word 2007 and later releases, select **Add-Ins > New MATLAB Notebook**, as shown in the following figure.



Microsoft product screen shot reprinted with permission from Microsoft Corporation.

Opening an Existing MATLAB Notebook

You can use the `notebook` command to open an existing MATLAB Notebook, as shown in the following code, where *filename* is the notebook you want to open.

```
notebook filename
```

Alternatively, you can double-click a notebook file in a Windows file management tool, such as Explorer.

When you double-click a notebook, Microsoft Word opens it and starts MATLAB if it is not already running.

Converting a Word Document to a MATLAB Notebook

To convert a Word document to a MATLAB Notebook, insert it into a notebook file, as described in the steps that follow. Choose the set of steps that corresponds to the version of Word you are using:

- “Microsoft Word 2002, or 2003” on page 11-6
- “Microsoft Word 2007 and Later Releases” on page 11-6

Microsoft Word 2002, or 2003.

- 1** Create a MATLAB Notebook.

For details, see “Creating or Opening a MATLAB Notebook” on page 11-2.

- 2** From the **Insert** menu, select **File**.
- 3** Select the file you want to convert.
- 4** Click **OK**.

Microsoft Word 2007 and Later Releases.

- 1** Create a MATLAB Notebook.

For details, see “Creating or Opening a MATLAB Notebook” on page 11-2.

- 2** From the **Insert** tab, in the **Text** group, click the arrow next to **Object**.
- 3** Select **Text from File**, as shown in the image that follows.

The Insert File dialog box opens.

- 4** 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 Commands in a MATLAB Notebook

Note A good way to learn how to use notebook is to open the sample MATLAB Notebook, `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 a notebook 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 MATLAB Notebook.

```
a = magic(3)
```

To execute the MATLAB magic command in this document, follow the steps described in these sections:

- “Defining MATLAB Commands as Input Cells for a MATLAB Notebook” on page 11-12
- “Evaluating MATLAB Commands in a MATLAB Notebook” on page 11-17

MATLAB displays the output of the command in the Word document in an output cell.

Protecting the Integrity of Your Workspace in MATLAB Notebooks

When you work on more than one MATLAB Notebook in a single word-processing session, note that:

- Each notebook uses the same “copy” of MATLAB.
- All notebooks share the same workspace.

If you use the same variable names in more than one notebook, data used in one notebook can be affected by another notebook. You can protect the integrity of your workspace by specifying the `clear` command as the first autoinit cell in the notebook.

Ensuring Data Consistency in MATLAB Notebooks

You can think of a MATLAB Notebook as a sequential record of a MATLAB session. When executed in order, from the first MATLAB command to the last, the notebook accurately reflects the relationships among these commands.

If, however, you change an input cell or output cell as you refine your notebook, it can contain inconsistent data. Input cells that depend on either the contents or the results of the changed cells do not automatically recalculate when you make a change.

When working on a notebook, consider selecting **Evaluate MATLAB Notebook** periodically to ensure that your notebook data is consistent. You can also use calc zones to isolate related commands in a section of the

notebook, and then use **Evaluate Calc Zone** to execute only those input cells contained in the calc zone.

Debugging and MATLAB Notebooks

Do not use debugging functions or the Editor while evaluating cells within a MATLAB Notebook. Instead:

- 1** Complete debugging files from within MATLAB.
- 2** Clear all the breakpoints.
- 3** Access the file using notebook.

If you debug while evaluating a notebook, you can experience problems with MATLAB.

Configuring the MATLAB notebook Software

After you install MATLAB Notebook software, but before you begin using it, specify that Word can use the notebook macros, and then configure notebook. (The notebook software installs as part of the MATLAB installation process on Microsoft Windows platforms. For more information, see the MATLAB installation documentation.)

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 running 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 MATLAB Notebook features, but still maintain a high security level for other macros you use in Word.
- In Word 2007 and later releases, follow the Word help instructions in the topic entitled “Enable or disable macros in Office documents.”

To configure MATLAB Notebook software, type the following in the MATLAB Command Window:

```
notebook -setup
```

MATLAB configures MATLAB Notebook software 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
```

- When MATLAB configures the software, it:
 - 1 Accesses the Microsoft Windows system registry to locate Microsoft Word and the Word templates folder. It also identifies the version of Word.
 - 2 Copies the m-book.dot template to the Word templates folder

The MATLAB Notebook software supports Word versions 2002, 2003, 2007, and 2010.

If you have previously configured the software, typing `notebook` in the MATLAB Command Window, starts Microsoft Word and creates a MATLAB Notebook. 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 reconfigure the software by typing:

```
notebook -setup
```

Defining MATLAB Commands as Input Cells for a MATLAB Notebook

In this section...

“Defining Input Cells for a MATLAB Notebook” on page 11-12

“Defining Cell Groups for a MATLAB Notebook” on page 11-13

“Defining Autoinit Input Cells for a MATLAB Notebook” on page 11-14

“Defining Calc Zones for a MATLAB Notebook” on page 11-14

“Converting an Input Cell to Text in a MATLAB Notebook” on page 11-15

For information about evaluating the input cells you define, see “Evaluating MATLAB Commands in a MATLAB Notebook” on page 11-17.

Defining Input Cells for a MATLAB Notebook

To define a MATLAB command in a Word document as an input cell, follow these steps:

- 1 Type the command into the MATLAB Notebook as text. For example,

This is a sample MATLAB Notebook.

```
a = magic(3)
```

- 2 Position the cursor anywhere in the command, and then 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. This defines the MATLAB command as an input cell:

This is a sample MATLAB Notebook.

```
[a = magic(3)]
```

Note how the character font of the text in the input cell changes to a bold, dark green color and appears 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 MATLAB Notebook Template” on page 11-23.

Defining Cell Groups for a MATLAB 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 immediately after the group, cell groups are useful when you need several MATLAB commands. For instance, to define a graphic fully.

If you define all the MATLAB commands that produce a graphic as a cell group, and then evaluate that cell group, it 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” on page 11-18 for information about evaluating a cell group. For information about ungrouping a cell group, see “Ungroup Cells” on page 11-35.

Creating a Cell Group

To create a cell group:

- 1 Use the mouse to select the input cells that are to make up the group.
- 2 Select **Notebook > Group Cells** or press **Alt+G**.

The selected cells convert into a cell group and cell markers convert to 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, they are deleted.

- A cell group cannot contain text. If the selection includes text, the text appears after the cell group. However, if the text precedes the first input cell in the selection, it remains where it is.
- If you select part or all of an output cell, but not its input cell, the cell group still includes the input cell.

When you create a cell group, it is an input cell, unless its first line is an autoinit cell. In that case, the group is an autoinit cell.

Defining Autoinit Input Cells for a MATLAB Notebook

You can use *autoinit cells* to specify MATLAB commands be evaluated automatically each time a MATLAB Notebook opens. This is a quick and easy way to initialize the workspace. *Autoinit cells* are input cells with the following additional characteristics:

- The autoinit cells evaluate when MATLAB Notebook opens.
- The commands in autoinit cells display in dark blue characters.

Autoinit cells are otherwise identical to input cells.

Creating an Autoinit Cell for a MATLAB 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 in a MATLAB Notebook” on page 11-17 for information about evaluating autoinit cells.

Defining Calc Zones for a MATLAB Notebook

You can partition a MATLAB Notebook into self-contained sections, called *calc zones*. A calc zone is a contiguous block of text, input cells, and output

cells. Section breaks appear before and after the section, defining 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. A notebook can contain any number of calc zones.

Note Using calc zones does not affect the scope of the variables in a notebook. 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 you want to include in the calc zone.
- 2** Select **Notebook > Define Calc Zone**.

Note Select an input cell and its output cell in their entirety to include them in the calc zone.

See “Evaluating a Calc Zone” on page 11-20 for information about evaluating a calc zone.

Converting an Input Cell to Text in a MATLAB Notebook

To convert an input cell (or an autoint 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 the cell converts to text, the cell contents reformat according to the Microsoft Word Normal style. For more information about MATLAB Notebook styles, see “Modifying Styles in the MATLAB Notebook Template” on page 11-23. When you convert an input cell to text, the corresponding output cell also converts to text.

Evaluating MATLAB Commands in a MATLAB Notebook

In this section...

“Evaluating Input Commands” on page 11-17

“Evaluating Cell Groups” on page 11-18

“Evaluating a Range of Input Cells” on page 11-20

“Evaluating a Calc Zone” on page 11-20

“Evaluating an Entire MATLAB Notebook” on page 11-20

“Using a Loop to Evaluate Input Cells Repeatedly” on page 11-21

“Converting Output Cells to Text” on page 11-22

“Deleting Output Cells” on page 11-22

Evaluating Input Commands

After you define a MATLAB command as an input cell, or as an autoinit cell, you can evaluate it in your MATLAB Notebook. Use the following steps to define and evaluate a MATLAB command:

- 1 Type the command into the notebook as text. For example:

```
This is a sample MATLAB Notebook
```

```
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**.

The MATLAB command becomes an input cell. For example:

```
This is a sample MATLAB Notebook
```

```
[a = magic(3)]
```

- 3 Specify the input cell you want to evaluate by selecting it with the mouse or by placing the cursor in it. Then select **Notebook > Evaluate Cell** or press **Ctrl+Enter**.

The input cell is evaluated and the results display in an output cell immediately following the input cell. If there is already an output cell, its contents refresh or update, wherever the output cell is in the notebook. For example:

This is a sample MATLAB Notebook.

```
[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 that enclose matrices in their size and weight. Error messages appear in red. For information about changing these default formats, see “Modifying Styles in the MATLAB Notebook Template” on page 11-23.

Evaluating Cell Groups

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 a MATLAB Notebook” on page 11-13.

When MATLAB evaluates a cell group, the output for all commands in the group appears in a single output cell. By default, the output cell appears immediately after the cell group the first time the cell group is evaluated. If

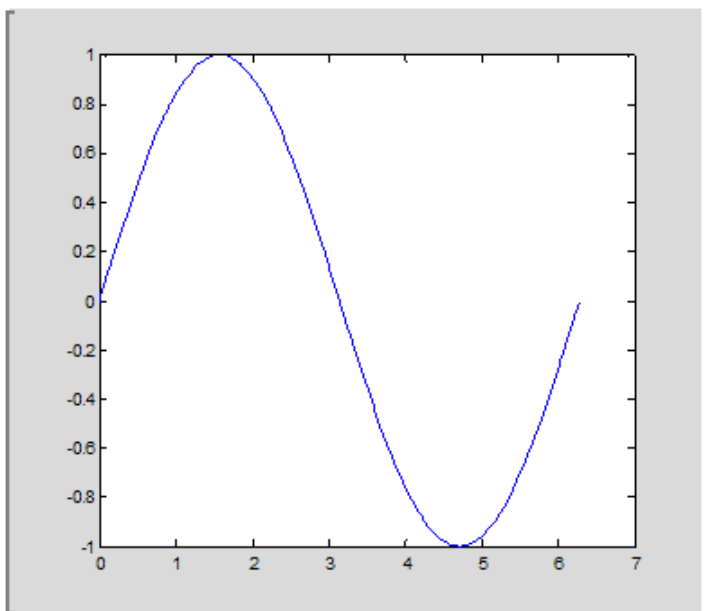
you evaluate a cell group that has an existing output cell, the results appear in that output cell, wherever it is located in the MATLAB Notebook.

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 Notebook with a cell group

