

Spreadsheet Link™ EX

User's Guide

R2011b

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.

Spreadsheet Link™ EX User's Guide

© COPYRIGHT 1996–2011 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

May 1996	First printing	New for Version 1.0
May 1997	Second printing	Revised for Version 1.0.3
January 1999	Third printing	Revised for Version 1.0.8 (Release 11)
September 2000	Fourth printing	Revised for Version 1.1.2
April 2001	Fifth printing	Revised for Version 1.1.3
July 2002	Sixth printing	Revised for Version 2.0 (Release 13)
September 2003	Online only	Revised for Version 2.1 (Release 13SP1)
June 2004	Online only	Revised for Version 2.2 (Release 14)
September 2005	Online only	Revised for Version 2.3 (Release 14SP3)
March 2006	Online only	Revised for Version 2.3.1 (Release 2006a)
September 2006	Online only	Revised for Version 2.4 (Release 2006b)
September 2006	Seventh printing	Revised for Version 2.4 (Release 2006b)
March 2007	Online only	Revised for Version 2.5 (Release 2007a)
September 2007	Online only	Revised for Version 3.0 (Release 2007b)
March 2008	Online only	Revised for Version 3.0.1 (Release 2008a)
October 2008	Online only	Revised for Version 3.0.2 (Release 2008b)
March 2009	Online only	Revised for Version 3.0.3 (Release 2009a)
September 2009	Online only	Revised for Version 3.1 (Release 2009b)
March 2010	Online only	Revised for Version 3.1.1 (Release 2010a)
September 2010	Online only	Revised for Version 3.1.2 (Release 2010b)
April 2011	Online only	Revised for Version 3.1.3 (Release 2011a)
September 2011	Online only	Revised for Version 3.1.4 (Release 2011b)

Getting Started

1

Microsoft® Excel and MATLAB Interaction	1-2
Installing the Spreadsheet Link EX Software	1-4
System Requirements	1-4
Product Installation	1-4
Files and Folders Created by the Installation	1-4
Modifying Your System Path	1-5
After You Upgrade the Spreadsheet Link EX Software ...	1-5
Configuring the Spreadsheet Link EX Software	1-7
Configuring Microsoft® Excel 2003 and Earlier Versions ..	1-7
Configuring Microsoft® Excel 2007 and 2010	1-9
Setting Spreadsheet Link EX Preferences	1-14
Using Particular Versions of MATLAB	1-15
Working with the Microsoft® Visual Basic Editor	1-15
Starting and Stopping the Spreadsheet Link EX Software	1-17
Automatically Starting the Spreadsheet Link EX Software	1-17
Manually Starting the Spreadsheet Link EX Software ...	1-17
Connecting to an Existing MATLAB Session	1-17
Stopping the Spreadsheet Link EX Software	1-19
About Functions	1-20
How Spreadsheet Link EX Functions Differ from Microsoft® Excel Functions	1-20
Types of Spreadsheet Link EX Functions	1-20
Using Worksheets	1-21
Working with Arguments in Spreadsheet Link EX Functions	1-23
Using the MATLAB Function Wizard for the Spreadsheet Link EX Software	1-25

Examples: Using Spreadsheet Link EX Functions in Macros	1-30
Working with Dates	1-33
Localization Information	1-35

Solving Problems with the Spreadsheet Link EX Software

2

Modeling Data Sets Using Data Regression and Curve Fitting	2-2
Using Worksheets	2-2
Using Macros	2-6
Interpolating Data	2-11
Pricing Stock Options Using the Binomial Model	2-15
Calculating and Plotting the Efficient Frontier of Financial Portfolios	2-19
Mapping Time and Bond Cash Flows	2-24

Error Messages and Troubleshooting

3

Worksheet Cell Errors	3-2
Microsoft® Excel Errors	3-5
Data Errors	3-8

Matrix Data Errors	3-8
Errors When Opening Saved Worksheets	3-8
Startup Errors	3-10
Audible Error Signals	3-10

Function Reference

4

Link Management	4-2
Data Management	4-3

Functions — Alphabetical List

5

Examples

A

Macro Examples	A-2
Financial Examples	A-2

Index

Getting Started

- “Microsoft® Excel and MATLAB Interaction” on page 1-2
- “Installing the Spreadsheet Link EX Software” on page 1-4
- “Configuring the Spreadsheet Link EX Software” on page 1-7
- “Starting and Stopping the Spreadsheet Link EX Software” on page 1-17
- “About Functions” on page 1-20
- “Working with Dates” on page 1-33
- “Localization Information” on page 1-35