```
[t = 0:pi/100:2*pi;  
y = sin(t);  
plot(t,y) ]
```



Evaluating a Range of Input Cells

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**.

Each input cell in the selection evaluates, new output cells appear or existing ones are replaced.

Evaluating a Calc Zone

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 a MATLAB Notebook” on page 11-14.

By default, the output cell appears immediately after the calc zone the first time the calc zone is evaluated. If you evaluate a calc zone with an existing output cell, the results appear in the output cell wherever it is located in the MATLAB Notebook.

Evaluating an Entire MATLAB Notebook

To evaluate an entire MATLAB Notebook, either select **Notebook > Evaluate MATLAB Notebook** or press **Alt+R**.

Evaluation begins at the top of the notebook, regardless of the cursor position and includes each input cell in the file. As it evaluates the file, notebook inserts new output cells or replaces existing output cells.

Controlling Execution of Multiple Commands

When you evaluate an entire MATLAB Notebook, and an error occurs, evaluation continues. If you want to stop evaluation if an error occurs:

- 1 Select **Notebook > Notebook Options**.

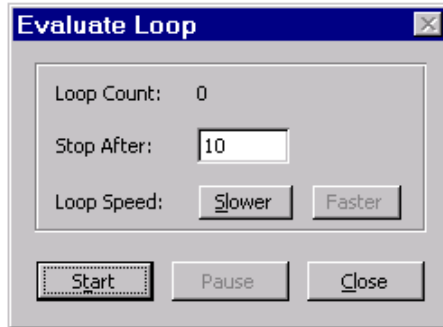
The **Notebook Options** dialog box opens.

- 2 Select the **Stop evaluating on error** check box, and then click **OK**.

Using a Loop to Evaluate Input Cells Repeatedly

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**. The **Evaluate Loop** dialog box displays.



- 3 Enter the number of times you want to evaluate the selected commands in the **Stop After** field, then click **Start**. The button changes to **Stop**. Command evaluation begins, and the number of completed iterations appears 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

You can convert an output cell to text by undefining cells. If the output is numeric or textual, the cell markers disappear and the cell contents convert to text according to the Microsoft Word Normal style. If the output is graphical, the cell markers disappear and the graphic dissociates from its input cell, but contents of the graphic do not change.

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

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, all the output cells in the selected range disappear, but any associated input cells remain intact.

Printing and Formatting a MATLAB Notebook

In this section...

“Printing a MATLAB Notebook” on page 11-23

“Modifying Styles in the MATLAB Notebook Template” on page 11-23

“Choosing Loose or Compact Format” on page 11-24

“Controlling Numeric Output Format” on page 11-25

“Controlling Graphic Output” on page 11-25

Printing a MATLAB Notebook

You can print all or part of a MATLAB Notebook 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 and later releases — Select **Microsoft Office Button > Print**

Word follows these rules when printing MATLAB Notebook cells and graphics:

- Cell markers do not print.
- Input cells, autoint cells, and output cells (including error messages) print 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 MATLAB Notebook Template

You can control the appearance of the text in your MATLAB Notebook by modifying the predefined styles stored in the notebook template, `m-book.dot`. These styles control the appearance of text and cells. By default, notebooks use the Word Normal style for all other text.

For example, if you print a notebook on a color printer, input cells appear dark green, output and autoint cells appear dark blue, and error messages appear red. If you print the notebook on a grayscale printer, these cells appear as

shades of gray. To print these cells using black type, modify the color of the Input, Output, AutoInit, and Error styles in the notebook template.

The following table describes the default styles used by MATLAB Notebook. If you modify styles, you can use the information in the table 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 notebook that use that style and gives you the option to change the template. Be cautious about changing the template. If you choose to apply the changes to the template, you affect all new notebooks that you create using the template. See the Word documentation for more information.

Choosing Loose or Compact Format

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, reevaluate the cells.

Controlling Numeric Output Format

To change how numeric output displays, 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, reevaluate the cells.

Controlling Graphic Output

This section describes how to control several aspects of the graphic output produced by MATLAB commands in a MATLAB Notebook, including

- “Embedding Graphic Output in a MATLAB Notebook” on page 11-25
- “Suppressing Graphic Output for Individual Input Cells” on page 11-26
- “Adjusting Graphic Output” on page 11-26

Embedding Graphic Output in a MATLAB Notebook

By default, graphic output is embedded in a MATLAB Notebook. 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 MATLAB Notebook** check box.
- 3** Click **OK**.

Note Embedded figures do not include Handle Graphics objects generated by the `uicontrol` and `uimenu` functions.

Whether to embed a figure in the MATLAB Notebook is determined by the value of the figure object's `Visible` property. If the value of the property is `off`, the figure embeds in the notebook. If the value of this property is `on`, all graphic output appears in the current figure window.

Suppressing Graphic Output for Individual Input Cells

If an input or autoint 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**.

Graphic output from the cell does not appear, and the string (no graph) appears after the input cell.

To allow graphic output for a cell, repeat the procedure. The (no graph) marker disappears and graphic output from the cell appears.

Note **Toggle Graph Output for Cell** overrides the **Embed figures in MATLAB Notebook** option, if that option is set.

Adjusting Graphic Output

To set the default size of embedded graphics in a MATLAB Notebook, 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 notebook.
- 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, reevaluate the cells.

Change the size of an existing embedded figure by selecting the figure, clicking the left mouse button anywhere in the figure, and then 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.

Notebook Feature Reference

In this section...
“Bring MATLAB to Front” on page 11-28
“Define Autoinit Cell” on page 11-29
“Define Calc Zone” on page 11-29
“Define Input Cell” on page 11-30
“Evaluate Calc Zone” on page 11-30
“Evaluate Cell” on page 11-31
“Evaluate Loop” on page 11-32
“Evaluate MATLAB Notebook” on page 11-32
“Group Cells” on page 11-32
“Hide Cell Markers” on page 11-33
“Notebook Options” on page 11-33
“Purge Selected Output Cells” on page 11-33
“Toggle Graph Output for Cell” on page 11-34
“Undefine Cells” on page 11-34
“Ungroup Cells” on page 11-35

This section provides reference information about each of the MATLAB Notebook features, listed alphabetically. To use these features, select them from the **Notebook** menu in Microsoft Word. In Word 2007 and later releases, 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 a MATLAB Notebook.

Result

If you select this feature while the cursor is in a paragraph of text, the entire paragraph converts to an autoinit cell. If you select this feature while text is selected, the text converts to an autoinit cell. If you select this feature while the cursor is in an input cell, the input cell converts to an autoinit cell.

Format

The autoinit cell format uses 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 a MATLAB Notebook” on page 11-14.

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

A calc zone is defined as a Word document section. Section breaks appear 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” on page 11-20. 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 autoint cell. An input cell contains a MATLAB command.

Result

If you select this feature while the cursor is in a paragraph of text, the entire paragraph converts to an input cell. If you select this feature while text is selected, the text converts to an input cell. If you select this feature while the cursor is in an autoint cell, the autoint cell converts to an input cell.

Format

The text appears enclosed in cell markers and the cell uses 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 a MATLAB Notebook” on page 11-12. For information about evaluating input cells, see “Evaluating MATLAB Commands in a MATLAB Notebook” on page 11-17.

Evaluate Calc Zone

Evaluate Calc Zone sends the input cells in the current calc zone to MATLAB for evaluation. The current calc zone is the Word section that contains the cursor.

Result

As each input cell evaluates, it generates an output cell. When you evaluate an input cell for which there is no output cell, the output cell appears immediately after the input cell that generated it. If you evaluate an input cell for which there is an output cell, the results in the output cell are replaced, wherever it is in the MATLAB Notebook.

See Also

For more information, see “Evaluating a Calc Zone” on page 11-20.

Evaluate Cell

Evaluate Cell sends the current input cell or cell group to MATLAB for evaluation. An input cell contains a MATLAB command. A cell group is a single, multiline input cell that contains more than one MATLAB command. The output or an error message displays in an output cell.

Result

If you evaluate an input cell for which there is no output cell, the output cell appears immediately after the input cell that generated it. If you evaluate an input cell for which there is an output cell, the results in the output cell are replaced, wherever that cell is in the MATLAB Notebook. 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 can 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 in a MATLAB Notebook” on page 11-17. For information about evaluating the entire MATLAB Notebook, see “Evaluating an Entire MATLAB Notebook” on page 11-20.

Evaluate Loop

Evaluate Loop evaluates the selected input cells repeatedly.

For more information, see “Using a Loop to Evaluate Input Cells Repeatedly” on page 11-21.

Evaluate MATLAB Notebook

Evaluate MATLAB Notebook evaluates the entire MATLAB Notebook, sending all input cells to MATLAB for evaluation. Evaluation begins at the top of the MATLAB Notebook regardless of the cursor position.

Result

As each input cell evaluations, it generates an output cell. When you evaluate an input cell for which there is no output cell, the output cell appears immediately after the input cell that generated it. If you evaluate an input cell for which there is an output cell, the results are replaced in the output cell wherever it is in the MATLAB Notebook.

See Also

For more information, see “Evaluating an Entire MATLAB Notebook” on page 11-20.

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, it appears after the cell group. However, if text precedes the first input cell in the group, the text remains before the group.

If you include output cells in the selection, they disappear. If you select all or part of an output cell before selecting this feature, its input cell appears 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 a MATLAB Notebook” on page 11-13. For information about converting a cell group to individual input cells, see “Ungroup Cells” on page 11-35.

Hide Cell Markers

Hide Cell Markers hides cell markers in the MATLAB Notebook.

When you select this feature, it changes to **Show Cell Markers**.

Note Cell markers do not print 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 a MATLAB Notebook” on page 11-23 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” on page 11-22.

Toggle Graph Output for Cell

Toggle Graph Output for Cell suppresses or allows graphic output from an input cell.

If an input or autoint cell generates figure output that you want to suppress, place the cursor in the input cell and choose this feature. The string (no graph) appears 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 disappears. This feature overrides the **Embed figures in MATLAB Notebook** option, if that option is set in the **Notebook Options** dialog box.

See Also

See “Embedding Graphic Output in a MATLAB Notebook” on page 11-25 and “Suppressing Graphic Output for Individual Input Cells” on page 11-26 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, that cell becomes undefined. Cell markers disappear and the cell reformats according to the Normal style.

If you undefine an input cell, its output cell automatically undefines. However, if you undefine an output cell, its input cell does not undefine. If you undefine an output cell containing an embedded graphic, the graphic remains in the MATLAB Notebook but is no longer associated with an input cell.

See Also

For information about the Normal style, see “Modifying Styles in the MATLAB Notebook Template” on page 11-23. For information about deleting output cells, see “Purge Selected Output Cells” on page 11-33.

Ungroup Cells