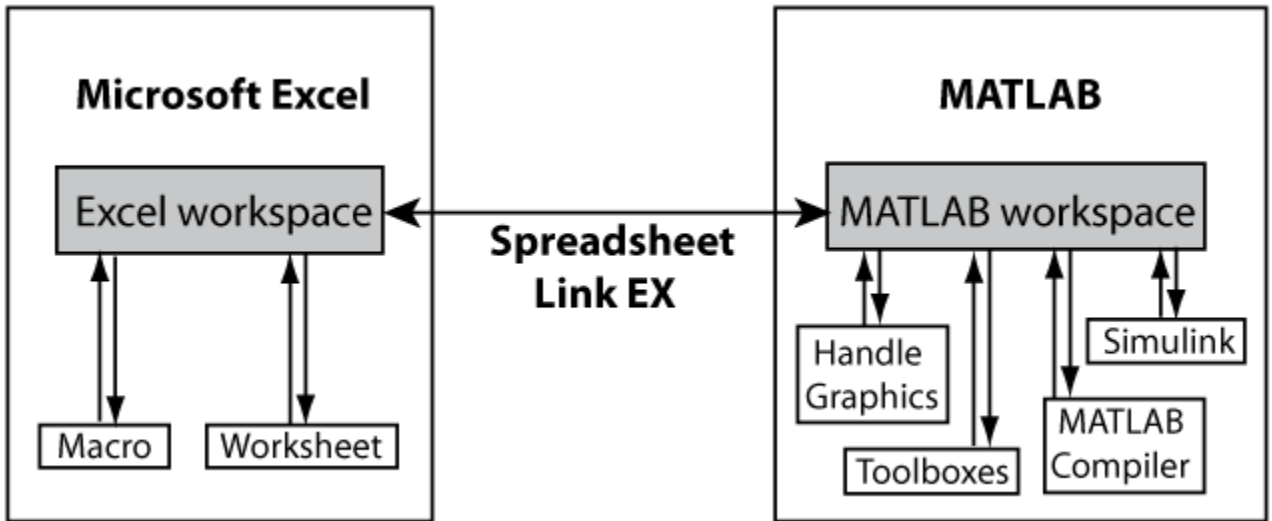
Microsoft Excel and MATLAB Interaction

Spreadsheet Link™ EX Add-In integrates the Microsoft® Excel® and MATLAB® products in a computing environment running Microsoft® Windows®. It connects the Excel® interface to the MATLAB workspace, enabling you to use Excel worksheet and macro programming tools to leverage the numerical, computational, and graphical power of MATLAB.

You can use Spreadsheet Link EX functions in an Excel worksheet or macro to exchange and synchronize data between Excel and MATLAB, without leaving the Excel environment. With a small number of functions to manage the link and manipulate data, the Spreadsheet Link EX software is powerful in its simplicity.

Note This documentation uses the terms *worksheet* and *spreadsheet* interchangeably.

The Spreadsheet Link EX software supports MATLAB two-dimensional numeric arrays, one-dimensional character arrays (strings), and two-dimensional cell arrays. It does not work with MATLAB multidimensional arrays and structures.



Installing the Spreadsheet Link EX Software

In this section...
“System Requirements” on page 1-4
“Product Installation” on page 1-4
“Files and Folders Created by the Installation” on page 1-4
“Modifying Your System Path” on page 1-5
“After You Upgrade the Spreadsheet Link EX Software” on page 1-5

System Requirements

For information on hardware and software requirements for this product, see <http://www.mathworks.com/products/excellink/requirements.html>.

Product Installation

Install the Microsoft Excel product *before* you install the MATLAB and Spreadsheet Link EX software. To install the Spreadsheet Link EX Add-In, follow the instructions in the MATLAB installation documentation. Select the **Spreadsheet Link EX** check box when choosing components to install.

Note If you have several versions of MATLAB installed on your computer, Spreadsheet Link EX software uses the version that you registered last.

Files and Folders Created by the Installation

Note Throughout this document the notation *matlabroot* is the MATLAB root folder, the folder where the MATLAB software is installed on your system.

The Spreadsheet Link EX installation program creates a subfolder under *matlabroot\toolbox*. The *exlink* folder contains the following files:

- `excllink.xla`: The Spreadsheet Link EX Add-In for Microsoft Excel 2003 and earlier versions
- `excllink2007.xlam`: The Spreadsheet Link EX Add-In for Microsoft Excel 2007 or 2010
- `ExliSamp.xls`: Spreadsheet Link EX example files described in this documentation

The Spreadsheet Link EX software uses `Kernel32.dll`, which should already be in the appropriate Windows system folder (for example, `C:\Winnt\system32`). If not, consult your system administrator.