Ungroup Cells converts the current cell group into a sequence of individual input cells or autoint cells. If the cell group is an input cell, the cell group converts to input cells. If the cell group is an autoint cell, the cell group converts to autoint cells. The output cell for the cell group disappears.

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 a MATLAB Notebook” on page 11-13.

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” on page 12-2
- “Setting Up the Source Control Interface on Microsoft Windows” on page 12-3
- “Checking Files Into and Out of Source Control from the MATLAB Desktop on Microsoft Windows” on page 12-11
- “Additional Source Control Actions on Microsoft Windows” on page 12-14
- “Performing Source Control Actions from the Editor, Simulink, or Stateflow File Menu on Microsoft Windows” on page 12-23
- “Troubleshooting Source Control Problems on Microsoft Windows” on page 12-24
- “Source Control Interface on UNIX Platforms” on page 12-26
- “Specifying the Source Control System on UNIX Platforms” on page 12-27
- “Checking Files Into the Source Control System on UNIX Platforms” on page 12-30
- “Checking Files Out of the Source Control System on UNIX” on page 12-33
- “Undoing the Checkout on UNIX Platforms” on page 12-36

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” on page 12-3

“Specify Source Control System with MATLAB Software” on page 12-5

“Register Source Control Project with MATLAB Software” on page 12-7

“Add Files to Source Control” on page 12-9

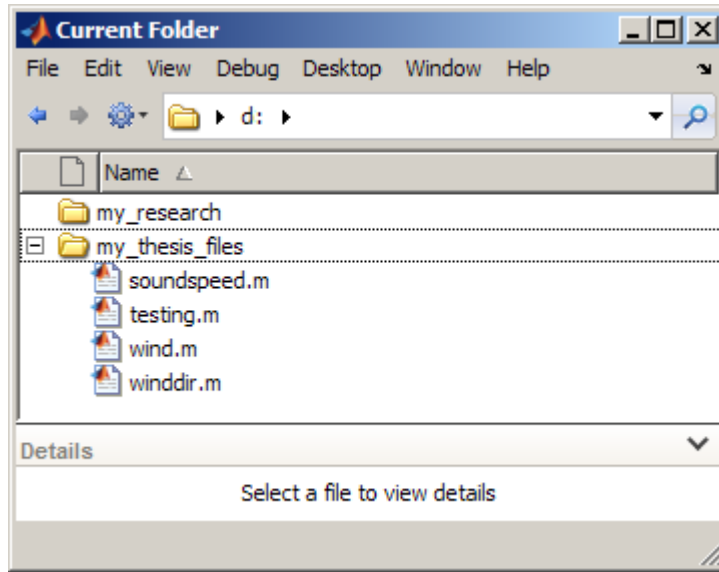
Create Projects in Source Control System

In your source control system, create the projects that your folders and files will be associated with.

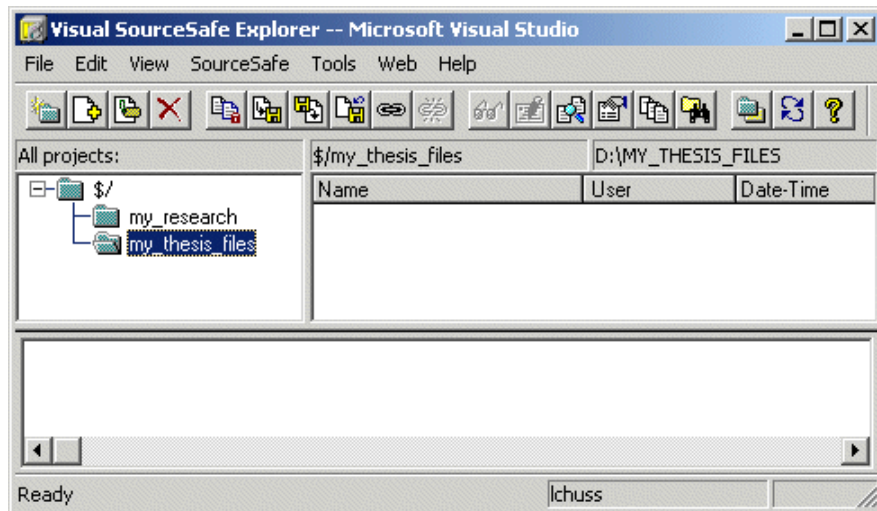
All files in a folder must belong to the same source control project. Be sure the working folder for the project in the source control system specifies the correct path to the folder on disk.

Example of Creating Source Control Project

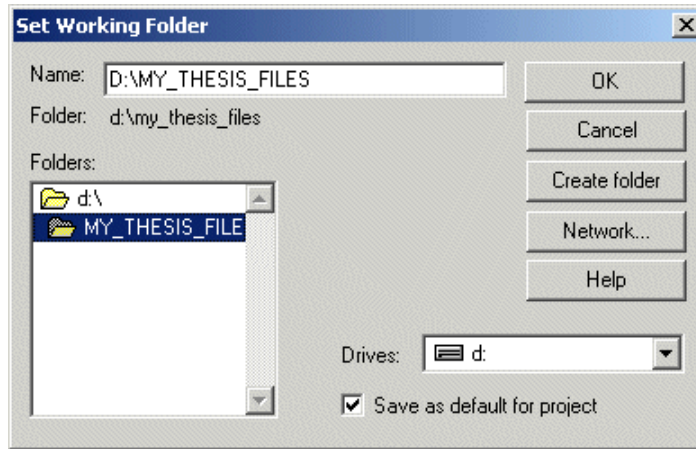
This example uses the project `my_thesis_files` in Microsoft Visual SourceSafe. This illustration of the Current Folder browser shows the path to the folder on disk, `C:\my_thesis_files`.



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.

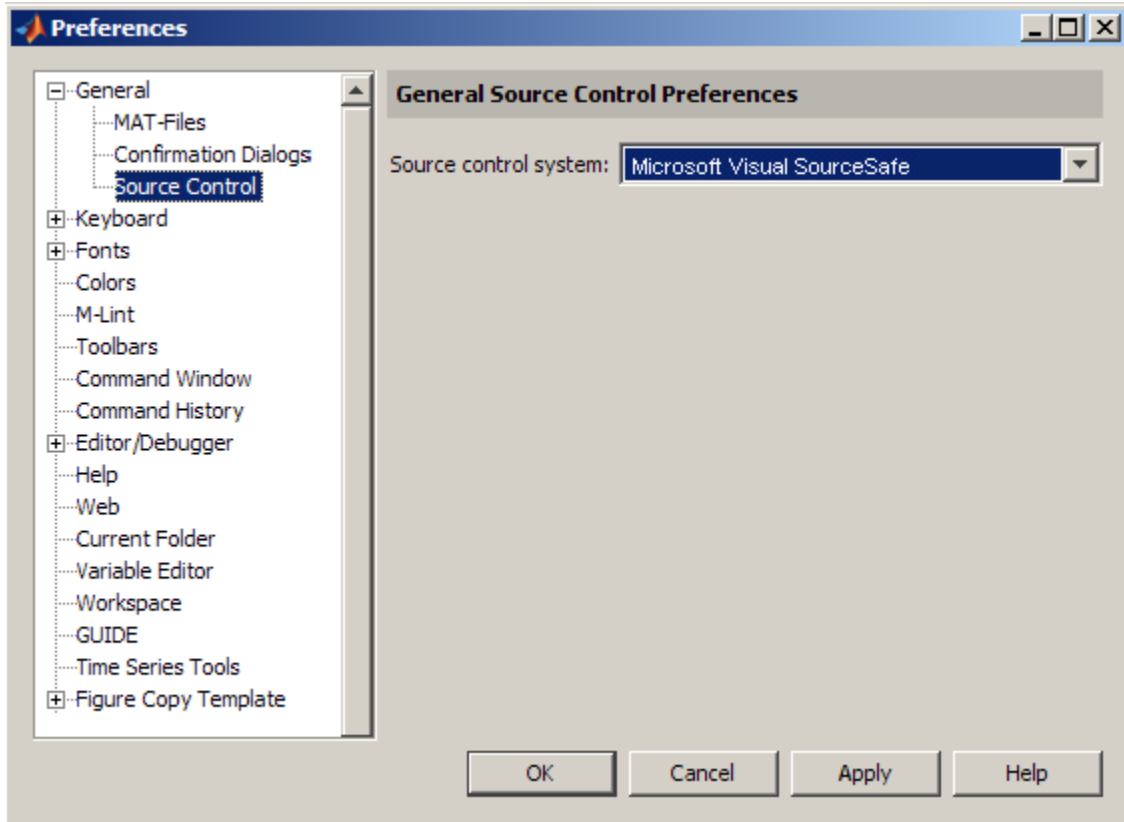


Specify Source Control System with MATLAB Software

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

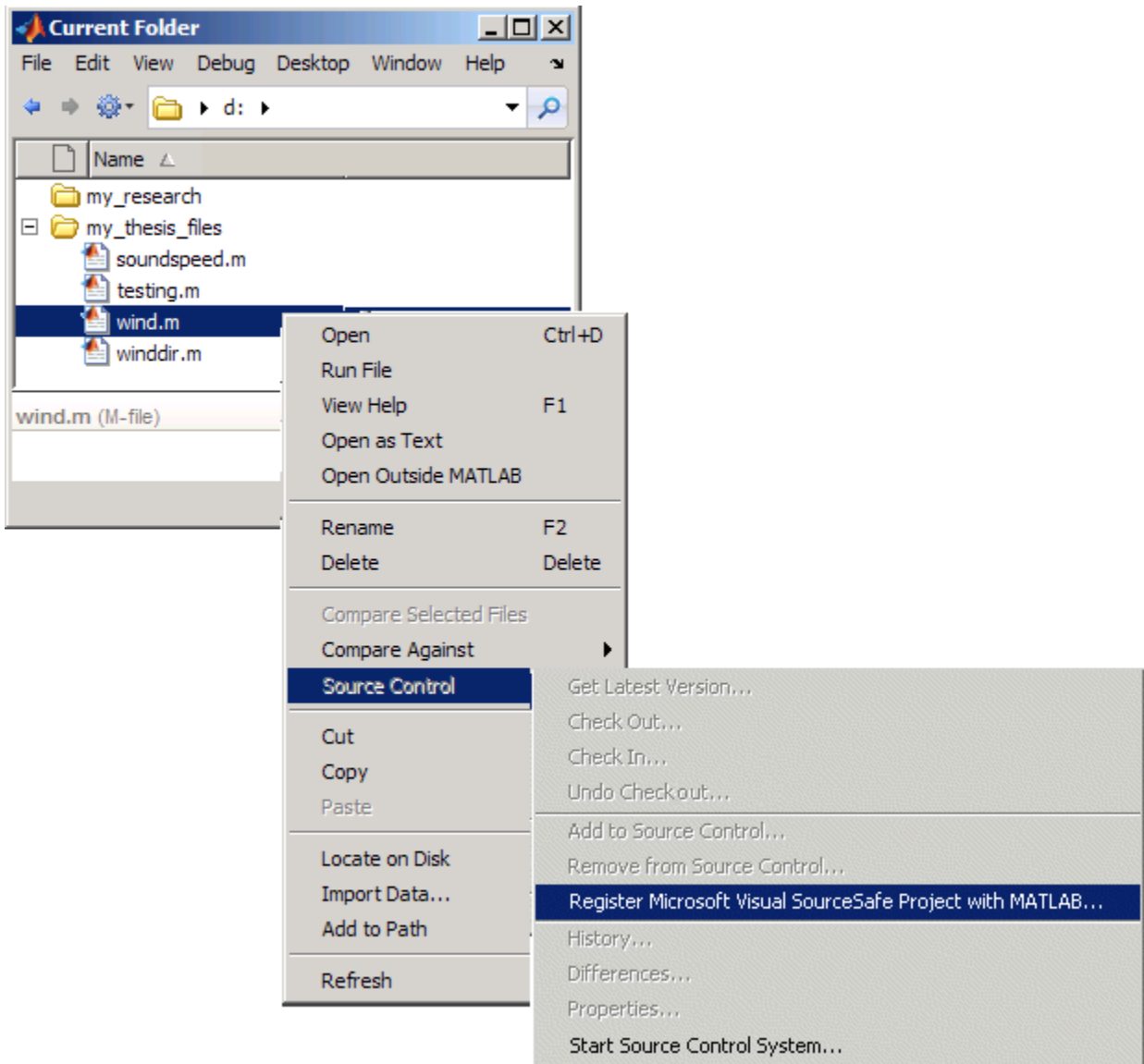
A function alternative to select a source control system is not available, but you can list all available source control systems using `verctrl` with the `all_systems` argument. Use `cmopts` to display the name of the currently selected source control system.

Register Source Control Project with MATLAB Software

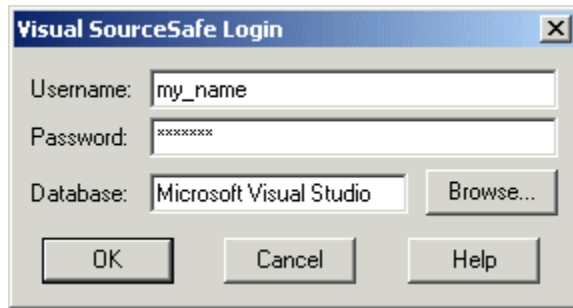
Register a source control system project with a folder in MATLAB, that is, associate a source control system project with a folder and all files in that folder. Do this only one time for any file in the folder, which registers all files in that folder:

- 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.

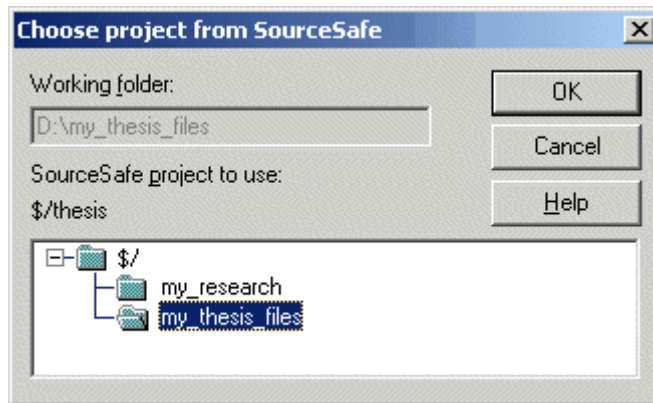
The following example shows Microsoft Visual SourceSafe.



- 3 In the resulting **Name_of_Source_Control_System Login** dialog box, provide the user name 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`.

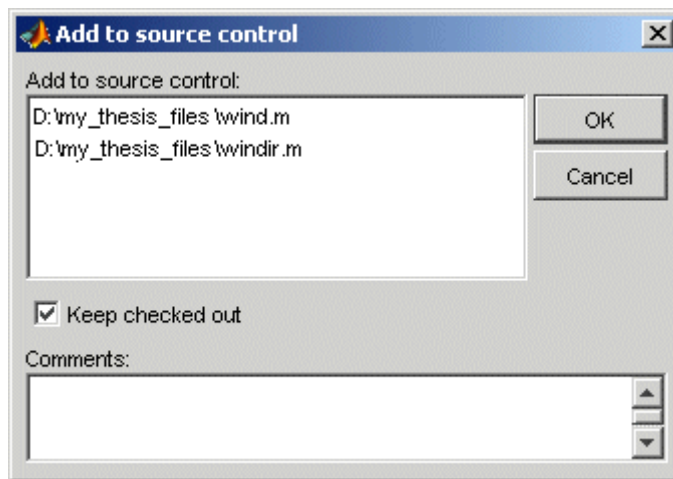


The selected file, its folder, and all files in the folder, are associated with the source control system project you selected. For the example, MATLAB associates all files in `D:\my_thesis_files` with the source control project `my_thesis_files`.

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...

“Check Files Into Source Control” on page 12-11

“Check Files Out of Source Control” on page 12-12

“Undoing the Checkout” on page 12-13

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.

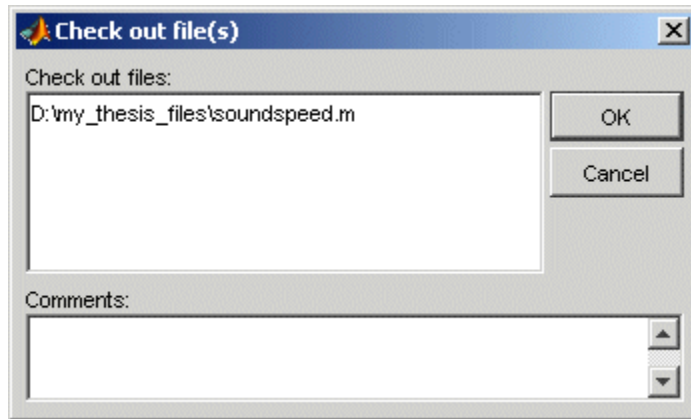
Function Alternative

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 a 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.

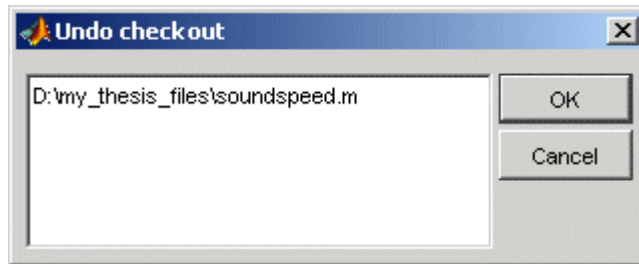
Undoing the Checkout

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” on page 12-14
“Removing Files from the Source Control System” on page 12-15
“Showing File History” on page 12-16
“Comparing the Working Copy of a File to the Latest Version in Source Control” on page 12-18
“Viewing Source Control Properties of a File” on page 12-20
“Starting the Source Control System” on page 12-21

Getting the Latest Version of Files for Viewing or Compiling

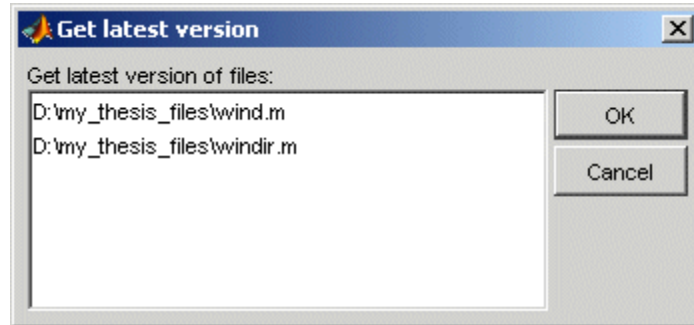
You can get the latest version of a file from the source control system for viewing or running. Getting a file differs from checking it out. When you get a file, it is write protected so you cannot edit it, but when you check out a file, you can edit it.

To get the latest version, follow these steps:

- 1 In the MATLAB Current 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

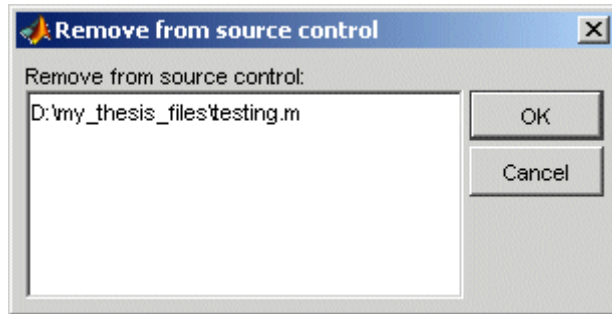
The function alternative is `verctrl` with the `get` argument.

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

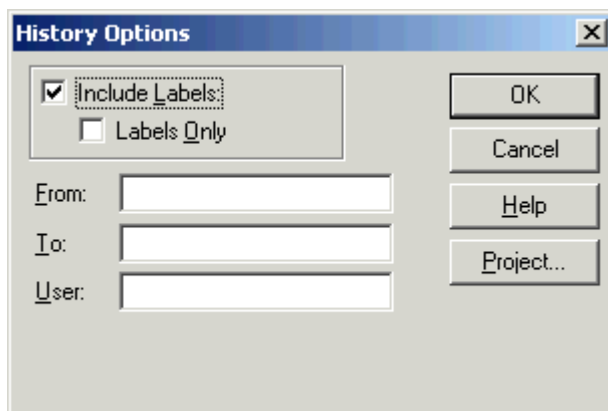
The function alternative is `verctrl` with the `remove` argument.

Showing File History

To show the history of a file in the source control system, follow these steps:

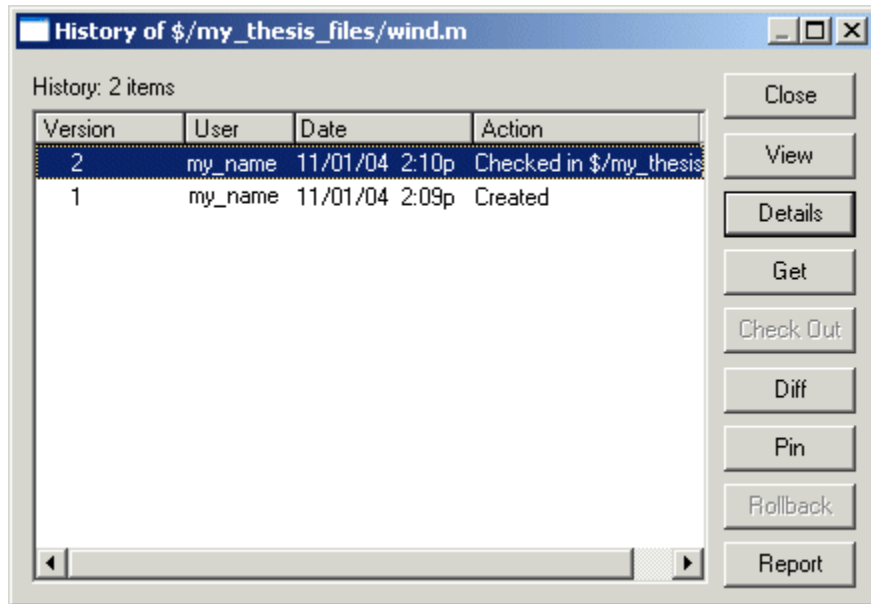
- 1 In the MATLAB Current Folder browser, select the file for which you want to view the history.
- 2 Right-click, and from the context menu, select **Source Control > History**.

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.

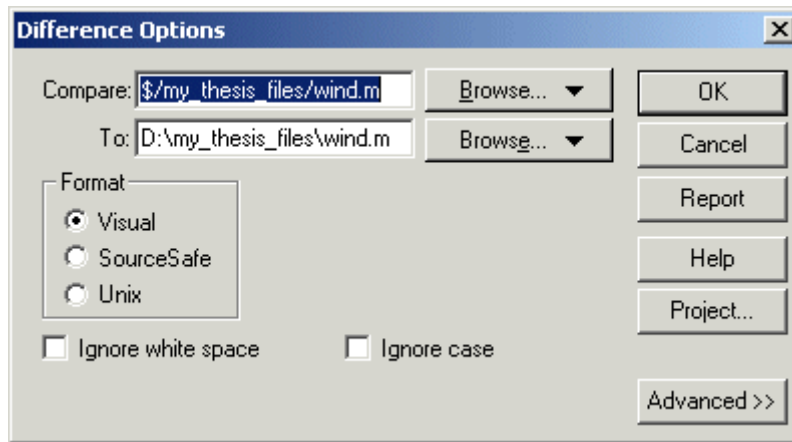
To view the differences, follow these steps:

- 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.

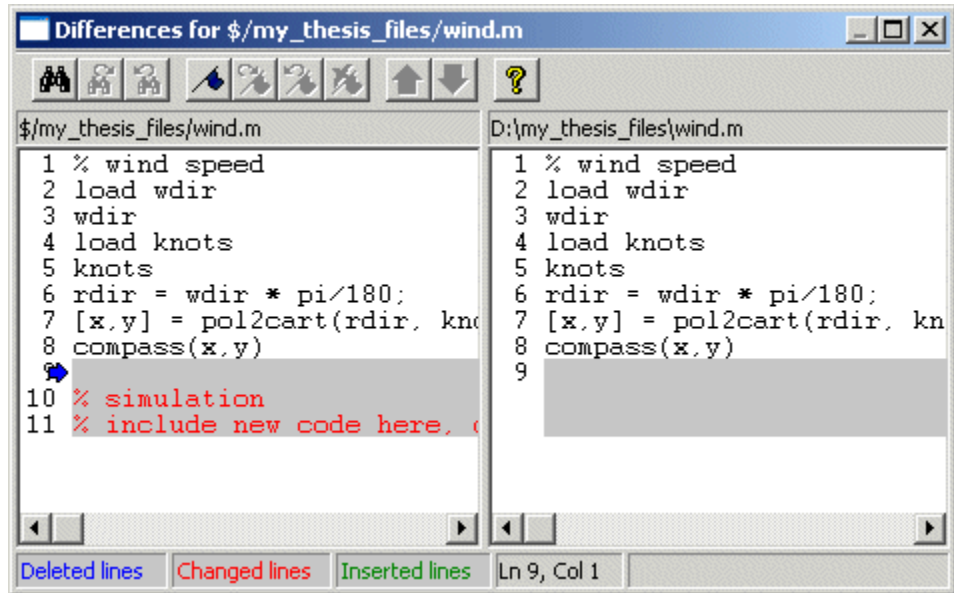
- 2 Right-click, and from the context menu, select **Source Control > Differences**.

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

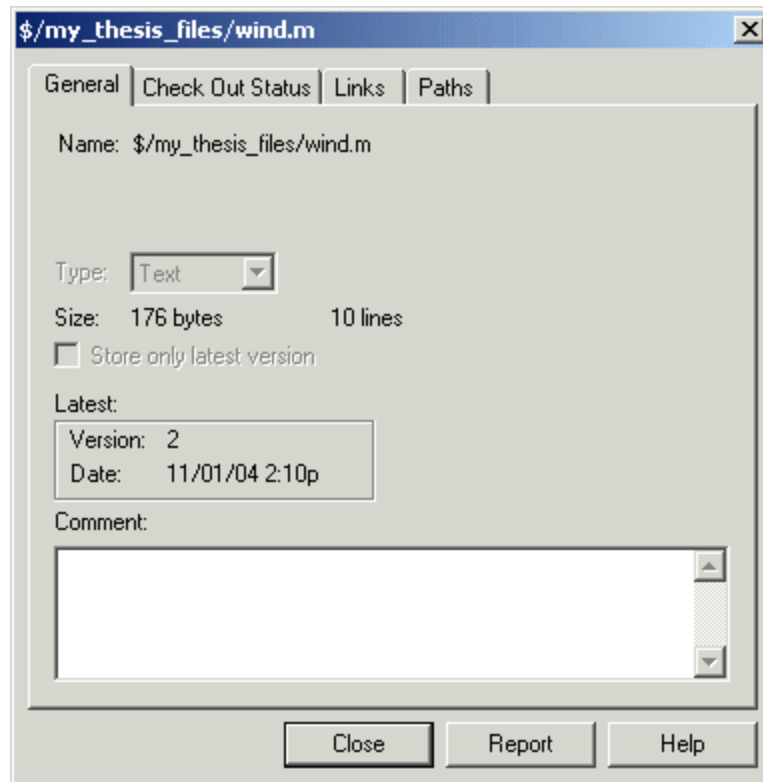
The function alternative is `verctrl` with the `showdiff` or `isdiff` argument.

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

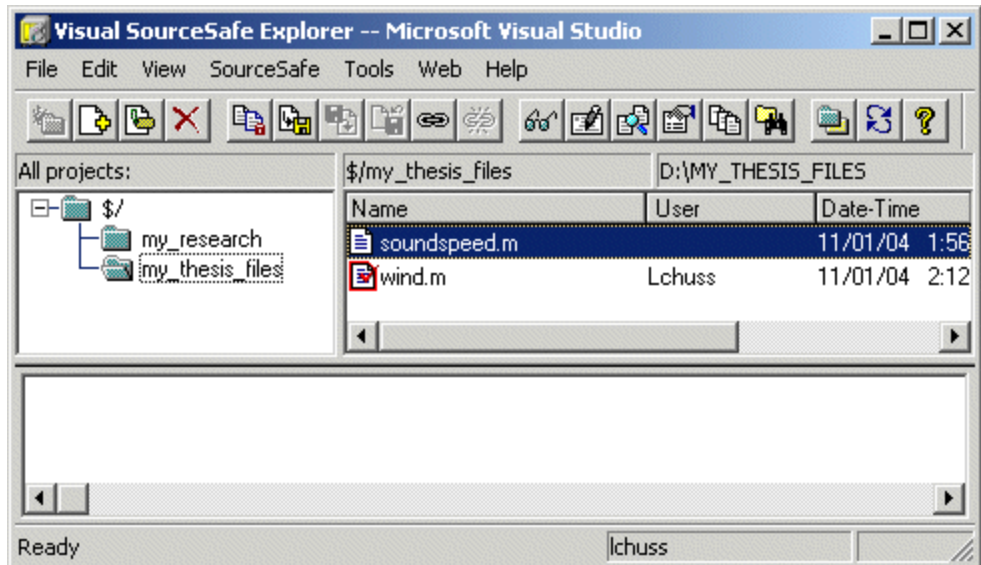
The function alternative is `verctrl` with the `properties` argument.

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 `runsc` 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 the MATLAB Editor 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” on page 12-24

“Restriction Against @ Character” on page 12-25

“Add to Source Control Is the Only Action Available” on page 12-25

“More Solutions for Source Control Problems” on page 12-25

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\ScServerPath
```