Modifying Your System Path

Add `matlabroot\bin` to your system path. For more information about editing your system path, consult your Windows documentation or your system administrator.

After You Upgrade the Spreadsheet Link EX Software

If MATLAB and Spreadsheet Link EX are installed on your computer, to upgrade to a newer version:

- 1 Install the new version of MATLAB and the Spreadsheet Link EX software.
- 2 Start MATLAB and a Microsoft Excel session.
- 3 Configure the Spreadsheet Link EX software.
- 4 If you have existing workbooks with macros that use the Spreadsheet Link EX software, update the references to the Spreadsheet Link EX software in each workbook.

To update the references in an existing workbook in Microsoft Excel 2003 or earlier versions:

- 1 In a Microsoft Excel session, open the Visual Basic® Editor window by selecting **Tools > Macros > Visual Basic Editor**.
- 2 In the left pane, select a module for which you want to update a reference.

- 3** From the main menu, select **Tools > References**.
- 4** In the References dialog box, select the **SpreadsheetLinkEx** check box.
- 5** Click **OK**.

To update the references in an existing workbook in Microsoft Excel 2007 or 2010:

- 1** In a Microsoft Excel session, open the Visual Basic Editor window by clicking **Visual Basic** on the **Developer** tab. (If you do not find the **Developer** tab, see the Excel Help.)
- 2** In the left pane, select a module for which you want to update a reference.
- 3** From the main menu, select **Tools > References**.
- 4** In the References dialog box, select the **SpreadsheetLink2007** or **SpreadsheetLink2010** check box.
- 5** Click **OK**.

Configuring the Spreadsheet Link EX Software

In this section...

“Configuring Microsoft® Excel 2003 and Earlier Versions” on page 1-7

“Configuring Microsoft® Excel 2007 and 2010” on page 1-9

“Setting Spreadsheet Link EX Preferences” on page 1-14

“Using Particular Versions of MATLAB” on page 1-15

“Working with the Microsoft® Visual Basic Editor” on page 1-15

Configuring Microsoft Excel 2003 and Earlier Versions

To enable the Spreadsheet Link EX Add-In, start Microsoft Excel and follow these steps:

- 1 Click **Tools > Add-Ins**. The Add-Ins dialog box appears.
- 2 Click **Browse**.
- 3 Select `matlabroot\toolbox\exlink\excllink.xla`.

Note Throughout this document the notation `matlabroot` is the MATLAB root folder, the folder where the MATLAB software is installed on your system.

- 4 Click **OK**.

In the Add-Ins dialog box, the **Spreadsheet Link EX for use with MATLAB** check box is now selected.

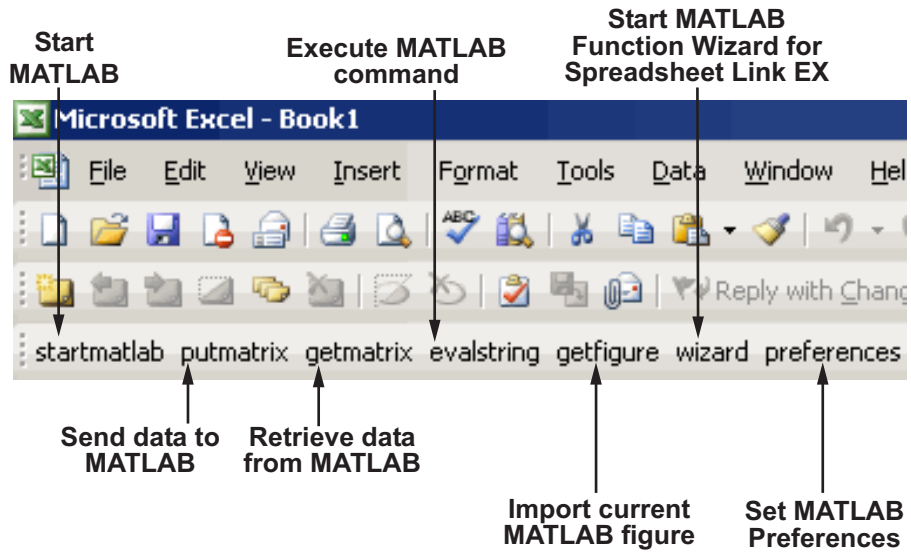
- 5 Click **OK** to exit the Add-Ins dialog box.

The Spreadsheet Link EX Add-In loads now and with each subsequent Excel session.

The **MATLAB Command Window** button appears on the Microsoft Windows taskbar.



The Spreadsheet Link EX toolbar appears on your Excel worksheet.




The Spreadsheet Link EX software is now ready for use.

Configuring Microsoft Excel 2007 and 2010

To enable the Spreadsheet Link EX Add-In, start a Microsoft Excel session and follow these steps. If you use Microsoft Excel 2007:



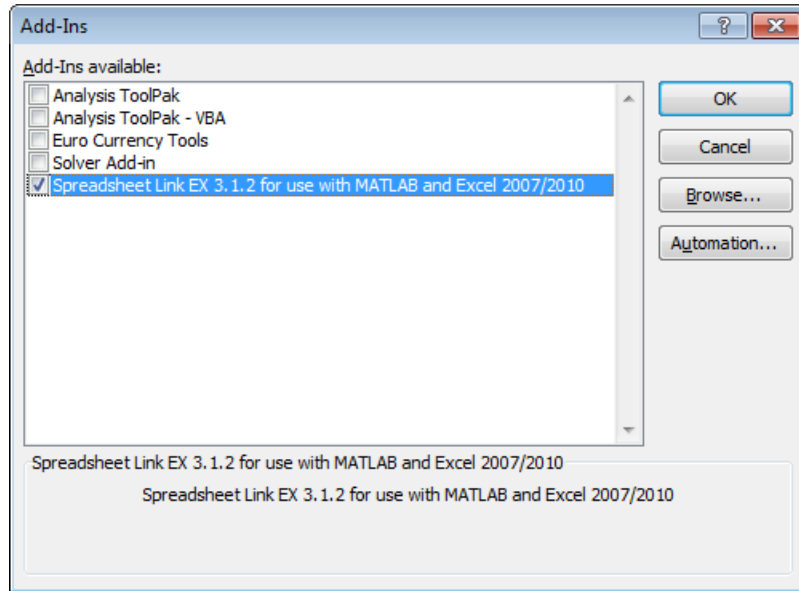
- 1 Click , the Microsoft Office Button.
- 2 Click **Excel Options**. The Excel Options dialog box appears.

If you use Microsoft Excel 2010:

- 1 Select **File** from the main menu.
- 2 Click **Options**. The Excel Options dialog box appears.

The next steps are the same for both versions:

- 3 Click **Add-Ins**.
- 4 From the **Manage** selection list, choose **Excel Add-Ins**.
- 5 Click **Go**. The Add-Ins dialog box appears.
- 6 Click **Browse**.
- 7 Select `matlabroot\toolbox\exlink\excllink2007.xlam`.
- 8 Click **Open**. In the Add-Ins dialog box, the **Spreadsheet Link EX for use with MATLAB** check box is now selected.



9 Click **OK** to close the Add-Ins dialog box.

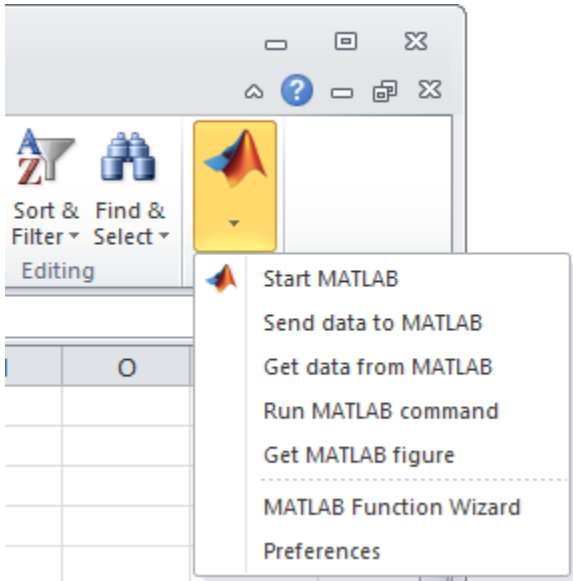
10 Click **OK** to close the Excel Options dialog box.

The Spreadsheet Link EX Add-In loads now and with each subsequent Excel session.

The **MATLAB Command Window** button appears on the Microsoft Windows taskbar.