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 MATLAB program 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” on page 12-27

“Function Alternative” on page 12-28

“Setting a View and Checking Out a Folder with ClearCase Software on UNIX Platforms” on page 12-29

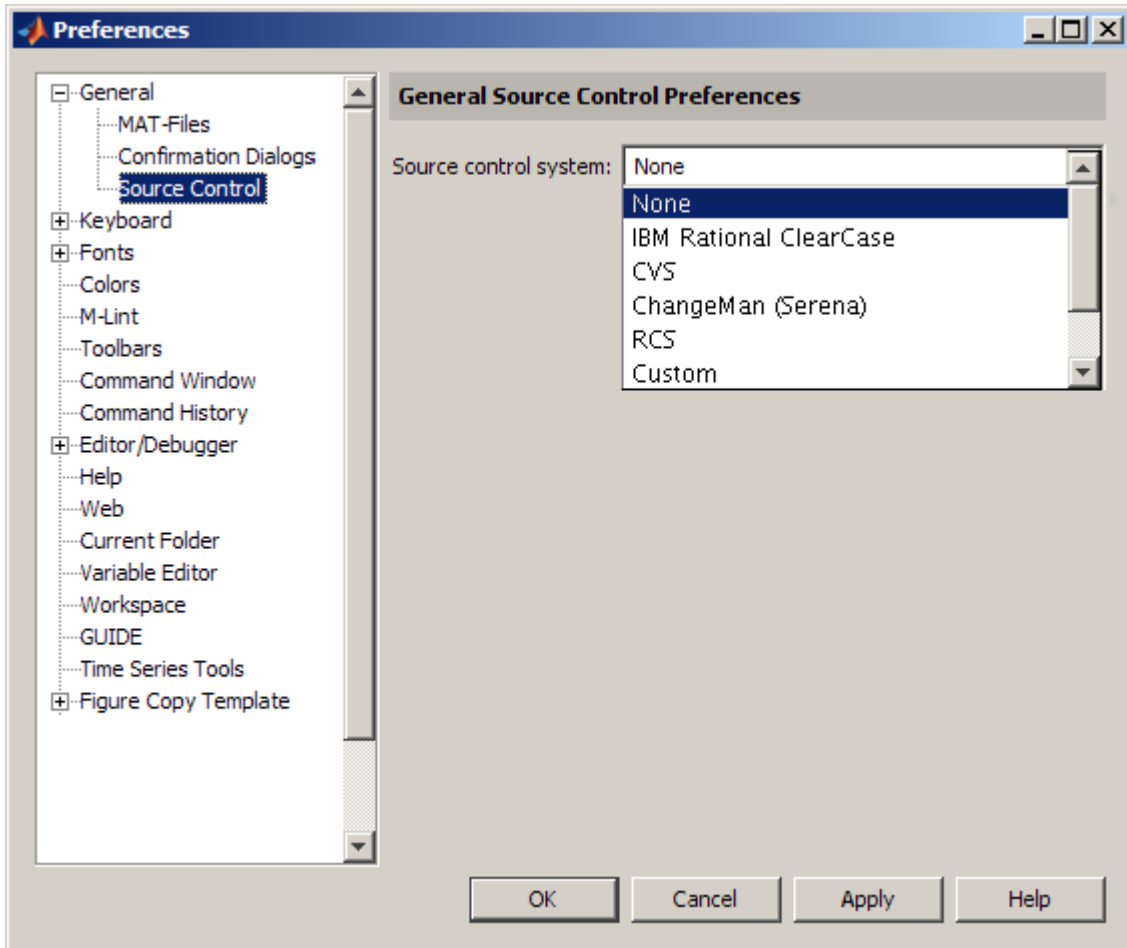
MATLAB Desktop Alternative

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.

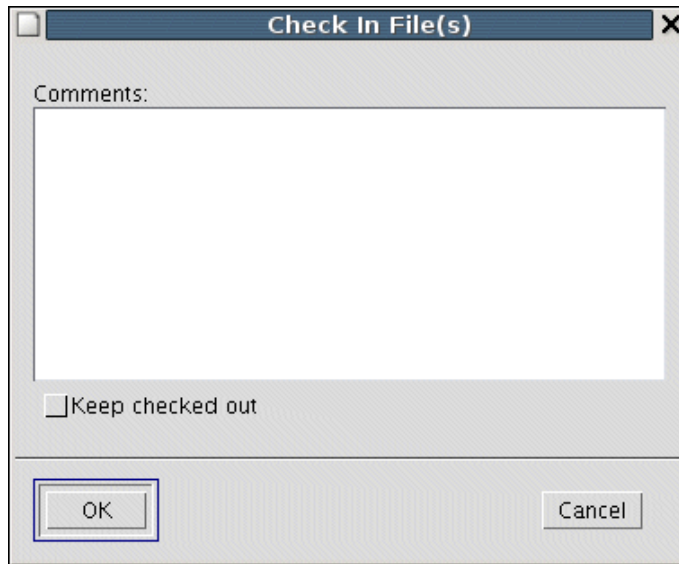
You can now use the MATLAB, Simulink, or Stateflow source control interfaces to ClearCase software.

Checking Files Into the Source Control System on UNIX Platforms

In this section...
“Checking In One or More Files Using the Current Folder Browser” on page 12-30
“Checking In One File Using the Editor, or the Simulink or Stateflow Products” on page 12-31
“Function Alternative” on page 12-32

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\myfiles\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...

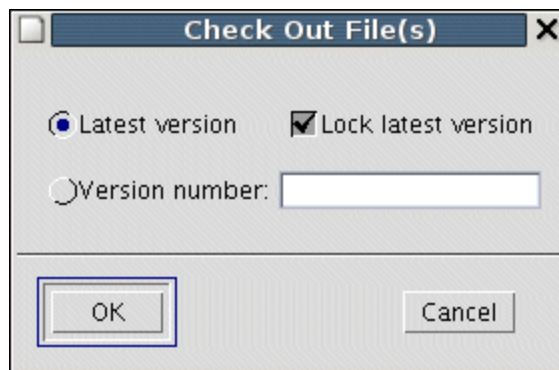
“Checking Out One or More Files Using the Current Folder Browser” on page 12-33

“Checking Out a Single File Using the Editor, or the Simulink or Stateflow Products” on page 12-34

“Function Alternative” on page 12-34

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 a 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 MATLAB program 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\myfiles\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” on page 12-36

“Undoing the Checkout for One or More Files Using the Current Folder Browser” on page 12-36

“Undoing the Checkout for a Single File Using the Editor, or the Simulink or Stateflow Products” on page 12-36

“Function Alternative” on page 12-37

Impact of Undoing a File Checkout

When you undo the checkout for a file, the file remains checked in, and does not have any of the changes you made since you checked it out. To save any changes you have made since checking out a file, select **File > Save As**, and supply a different 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 MATLAB program 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 `file` and include the file extension. For example, to undo the checkout for the files `clock.m` and `calendar.m`, type

```
undocheckout({'\myserver\myfiles\clock.m',...  
'\myserver\myfiles\calendar.m'})
```


Internationalization

- “How the MATLAB Process Uses Locale Settings” on page 13-2
- “Setting the Locale” on page 13-4
- “Troubleshooting I18n Messages and Settings” on page 13-9

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, you 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.

To see what your current settings are, use the instructions in “Setting Locale on Windows Platforms” on page 13-4, “Setting Locale on Linux Platforms” on page 13-6, or “Setting Locale on Macintosh Platforms” on page 13-7. For more information, “Troubleshooting I18n Messages and Settings” on page 13-9.

Consider the following when choosing your locale settings.

- **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 Character Set** — MATLAB supports the character set specified by the user locale setting. However, MATLAB might not properly handle character codes greater than 2 bytes.
- **Script Compatibility** — Non-7-bit ASCII characters in MATLAB scripts created with one locale setting might not be compatible with a different locale setting.

For example, if you create a script with the `ja_JP.UTF-8` locale setting, the script might not be compatible when executed on a platform with the `ja_JP.eucJP` locale setting.

- **Numeric Format Uses C Locale** — MATLAB 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 more information, see “Numbers Display Period for Decimal Point” on page 13-10.

- **Platform-Specific Localized Formats** — MATLAB usually uses platform-neutral localized formats and rules. You can, however, use the operating system short date format to display files, as described in “Customizing the Column Display” on page 6-21.

Windows Platform-Specific Behavior

The user locale and system locale must be the same value on the Microsoft Windows platform. If these values are not the same, you might see garbled text or incorrect characters. For information on controlling these settings, see “Setting Locale on Windows Platforms” on page 13-4.

Macintosh Platform-Specific Behavior

On the Apple Macintosh OS X platform, MATLAB reads the user locale setting and the user UI language setting. For information on controlling these settings, see “Setting Locale on Macintosh Platforms” on page 13-7. MATLAB ignores the LANG environment variable and the Terminal application locale setting.

MATLAB automatically chooses a codeset for each combination of language and territory on the Mac OS X platform. If you customize the locale setting on OS X, MATLAB ignores the customized portion.

Setting the Locale

In this section...
“Setting Locale on Windows Platforms” on page 13-4
“Setting Locale on Linux Platforms” on page 13-6
“Setting Locale on Macintosh Platforms” on page 13-7

Setting Locale on Windows Platforms

MATLAB software uses the *system locale* and *user locale* on Windows platforms:

- “Setting User Locale on Windows 7 Platforms” on page 13-4
- “Setting System Locale on Windows 7 Platforms” on page 13-4
- “Setting User Locale on Windows Vista Platforms” on page 13-5
- “Setting System Locale on Windows Vista Platforms” on page 13-5
- “Setting User Locale on Windows XP Platforms” on page 13-6
- “Setting System Locale on Windows XP Platforms” on page 13-6

Setting User Locale on Windows 7 Platforms

- 1 Select **Start** -> **Control Panel** -> **Clock, Language, and Region** -> **Regional and Language**.
- 2 Open **Formats** tab.
- 3 Select a target locale from the **Format:** drop-down list.

Setting System Locale on Windows 7 Platforms

- 1 Select **Start** -> **Control Panel** -> **Clock, Language, and Region** -> **Regional and Language**.
- 2 Open **Administrative** tab.

- 3** Look in the **Language for non-Unicode programs** section.
- 4** Click **Change system locale...** button.
- 5** Select a target locale from the **Current system locale:** drop-down list.
- 6** 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 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 Platforms

Linux 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.
- 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.

Configuring Fonts to Display Asian Characters

On some Linux systems, to properly display Asian characters in the MATLAB Desktop, you must configure the font with the Java Runtime Environment (JRE™). If you previously configured fonts for your system, you must also make the configuration changes for the JRE distributed with MATLAB.

To configure, make a symbolic link between your font and the MATLAB font fallback directory. For example, to use the Kochi font, at the Linux system prompt type:

```
ln -s /usr/share/fonts/truetype/kochi  
matlabroot/sys/java/jre/glnxa64/jre/lib/fonts/fallback
```

where *matlabroot* is the folder where you installed MATLAB.

Alternatively, edit the fontconfig.properties file. See your Java documentation for information about this file.

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 ->Language & Text

2 Open **Formats** tab

3 Select an item from the **Region** pop-up menu

Setting UI Language

1 Select **System Preferences ->Language & Text**

2 Open **Language** tab

3 Drag an item to the top of the **Languages** list

Troubleshooting I18n Messages and Settings

The term *I18n* is an abbreviation for internationalization, where 18 stands for the number of letters between the i and the n.

In this section...

“Asian Characters Incorrectly Displayed on Linux Systems” on page 13-9

“Characters Incorrectly Displayed on Windows Systems” on page 13-10

“datenum Might Not Return Correct Value” on page 13-10

“Numbers Display Period for Decimal Point” on page 13-10

“MATLAB Displays Messages in English” on page 13-11

“File or Folder Names Incorrectly Displayed” on page 13-11

Asian Characters Incorrectly Displayed on Linux Systems

On some Linux systems, to properly display Asian characters in the MATLAB Desktop, you must configure the font with the Java Runtime Environment (JRE). If you previously configured fonts for your system, you must also make the configuration changes for the JRE distributed with MATLAB.

To configure, make a symbolic link between your font and the MATLAB font fallback directory. For example, to use the Kochi font, at the Linux system prompt type:

```
ln -s /usr/share/fonts/truetype/kochi
matlabroot/sys/java/jre/glnxa64/jre/lib/fonts/fallback
```

where *matlabroot* is the folder where you installed MATLAB.

Alternatively, edit the `fontconfig.properties` file. See your Java documentation for information about this file.

Characters Incorrectly Displayed on Windows Systems

The user locale and system locale must be the same value on the Microsoft Windows platform. If these values are not the same, you might see garbled text or incorrect characters. For information on controlling these settings, see “Setting Locale on Windows Platforms” on page 13-4.

datenum Might Not Return Correct Value

To ensure the correct calculation of functions using date values associated with files and folders, 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 the `date` field of the `dir` command, similar to:

```
n=datenum(f.date);
```

replace it with the `datenum` field:

```
n=f.datenum;
```

Numbers Display Period for Decimal Point

MATLAB uses a period for a decimal point, regardless of the format specified by the user locale. For example, the value of `pi` can be displayed as `3,1416` or `3.1416`, depending on the format used by a locale. MATLAB always displays `3.1416`.

The MATLAB language reserves the use of commas to the cases described in the “Comma — ,” topic of the Programming Fundamentals Symbol Reference.

MATLAB Displays Messages in English

MATLAB displays messages in English, regardless of the UI language setting, except when running in a Japanese Microsoft Windows environment.

File or Folder Names Incorrectly Displayed

On Windows and Linux platforms, characters used in file or folder names must be in the supported character set. See **Supported Character Set** in “How the MATLAB Process Uses Locale Settings” on page 13-2.

On Macintosh platforms, for files and folders used by MATLAB, characters in the file or folder name must be in the 7-bit ASCII character set.

Symbols and Numerics

- , after functions 3-39
- ; after functions 3-39
- % comment
 - creating 8-27
- % comment symbol 8-27
- ! function 3-15
 - argument length restrictions 3-15
- %% 8-61
- {% block comment symbol 8-28
- >> prompt in Command Window 3-3

A

- absolute path name 6-8
 - copying 6-9
- accelerators
 - Command Window 2-37
- accelerators, keyboard 2-37
- Access Bridge 2-84
- accessibility 2-81
 - documentation 2-82
 - installation 2-84
 - troubleshooting 2-87
- activate license 2-76
- antialiasing
 - desktop fonts 2-104
- AppleScript
 - running from MATLAB 3-16
- archive files
 - adding files to 6-39
 - creating 6-37
 - extracting files from 6-38
- arrays
 - editing 5-22
 - workspace 5-2
- assistive technology 2-81
- asv 8-7
- autoinit cells
 - converting input cells to 11-29

- converting to input cells 11-30
 - defining 11-14
- AutoInit style
 - definition of 11-24
- automatic completion of statement
 - Command Window 3-43
 - Editor 8-85
- automatic fix
 - Code Analyzer 8-95
- autosave files 8-7

B

- back and forward navigation 8-38
- backup files
 - MATLAB Editor autosave and 8-7
- bang (!) function 3-15
- base workspace 5-2
- batch mode for starting MATLAB 1-21
- beep
 - preferences 3-65
- binary files
 - comparing 6-61
- block comments 8-28
 - extending 8-28
- Blocks of LaTeX math symbols
 - in published MATLAB code 10-41
- blue breakpoint icon 8-143
- bold text
 - in published MATLAB code 10-44
 - within cell 10-44
- bookmarks
 - in files in Editor 8-34
 - in Help browser 4-9
- Boolean searching in Help browser 4-5
- breakpoints
 - anonymous functions 8-143
 - blue icon 8-143
 - clearing (removing) 8-135
 - clearing, automatically 8-136

- conditional 8-141
 - disabling and enabling 8-134
 - multiple per line 8-143
 - running file 8-123
 - setting 8-119
 - types 8-119
- Bring MATLAB to Front 11-28
- browser
- for Web 2-71
- bugs, reporting to MathWorks 4-11
- ## C
- caching
- files 8-6
 - search path 6-6
- calc zones
- defining 11-14
 - ensuring workspace consistency in MATLAB Notebooks 11-8
 - evaluating 11-20
 - output from 11-20
- callbacks
- in shortcuts 3-23
- case
- changing lower to upper in Editor 8-16
 - changing upper to lower in Editor 8-16
- case sensitivity
- of file names 6-9
 - of path names 6-9
- cell arrays
- editing 5-25
- cell breaks 8-61 10-18
- cell dividers. *See* cell breaks
- cell groups
- converting to input cells 11-35
 - creating 11-13
 - definition of 11-13
 - evaluating 11-18
 - output from 11-18
- cell highlighting
- troubleshooting 8-65
- cell markers
- defined 11-12
 - hiding 11-33
 - printing 11-23
- cell mode 8-58
- cell scripts 8-58
- cell titles 8-62
- cells
- files and 8-58
- cells in files 8-58
- removing 8-67
- changing
- search path 6-73
- character set
- preference for MAT-files 2-97
- checkcode 8-84
- checkin
- on UNIX platforms 12-32
- checking in files
- on UNIX platforms 12-30
- checking out files
- on UNIX platforms 12-33
 - on Windows platforms 12-12
 - undoing on UNIX platforms 12-36
 - undoing on Windows platforms 12-13
- checkout
- on UNIX platforms 12-34
- class help
- adjusting wrapping for 8-29
- clear 5-8
- ClearCase source control system
- configuring on UNIX platforms 12-29
- clearing
- variables 5-8
- clicking on multiple items 2-66
- clipboard 2-63
- closing
- MATLAB 1-7

- code
 - automatically analyzing for warnings and errors 8-91
 - checking 9-22
- code analysis 8-84
- Code analysis
 - Editor access 8-84
- Code Analyzer messages
 - suppressing 8-100
- Code Analyzer preferences 8-161
 - setting 8-161
- Code Analyzer Report 9-2 9-22
 - checking MATLAB code 9-22
- code cells 8-59
 - evaluating 8-77
 - files and 8-58
 - publishing and 10-2
- code cells in files
 - beep 8-78
 - defining 8-61
 - evaluating 8-78
 - evaluating code in 8-78
 - modifying values in 8-79
 - nested 8-68
 - toolbar 8-60
- code examples 8-151
- code folding
 - viewing code in Tooltip 8-20
- code folding in files 8-20
- code folding preferences in files 8-158
- code iteration 8-58
- code resources 8-151
- code samples
 - sample code 8-151
- collapsing
 - code in files 8-20 8-158
- Collatz problem 8-117
- colors
 - general preferences 2-9
 - Help browser 4-7
 - in files 8-84
 - indicators for syntax 3-41
 - printing MATLAB Notebook 11-23
- column numbers 8-9
- columns
 - customizing in Current Folder browser 6-21
- command flags 1-18
- Command History
 - about 3-4
 - changing date format in 6-22
 - deleting entries in window 3-34
 - file 3-4
 - find entry by letter 3-61
 - preferences 3-67
 - running functions from window 3-30
- command line
 - defined 3-3
- command name completion
 - Command Window 3-43
 - Editor 8-85
- command switches 1-18
- Command Window
 - bringing to front in Notebook 11-28
 - getting started message bar 3-63
 - matrix output display 3-38
 - paging of output in 3-39
 - preferences 3-63
 - scroll buffer 3-63
 - width 3-63
- commands
 - executing a group of 3-23
 - to operating system 3-15
- comments
 - adding/removing in C/C++ files 8-27
 - adding/removing in Java files 8-27
 - adding/removing with any text editor 8-27
 - adding/removing with Editor 8-27
 - adjusting wrapping for class help 8-29
 - block 8-28
 - color indicators 2-9

- creating in MATLAB code files 8-26
 - marking up within code cell 10-18
 - multiline statements 8-28
 - using ... (ellipsis) 8-28
 - within a line 8-28
 - wrapping in files 8-29
- comparing
- directories 6-47
 - files 6-47
- comparing working copy to source control version
- on Windows platforms 12-18
- Comparison Tool
- features of 6-61
- completing statements automatically
- Command Window 3-43
 - Editor 8-85
- compression
- MAT-files and Fig-Files 2-97
- conditional breakpoints 8-141
- configuration management
- See* source control system interface 12-1
- configuration, desktop 2-13
- configurations
- reassociating 8-56
 - renaming 8-56
 - See also* publish configurations 8-46
 - See also* run configurations 8-46
- configuring Notebook 11-10
- confirmation dialog boxes
- preferences 2-98
- Contents Report 9-10
- conversion
- Word document to MATLAB Notebook 11-6
- copying
- files and folders 6-43
- Coverage Report 9-19
- crash 1-9
- creating
- files and folders
 - using functions 6-39
- cropping graphics
- in MATLAB Notebooks 11-27
- current folder
- at startup for MATLAB 1-12
 - changing 6-4
 - viewing 6-4
- Current Folder browser 6-14
- asterisks and 6-21
 - changing date format in 6-22
 - columns 6-21
 - details panel 6-23
 - preferences 6-16
 - refresh display 6-17
 - running scripts from 6-45
 - viewing image thumbnails in 6-23

D

- data consistency
- calc zones in MATLAB Notebooks 11-8
 - evaluating MATLAB Notebooks 11-8
 - in MATLAB Notebook 11-8
- data tips
- example 8-128
- date format
- changing in Command History window 6-22
 - changing in Current Folder browser 6-22
- dbclean 8-135
- dbstop
- example 8-123
- deactivate license 2-76
- Debugger 8-1
- debugging
- ending 8-133
 - example 8-117
 - features 8-116
 - Notebook 11-9
 - prompt 8-124
 - stepping 8-125
 - with unsaved changes 8-140