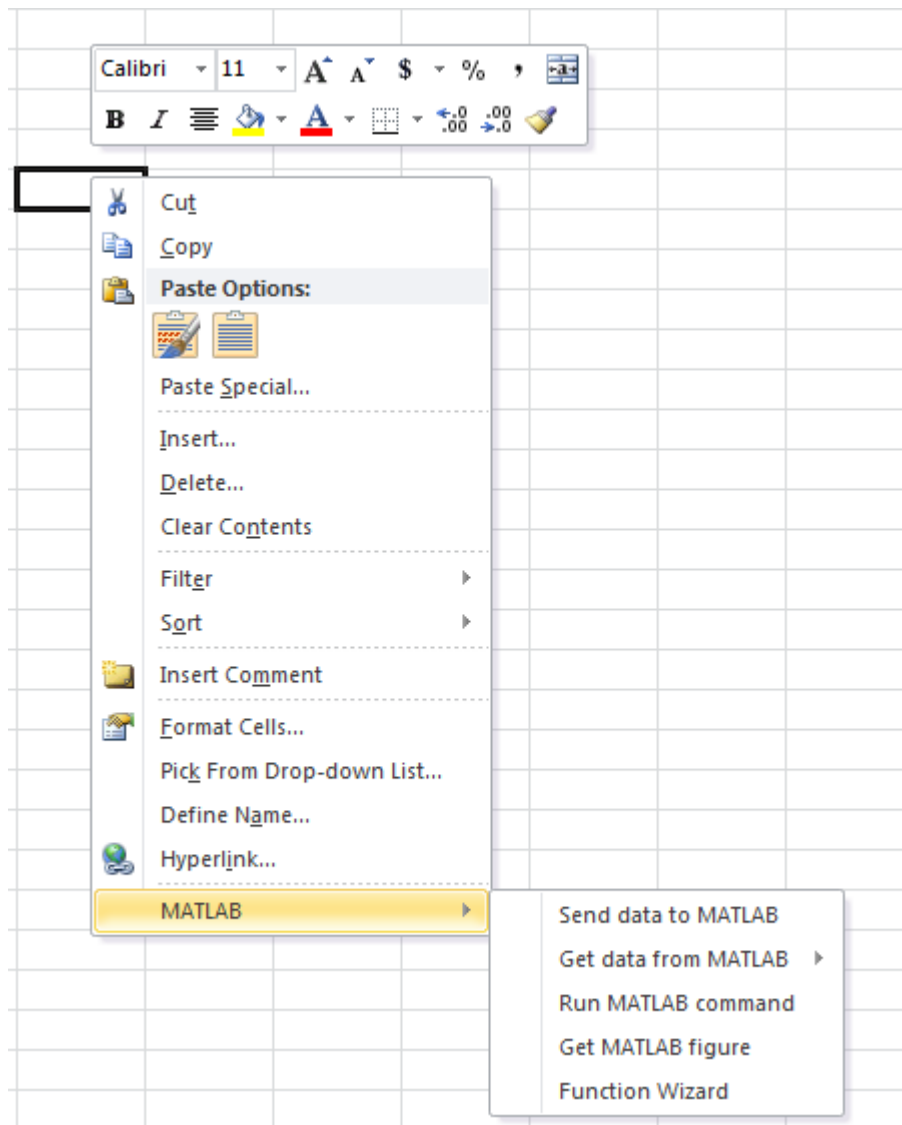


The MATLAB group appears on the top right of the Home tab in your Excel worksheet:



The Spreadsheet Link EX software is now ready for use.

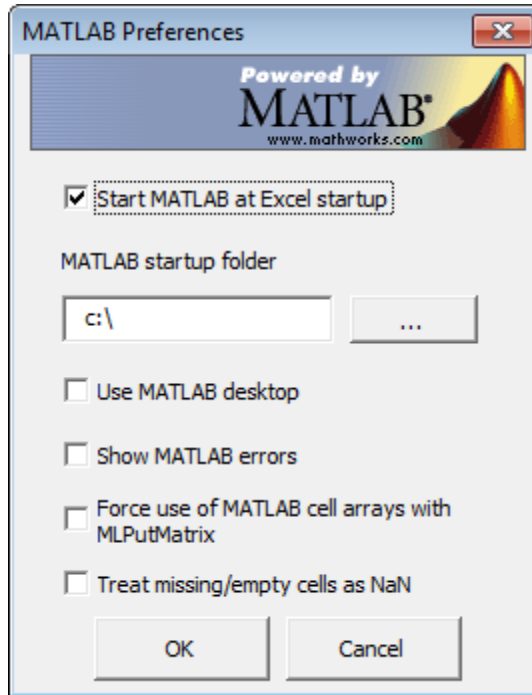
Right-click a cell for MATLAB options. The following menu appears:



Caution Using both the 2003 and 2007/2010 Add-Ins referenced in Excel 2007/2010 causes problems with the context-sensitive menu. Use only one Add-In at a time to avoid this issue.

Setting Spreadsheet Link EX Preferences

Use the Preferences dialog box to set Spreadsheet Link EX preferences. Click the **preferences** button in the Excel toolbar or MATLAB group to open this dialog box.