- defaults
 - preferences for MATLAB 2-93
 - setting in startup file for MATLAB 1-19
 - Define Autoinit Cell 11-29
 - Define Calc Zone 11-29
 - Define Input Cell 11-30
 - deleting
 - files and folders
 - using Current Folder browser 6-41
 - using functions 6-42
 - variables 5-8
 - delimiter
 - matching in Editor 3-65
 - preferences for matching 3-65
 - Dependency Report 9-14
 - description for file
 - viewing in Current Folder browser 6-22
 - desktop
 - configuration 2-13
 - description 2-2
 - docking 2-19
 - font preferences for 2-103
 - grouping tools 2-19
 - layout
 - saving 2-33
 - maximizing tools 2-21
 - minimizing tools 2-22
 - predefined layouts 2-34
 - saving layout 2-33
 - windows
 - docking 2-14
 - grouping together 2-19
 - sizing 2-14
 - sizing using keyboard 2-14 2-18
 - undocking 2-18 2-33
 - development environment for MATLAB 2-2
 - diagnostics
 - startup
 - Macintosh 1-7
 - difference reporting for files 6-47
 - directories
 - comparing 6-47 6-49
 - disabling
 - breakpoints 8-134
 - displaying
 - output 3-39
 - displaying source control properties of a
 - file 12-20
 - dividers for code cells. *See* cell breaks
 - do not show again
 - preferences 2-98
 - docking tools in desktop 2-19
 - docking windows in desktop 2-14
 - document bar
 - document name 2-27
 - width 2-27
 - document titles
 - in published MATLAB code 10-18
 - documentation
 - accessibility 2-82
 - dynamic hyperlinks
 - inserting in published code 10-50
 - inserting to run MATLAB code 10-51
- ## E
- editing
 - MATLAB files 8-1
 - Editor 8-1
 - changing casing in 8-16
 - closing 8-8
 - closing files 8-7
 - example 8-117
 - go to
 - bookmark 8-34
 - function 8-34
 - line number 8-34
 - highlighting current line in 8-16
 - horizontal lines 8-62
 - modifying values 8-77

- navigating 8-34
- navigating back and forward 8-38
- opening files in 8-2
- publishing files 10-66
- redoing an activity 8-16
- rule displayed 8-10
- running files 8-46
- running with unsaved changes 8-140
- status bar
 - function 8-11
- undoing an activity 8-16

Editor/Debugger

- publishing images preferences 10-76
- publishing preferences 10-76

EDU>> prompt in Command Window 3-3

Embed figures in MATLAB Notebook 11-26

embedding graphics

- in MATLAB Notebook 11-25

encoding

- preference when saving 2-97

ending MATLAB 1-7

environment settings at startup 1-19

environment variables 3-16

error breakpoints

- stop for errors 8-145

error logs 1-9

error message identifiers 8-147

error messages

- in Command Window 3-7

error style

- definition 11-24

errors

- color indicators 2-9
- source control 12-24

errors and warnings

- analyzing code for 8-91

Evaluate Calc Zone 11-30

Evaluate Cell 11-31

Evaluate Loop 11-32

Evaluate Loop dialog box 11-21

Evaluate MATLAB Notebook 11-32

evaluating

- MATLAB Notebooks, ensuring data consistency 11-8
- selection in Command History window 3-30
- selection in Command Window 3-18

evaluating sections of a file 8-78

exact phrase

- Help browser search 4-5

exe 3-15

executables

- running from MATLAB 3-15

executing

- group of statements 3-23

execution

- stopping 3-14

exiting MATLAB 1-7

expanding

- code in files 8-20 8-158

F

f button 8-34

F Inc Search field 8-45

fatal error 1-9

favorites in Help browser 4-9

FIG files

- opening in GUIDE 6-44

Fig-files

- compatibility 2-97
- save options 2-97

file management system

- See* source control system interface 12-1

file name

- case sensitivity 6-9

files

- appearance of 8-9
- backing up 8-4 8-7
- cleanup before publishing 10-55
- closing 8-7

- code cells and 8-58
- colors in 8-84
- comparing 6-47
- creating
 - from Command History window 3-30
- creating new 8-2
- finding by name 6-29
- formatting for publishing 10-10
- formatting MATLAB comments in 8-29
- log 1-20
- marking up code for publishing 10-62
- marking up for publishing
 - section titles 10-22
 - table of contents 10-21
- naming
 - avoiding conflicts 6-67
- opening 8-2
- opening as text files 6-44
- opening outside MATLAB 6-44
- performance of 9-27
- profiling 9-27
- publishing 10-66
 - before and after formatting 10-4
 - bold text 10-44
 - graphics 10-32
 - HTML markup tags 10-35
 - hyperlinks 10-48
 - inline LaTeX math symbols 10-40
 - italic text 10-44
 - LaTeX markup 10-37
 - LaTeX math symbols as blocks 10-41
 - lists 10-29
 - monospaced text 10-44
 - preformatted text 10-25
 - sample code with syntax
 - highlighting 10-27
 - trademark symbols 10-47
- publishing process 10-3
- recommendations on saving 8-6
- run configurations for 8-46
- running sections of 8-58
- saving 8-4
- saving automatically in Editor 8-7
- search path 6-64
- searching contents of 6-29
- snapshot of output in published MATLAB
 - code 10-43
- summary of markup for publishing 10-58
- syntax highlighting in 8-84
- Files
 - opening file or variable from 8-39
- Find Files dialog box 6-29
- finding
 - files and folders
 - by name 6-28
 - files by name and content 6-29
 - text in Command History window 3-61
 - text in Command Window 3-53
- finish.m file running when quitting 1-9
- firewall
 - settings to work through 2-73 2-108
- fix me reports 9-3
- flags
 - for startup 1-18
- folders
 - comparing 6-49
 - creating 6-35
 - MATLAB
 - caching 8-6
- font
 - antialiasing in desktop 2-104
 - Help browser 4-7
 - preferences in MATLAB 2-103
 - smoothing in desktop 2-104
- format
 - controlling numeric format in MATLAB
 - Notebook 11-25
 - in MATLAB Notebook 11-25
 - preferences 3-63
- FTP

- transferring files via link 3-19
- full path name 6-8
 - copying 6-9
- function name
 - automatic completion
 - Command Window 3-43
 - Editor 8-85
- function workspace 5-2
- functions
 - color indicators 2-9
 - executing a group of 3-23
 - naming
 - avoiding conflicts 6-67

G

- get latest version of file on Windows
 - platforms 12-14
- getting files 12-33
- graphical debugger 8-1
- graphics
 - controlling output in MATLAB
 - Notebook 11-26
 - embedding in MATLAB Notebook 11-25
 - in MATLAB Notebooks 11-25
 - in published MATLAB code 10-32
 - within cell 10-29
- gray breakpoint icons 8-122
- gray horizontal lines in Editor 8-62
- green indicator in Editor 8-84
- Group Cells 11-32
- grouping
 - tools in desktop 2-19

H

- HDF
 - preference when saving 2-97
- headings
 - within code cell 10-18

- Help browser
 - color preferences 4-7
 - font preferences 4-7
 - viewing page location 4-9
- Help Report 9-7
- hidden files
 - viewing 6-17
- Hide Cell Markers 11-33
- history
 - automatic log file 1-20
 - source control on Windows platforms 12-16
- history file 3-4
- history of statements 3-4
- history.m file 3-4
- horizontal lines in Editor 8-62
- hot keys 2-37
 - desktop 2-37
 - Variable Editor 5-32
- HTML markup tags
 - in published MATLAB code 10-35
- HTML viewer in MATLAB 2-71
- hyperlinks
 - Command Window 3-18
 - in published MATLAB code 10-48
 - inserting in published code 10-50
 - running functions by 3-19

I

- image files
 - previewing in Current Folder Browser 6-23
- images
 - resizing in published MATLAB code 10-89
- import
 - files for use with MATLAB 6-70
- in files 8-59
- include
 - files with MATLAB 6-70
- incremental searching
 - in Editor 8-45

- indented text
 - within cell 10-29
- indenting
 - in Command Window 3-41
 - in Editor 8-17
- info.xml
 - Start button 2-67
- initiation (init) file for MATLAB 1-19
- inline LaTeX math symbols
 - in published MATLAB code 10-40
- inline links
 - within cell 10-48
- input cells
 - controlling evaluation 11-21
 - controlling graphic output 11-26
 - converting autoinit cell to 11-30
 - converting text to 11-30
 - converting to autoinit cell 11-29
 - converting to cell groups 11-35
 - converting to text 11-15
 - defining in MATLAB Notebooks 11-12
 - evaluating 11-17
 - evaluating cell groups 11-18
 - evaluating in loop 11-21
 - maintaining consistency 11-8
 - timing out during evaluation 11-31
 - use of Word Normal style 11-16
- Input style
 - definition of 11-24
- Insert key
 - Editor 8-15
- insert mode
 - Editor 8-15
- Internet
 - proxy server settings 2-73 2-108
- invalid breakpoints 8-122
- italic text
 - in published MATLAB code 10-44
 - within cell 10-44
- iterative programming 8-58

J

- Java Heap
 - preferences 2-100
- Java VM
 - starting without 1-21
- JAWS 2-83

K

- K>>
 - prompt in Command Window 3-3
- K>> prompt in Command Window
 - debugging mode 8-124
- keyboard shortcuts
 - Command Window 2-37
 - Variable Editor 5-32
- keywords
 - color indicators 2-9
 - matching in Editor 3-65

L

- LaTeX display math
 - in published MATLAB code 10-41
- LaTeX markup
 - in MATLAB code 10-37
- layout for desktop
 - saving 2-33
- license information 4-16
- license management 2-76
- line
 - horizontal
 - in Editor 8-62
 - vertical
 - in Editor 8-10
- line numbers 8-9
 - going to 8-34
- line wrapping 3-63
- links
 - Command Window 3-18

- in published MATLAB code 10-48
- lists
 - in published MATLAB code 10-29
 - within cell 10-29
- load 5-7
- locking files on checkout 12-33
- log
 - automatic 1-20
 - file 1-20
 - statements 3-4
- logfile startup option 1-20
- login
 - remote on Macintosh 1-7
- looping
 - to evaluate input cells 11-21

M

- Macintosh
 - startup
 - remote login 1-7
- MAT-files
 - comparing 6-58
 - compatibility 2-97
 - compression options 2-97
 - creating 5-6
 - defined 5-6
 - loading 5-7
 - preferences 2-97
 - updating using Current Folder Browser 6-36
 - viewing variables without loading 6-23
- matched delimiters
 - preferences 3-65
- matching parentheses
 - in Editor 3-65
- MATLAB
 - commands, executing in a Word
 - document 11-17
 - quitting 1-7
 - confirmation 1-8
 - search path 6-70
- MATLAB code file comments
 - purpose of 8-26
- MATLAB code files
 - file association (Windows) 1-3
 - running
 - at startup 1-21
- MATLAB files
 - editing 8-1
- matlab folder 1-13
- MATLAB functions
 - running by hyperlink 3-19
- MATLAB installations
 - search path with 6-78
- MATLAB Notebooks
 - creating 11-2
 - data consistency 11-8
 - data integrity 11-8
 - entering text and commands 11-7
 - evaluating all input cells 11-20
 - modifying style template 11-23
 - opening 11-5
 - printing 11-23
 - sizing graphic output 11-26
 - styles 11-23
- matlab:
 - running functions with 3-19
- matlab.mat 5-7
- matlabrc.m, startup file 1-19
- matrices
 - editing 5-22
- maximizing
 - tools in desktop 2-21
- measuring performance of your code 9-27
- message identifiers 8-147
- messages
 - suppressing 8-100
 - suppressing indicators 8-100
- Microsoft Word

- converting document to MATLAB Notebook 11-6
- minimize
 - Windows startup option 1-20
- minimizing
 - tools in desktop 2-22
- model files
 - description in Current Folder browser 6-22
- monospaced text
 - in published MATLAB code 10-44
 - within cell 10-44
- more 3-39
- moving
 - files and folders 6-43
- multidimensional arrays
 - editing 5-25
- multiple item selection 2-66
- multithreading
 - turning off 1-21

N

- name clashes 6-67
- naming
 - functions and variables
 - avoiding conflicts 6-67
- navigating
 - files 8-34
- nested
 - code cells in files 8-68
- nested comments 8-28
- nojvm startup option 1-21
- Normal style (Microsoft Word)
 - default style in MATLAB Notebook 11-23
 - defaults 11-24
 - used in undefined input cells 11-16
- notebook
 - function 11-2
 - overview 11-2

- platforms supported 11-1
- Notebook
 - configuring 11-10
 - debugging 11-9
 - options 11-33
- Notebook menu
 - Word menu bar 11-2
- numbering lines 8-9
- numeric format
 - controlling in MATLAB Notebook 11-25
 - preferences 3-63

O

- objects
 - editing 5-25
- operating system commands 3-15
- operators
 - searching for 4-5
- optimizing code performance 9-27
- options
 - shutdown 1-9
 - startup 1-18
- orange underline in file 8-94
- output
 - display
 - hidden 3-39
 - hiding 3-39
 - paging 3-39
 - spaces per tab 3-64
 - spacing of 3-63
 - suppressing 3-39
- output cells
 - converting to text 11-22
 - purging 11-22
- Output style
 - definition 11-24
- overwrite mode
 - Editor 8-15

P

- paging in the Command Window 3-39
- parentheses
 - matching 3-65
- parentheses matching
 - preferences 3-65
- partial
 - path name 6-12
- partial word
 - Help browser search 4-5
- path. *See* search path
- PATH environment variable 3-16
- path name 6-8
 - absolute 6-8
 - case sensitivity 6-9
 - length 6-11
 - partial 6-12
 - relative 6-8
 - See also* absolute path name; full path name; relative path name
- path names
 - relative 6-8
- pathdef.m
 - location 6-71
- pausing execution of file 8-119
- PDF
 - reader, preference for Help browser 4-13
- performance
 - improving for you code 9-27
- Perl variables
 - passing
 - at startup 1-21
- Plot Selector tool
 - using the 5-9
- plotting
 - from the Workspace browser 5-9
- preferences
 - Current Folder browser 6-16
 - publishing 10-76
 - publishing images 10-76
- preformatted text
 - in published MATLAB code 10-25
- printing a MATLAB Notebook
 - cell markers 11-23
 - color 11-23
 - defaults 11-23
- problems, reporting to MathWorks 4-11
- product filter in Help browser
 - preference 4-13
- profile 9-46
 - example 9-47
- profiling 9-27
- program control blocks
 - code folding and 8-158
- program elements
 - going to 6-25
- program files
 - help
 - viewing in Current Folder browser 6-22
 - naming
 - avoiding conflicts 6-67
 - viewing help for 6-23
- programs
 - running from MATLAB 3-15
 - stopping execution of 3-14
- prompt
 - when debugging 8-124
- properties
 - source control on Windows platforms 12-20
 - tab completion
 - Command Window 3-48
 - Editor 8-89
- proxy server settings 2-73 2-108
- publish configuration
 - creating multiple 10-96
 - running 10-95
- publish configurations
 - creating 10-68
 - finding 8-53
 - for files in Editor 10-67

- porting 10-107
- publish settings 10-72
- removing 8-55
- publish settings
 - in publish configurations 10-72
 - template 10-93
- publish_configurations.m file 10-107
- published MATLAB code
 - resizing images in 10-89
- publishing
 - MATLAB code and results 10-2
 - syntax highlighting and 10-77
 - using code cells and 10-2
- publishing images preferences 10-76
- publishing preferences 10-76
- Purge Output Cells 11-33
- purging output cells 11-22

Q

- quitting
 - saving workspace 1-9
- quitting MATLAB 1-7
 - confirmation 1-8

R

- R Inc Search field 8-45
- rapid code iteration 8-58
 - scenarios 8-59
- rapid development 8-58
- recovering deleted files 6-41
- recycle 6-41
- red breakpoint icons 8-122
- red underline in file 8-94
- redo
 - in Editor 8-16
- refresh
 - Current Folder browser 6-17
- registered trademarks
 - within cell 10-47
- relative path 6-8
- release
 - latest 2-77
- remote login
 - Macintosh 1-7
- removing files from source control system 12-15
- renaming
 - files and folders
 - using Current Folder browser 6-40
 - using functions 6-41
- Report
 - Code Analyzer 9-22
 - folder 9-2
- reports
 - accessing 9-2
 - Contents 9-10
 - Dependency 9-14
 - Help 9-7
 - To do 9-3
 - TODO/FIXME 9-3
 - using 9-2
- Reports
 - Coverage 9-19
 - Fix me 9-3
- requirements
 - MATLAB 1-1
- resizing windows in the desktop 2-14
- restoring
 - tools in desktop 2-22
- results in MATLAB, displaying 5-25
- revision control
 - See* source control system interface 12-1
- right-hand text limit 8-10
- rule
 - in Editor 8-10
- rules (lines) in Editor 8-62
- run configurations
 - creating 8-47
 - creating multiple 8-49

- exporting 8-52
- finding 8-53
- for files in Editor 8-46
- importing 8-52
- porting 8-52
- removing 8-55
- using 8-47

run_configurations.m file 8-52

S

save

- function 5-7

saving

- MAT-files
 - preferences 2-97
 - workspace upon quitting 1-9

screen reader 2-83

script for startup 1-19

scroll buffer for Command Window 3-63

scrolling in Command Window 3-39

search path

- adding folders to 6-73
- behavior when changing folders 6-81
- changing 6-73
- default 6-64
- description 6-64
- problems and recovering 6-79
- saving 6-77
- using 6-70
- using with different MATLAB installations 6-78
- viewing 6-72

searching

- for files by name and content 6-29
- Help browser
 - Boolean 4-5
 - exact phrase (" ") 4-5
 - wildcard (*) or partial word 4-5
- in Current Folder browser
 - by name 6-28
 - typeahead 6-28
- special characters 4-5
- text
 - Command History window 3-61
 - Command Window 3-53
 - incrementally 8-45

section breaks

- in calc zones 11-29

section titles

- in published MATLAB code 10-22

segmentation violation 1-9

segv 1-9

selecting multiple items 2-66

semicolon (;)

- after functions 3-39

session

- automatic log file 1-20

session log

- Command History 3-4

setting breakpoints 8-119

shadowed functions 6-67

shell escape 3-15

shortcut

- for MATLAB in Windows 1-2
- keys in MATLAB 2-37

shortcut keys

- Variable Editor 5-32

shortcuts

- categories 3-26
- creating
 - from Command History window 3-31
- defined 3-23
- deleting 3-26
- displaying hidden labels on the toolbar 3-28
- editing 3-26
- file 3-25
- labels, hiding 3-27
- moving 3-26
- organizing 3-26

- toolbar 3-25
- Shortcuts
 - Deleting from toolbar 3-28
- shortcuts.xml 3-25
- Show Cell Markers 11-33
- show file history on Windows platforms 12-16
- shutdown
 - MATLAB 1-7
 - options 1-9
- Simulink model
 - opening from a file 8-39
- Simulink models
 - viewing complete descriptions of 6-23
- singleCompThread startup option 1-21
- sizing windows in the desktop 2-14
- source control on UNIX platforms
 - getting files 12-33
 - locking files 12-33
- source control system interface 12-1
 - UNIX platforms 12-26
 - preferences 12-27
 - selecting system 12-27
 - supported systems 12-26
 - Windows platforms
 - adding files 12-9
 - preferences 12-5
 - selecting system 12-5
 - supported systems 12-2
- source control system interface on UNIX platforms
 - checking in files 12-30
 - checking out files 12-33
 - configuring ClearCase source control system 12-29
 - undoing file check-out 12-36
- source control system interface on Windows platforms
 - checking out files 12-12
 - comparing working copy to source control version 12-18
 - displaying file properties 12-20
 - get latest version of file 12-14
 - removing files 12-15
 - showing file history 12-16
 - starting source control system 12-21
 - troubleshooting 12-24
 - undoing file check-out 12-13
- spacing
 - output in Command Window 3-63
 - tabs in Command Window 3-64
- special characters
 - searching for 4-5
- splash screen
 - startup option 1-21
- split screen display
 - Editor 8-11
- stack
 - in Editor 8-124
 - viewing 5-2
- Start button
 - customizing 2-67
- starting MATLAB
 - DOS 1-2
 - Linux 1-6
 - Windows 1-2
- startup
 - diagnostics
 - Macintosh 1-7
 - files for MATLAB 1-19
 - folder for MATLAB 1-12
 - Macintosh, remote login 1-7
 - options for MATLAB 1-18
 - script 1-19
- startup.m
 - location 1-20
 - startup file 1-19
- statements
 - executing a group of 3-23
- stepping through files 8-125
- stops

- in files 8-119
 - strings
 - color indicators 2-9
 - saving as Unicode 2-97
 - structures
 - editing 5-25
 - tab completion 8-88
 - style preferences for text 2-103
 - styles in MATLAB Notebook
 - modifying 11-23
 - subfunctions
 - displaying in Editor status bar 8-11
 - going to in file 8-34
 - support
 - technical 4-11
 - suppressing output 3-39
 - switches
 - for startup 1-18
 - symbols
 - searching for 4-5
 - syntax
 - color indicators 2-9
 - coloring and indenting 3-41
 - highlighting 8-84
 - syntax highlighting
 - of sample MATLAB code 10-27
 - publishing code and 10-77
 - system browser
 - UNIX 2-75
 - system environment variables 3-16
 - system path for UNIX 3-16
 - system requirements
 - MATLAB 1-1
 - system Web browser 2-71
- T**
- tab
 - indenting in Editor 8-17
 - spacing in Command Window 3-64
 - tab completion
 - Command Window 3-43
 - Editor 8-85
 - tabbing desktop windows together 2-19
 - Technical Support
 - contacting 4-11
 - templates
 - for publish settings 10-93
 - MATLAB Notebook 11-23
 - temporary folder
 - for deleted files 6-41
 - text
 - converting to input cells 11-30
 - finding and replacing 8-45
 - finding in current file 8-41
 - preferences in MATLAB 2-103
 - styles in MATLAB Notebook 11-23
 - text files
 - comparing 6-53
 - threads
 - turning off multithreading 1-21
 - time
 - measured for your code 9-27
 - time-out message
 - while evaluating multiple input cells in a MATLAB Notebook 11-31
 - titles
 - in published MATLAB code 10-18 10-21 to 10-22
 - tmp/MATLAB_Files folder 6-42
 - to do reports 9-3
 - TODO/FIXME Report 9-3
 - Toggle Graph Output for Cell 11-34
 - token matching
 - preferences 3-65
 - toolbars
 - customizing 2-11
 - Editor cell mode 8-60
 - shortcuts 3-25
 - toolbox path cache

- preferences 1-24
- toolboxes
 - custom, adding to Start button 2-67
- tools in desktop
 - description 2-2
- Tooltips
 - for data 8-128
 - viewing folded code in 8-20
- trademark symbols
 - in published MATLAB code 10-47
 - within cells 10-47
- troubleshooting
 - cell highlighting 8-65
 - source control problems 12-24

U

- UNC (Universal Naming Convention) path 9-3
- uncomment 8-27
- Undefine Cells 11-34
- undo
 - in Editor 8-16
- unlocking windows from desktop 2-18 2-33
- undoing file check-out
 - on UNIX platforms 12-36
 - on Windows platforms 12-13
- Ungroup Cells 11-35
- Unicode
 - preference when saving 2-97
- UNIX
 - system path 3-16
- updates
 - to newer versions 2-77
- utilities
 - running from MATLAB 3-15

V

- validating
 - MATLAB code 9-22

- values
 - examining 8-127
- Variable Editor 5-22
 - cut, copy, paste, clear 5-34
 - decimal separator 5-46
 - keyboard shortcuts 5-32
 - opening variables in 5-25
 - preferences 5-46
 - size limitations 5-24
- variables
 - deleting or clearing 5-8
 - displaying values of 5-25
 - editing values 5-22
 - finding in current file 8-41
 - naming
 - avoiding conflicts 6-67
 - opening from a file 8-39
 - opening in the Variable Editor 5-25
 - saving 5-6
 - viewing 6-23
 - viewing during execution 8-127
 - viewing values in Editor 8-128
 - workspace 5-2
- version 2-77
 - information for MathWorks products 4-16
 - latest available 2-77
- version control
 - See* source control system interface 12-1
- vertical line
 - in Editor 8-10
- viewing desktop tools 2-14
- Visible figure property
 - embedding graphics in MATLAB Notebook 11-26

W

- warning breakpoints 8-145
- warning message identifiers 8-147
- Web

- preferences 2-108
- proxy server settings 2-73 2-108
- Web Browser
 - in MATLAB 2-71
- who 5-5
- whos 5-5
- wildcard (*)
 - Help browser search 4-5
- windows in desktop
 - about 2-2
 - arrangement 2-13
 - docking 2-14
 - opening 2-14
 - sizing 2-14
 - undocking 2-18 2-33
- Word documents
 - converting to MATLAB Notebook 11-6
- workspace
 - base 5-2
 - clearing 5-8
 - defined 5-2
 - functions 5-2
 - initializing in MATLAB Notebook 11-14
 - loading 5-7
 - MATLAB Notebook contamination 11-8

- opening 5-7
- protecting integrity 11-8
- saving 5-6
- tool 5-2
- viewing 5-3
- viewing during execution 8-127
- Workspace browser
 - description 5-2
 - plotting variables from 5-9
- wrapping
 - lines in Command Window 3-63

Y

- yellow highlighting in file 8-94
 - current cell 8-63
 - data tip 8-128

Z

- zip files
 - adding files to 6-39
 - creating 6-37
 - extracting files from 6-38
 - viewing contents of 6-37