Preferences include:

- **Start MATLAB at Excel startup** starts a MATLAB session automatically when an Excel session starts. By default, this option is enabled.
- **MATLAB startup folder** lets you specify the startup folder for your MATLAB session.
- **Use MATLAB desktop** starts the MATLAB desktop, including the current folder, workspace, command history, and Command Window panes, when an Excel session starts.

- **Show MATLAB errors** displays MATLAB error messages in Excel worksheet cells. Without this option, worksheet cells display Excel error messages. See “Worksheet Cell Errors” on page 3-2.
- **Force use of MATLAB cell arrays with MLPutMatrix** enables the MLPutMatrix function to use cell arrays for data transfer between the Excel software and the MATLAB workspace.
- **Treat missing/empty cells as NaN** sets data in missing or empty cells to NaN or zero.

Using Particular Versions of MATLAB

If there are several versions of MATLAB installed on your computer, the Spreadsheet Link EX software uses the last registered version by default. Typically, the last registered version is the latest version you have installed. To choose a version of MATLAB other than the default one, use one of the following approaches:

- Start that version of MATLAB as an automation server. For this, open the DOS window, cd to the folder in which you want to start MATLAB, and type `matlab -automation` at the DOS prompt. Then connect to that (already running) MATLAB session. For details, see “Connecting to an Existing MATLAB Session” on page 1-17.
- Change the last registered version to the version that you want to use.

To change the last registered version of MATLAB:


- 1 Shut down all MATLAB and Excel sessions.
- 2 Open a Command Prompt window, and using `cd`, change to the `bin\win64` or `bin\win32` subfolder of the MATLAB installation folder.
- 3 Enter the command:

```
.\matlab /regserver
```

Working with the Microsoft Visual Basic Editor

Follow these steps to enable the Spreadsheet Link EX software as a Reference in the Microsoft® Visual Basic® Editor:

1 Open a Visual Basic session.

- If you are running the Excel 2003 software, click **Tools > Macro > Visual Basic Editor**.
- If you are running the Excel 2007 or 2010 software, click the Visual Basic button () on the **Developer** tab, or press **Alt+F11**.

Note For instructions how to display the Developer tab, see Excel Help.

2 In the Visual Basic toolbar, click **Tools > References**.

3 In the References — VBA Project dialog box, select the **SpreadsheetLinkEX** check box.

4 Click **OK**.

Starting and Stopping the Spreadsheet Link EX Software

In this section...

“Automatically Starting the Spreadsheet Link EX Software” on page 1-17

“Manually Starting the Spreadsheet Link EX Software” on page 1-17

“Connecting to an Existing MATLAB Session” on page 1-17

“Stopping the Spreadsheet Link EX Software” on page 1-19

Automatically Starting the Spreadsheet Link EX Software

When installed and configured according to the instructions in “Configuring the Spreadsheet Link EX Software” on page 1-7, the Spreadsheet Link EX and MATLAB software automatically start when you start a Microsoft Excel session.

Manually Starting the Spreadsheet Link EX Software

To start the Spreadsheet Link EX and MATLAB software manually from the Excel interface:

1 Click **Tools > Macro**. (In Excel 2007, click the **View** tab or the **Developer** tab, and then click the **Macros** button.)

2 Enter `matlabinit` into the **Macro Name/Reference** box.

For more information about the `matlabinit` function, see Chapter 4, “Function Reference”.

3 Click **Run**. The MATLAB Command Window button appears on the Microsoft Windows taskbar.

Connecting to an Existing MATLAB Session

To connect a new Excel session to an existing and already running MATLAB session:

1 In MATLAB, enter the following command:

```
enableservice('AutomationServer', true)
```

This command converts an existing MATLAB session into an Automation server.

- 2 Start a new Excel session. It automatically connects to the running MATLAB session.

Alternatively, you can start MATLAB as an automation server from the beginning. To start MATLAB as an automation server, use the `automation` command-line option:

```
matlab -automation
```

This command does not start MATLAB in a full desktop mode. To start MATLAB in a full desktop mode, use the `-desktop` option:

```
matlab -automation -desktop
```

If you always use MATLAB as an automation server, modify the shortcut that you use to start MATLAB:

- 1 Right-click your MATLAB shortcut icon. (You can use the icon on your desktop or in the Windows **Start** menu.)
- 2 Select **Properties**.
- 3 Click the **Shortcut** tab.
- 4 Add the string `-automation` in the **Target** field. Remember to leave a space between `matlab.exe` and `/automation`.
- 5 Click **OK**.

Note This option works only if the current MATLAB session is a registered automation version. If not, the Excel software starts a new MATLAB session rather than connecting to the existing one.

Stopping the Spreadsheet Link EX Software

If you started the Spreadsheet Link EX and MATLAB software from the Excel interface:

- To stop both the Spreadsheet Link EX and MATLAB software, close the Excel session as you normally would.
- To stop the Spreadsheet Link EX and MATLAB software and leave the Excel session running, enter the `=MLClose()` command into an Excel worksheet cell. You can use the `MLOpen` or `matlabinit` functions to restart the Spreadsheet Link EX and MATLAB sessions manually.

If you connected an Excel session to an existing MATLAB session, close Excel and MATLAB sessions separately. Closing one session does not automatically close the other.

About Functions

In this section...
“How Spreadsheet Link EX Functions Differ from Microsoft® Excel Functions” on page 1-20
“Types of Spreadsheet Link EX Functions” on page 1-20
“Using Worksheets” on page 1-21
“Working with Arguments in Spreadsheet Link EX Functions” on page 1-23
“Using the MATLAB Function Wizard for the Spreadsheet Link EX Software” on page 1-25
“Examples: Using Spreadsheet Link EX Functions in Macros” on page 1-30

How Spreadsheet Link EX Functions Differ from Microsoft Excel Functions

- Spreadsheet Link EX functions *perform an action*, while Microsoft Excel functions *return a value*.
- Spreadsheet Link EX function names *are not* case sensitive; that is, `MLPutMatrix` and `mlputmatrix` are the same.
- MATLAB function names and variable names *are* case sensitive; that is, `BONDS`, `Bonds`, and `bonds` are three different MATLAB variables.

Note Excel operations and function keys may behave differently with Spreadsheet Link EX functions.

Types of Spreadsheet Link EX Functions

Spreadsheet Link EX functions manage the connection and data exchange between the Excel software and the MATLAB workspace, without your ever needing to leave the Excel environment. You can run functions as worksheet cell formulas or in macros. The Spreadsheet Link EX software enables the Excel product to act as an easy-to-use data-storage and

application-development front end for the MATLAB software, which is a powerful computational and graphical processor.

There are two types of Spreadsheet Link EX functions: link management functions and data management functions.

Link management functions initialize, start, and stop the Spreadsheet Link EX and MATLAB software. You can run any link management function other than `matlabinit` as a worksheet cell formula or in macros. You must run the `matlabinit` function from the Excel **Tools > Macro** menu, or in macro subroutines.

Data management functions copy data between the Excel software and the MATLAB workspace, and execute MATLAB commands in the Excel interface. You can run any data management function other than `MLPutVar` and `MLGetVar` as a worksheet cell formula or in macros. The `MLPutVar` and `MLGetVar` functions can run only in macros.

For more information about Spreadsheet Link EX functions, see Chapter 4, “Function Reference”.

Using Worksheets

Entering Functions into Worksheet Cells

Spreadsheet Link EX functions expect A1-style worksheet cell references; that is, columns designated with letters and rows with numbers (the default reference style). If your worksheet shows columns designated with numbers instead of letters:

- 1 Click **Tools > Options**.
- 2 Click the **General** tab.
- 3 Under **Settings**, clear the **R1C1 reference style** check box.

Enter Spreadsheet Link EX functions directly into worksheet cells as worksheet formulas. Begin worksheet formulas with `+` or `=` and enclose function arguments in parentheses. The following example uses `MLPutMatrix` to put the data in cell C10 into matrix A:

```
=MLPutMatrix("A", C10)
```

For more information on specifying arguments in Spreadsheet Link EX functions, see “Working with Arguments in Spreadsheet Link EX Functions” on page 1-23.

Note Do not use the Excel Function Wizard. It can generate unpredictable results.

After a Spreadsheet Link EX function successfully executes as a worksheet formula, the cell contains the value 0. While the function executes, the cell might continue to show the formula you entered.

To change the active cell when an operation completes, click **Excel Tools Options > Edit > Move Selection after Enter**. This action provides a useful confirmation for lengthy operations.

Automatic Calculation Mode Vs. Manual Calculation Mode

Spreadsheet Link EX functions are most effective in automatic calculation mode. To *automate* the recalculation of a Spreadsheet Link EX function, add to it a cell whose value changes. In the following example, the `MLPutMatrix` function reexecutes when the value in cell C1 changes:

```
=MLPutMatrix("bonds", D1:G26) + C1
```

Note Be careful to avoid creating endless recalculation loops.

To use `MLGetMatrix` in manual calculation mode:

- 1** Enter the function into a cell.
- 2** Press **F2**.
- 3** Press **Enter**. The function executes.

Spreadsheet Link EX functions do not automatically adjust cell addresses. If you use explicit cell addresses in a function, you must edit the function arguments to reference a new cell address when you do either of the following:

- Insert or delete rows or columns.
- Move or copy the function to another cell.

Note Pressing **F9** to recalculate a worksheet affects only Excel functions. This key does not operate on Spreadsheet Link EX functions.

Working with Arguments in Spreadsheet Link EX Functions

This section describes tips for managing variable-name arguments and data-location arguments in Spreadsheet Link EX functions.

Variable-Name Arguments

- You can *directly* or *indirectly* specify a variable-name argument in most Spreadsheet Link EX functions:
 - To specify a variable name directly, enclose it in double quotation marks; for example, `MLDeleteMatrix("Bonds")`.
 - To specify a variable name as an indirect reference, enter it without quotation marks. The function evaluates the contents of the argument to get the variable name. The argument must be a worksheet cell address or range name; for example, `MLDeleteMatrix(C1)`.

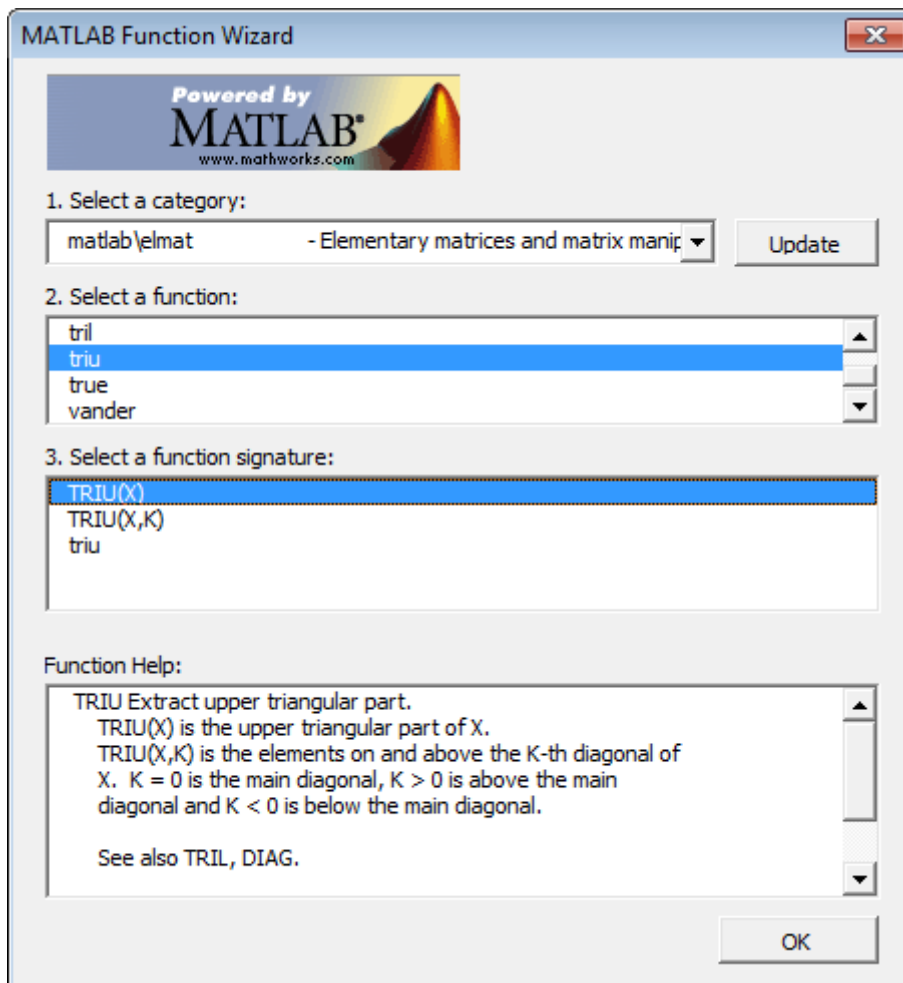
Data-Location Arguments

- A data-location argument must be a worksheet cell address or range name.
- Do not enclose a data-location argument in quotation marks (except in `MLGetMatrix`, which has unique argument conventions).
- A data-location argument can include a worksheet number; for example, `Sheet3!B1:C7` or `Sheet2!OUTPUT`.

Note You can reference special characters as part of a worksheet name in `MLGetMatrix` or `MLPutMatrix` by embedding the worksheet name within single quotation marks (' ').

Using the MATLAB Function Wizard for the Spreadsheet Link EX Software

The MATLAB Function Wizard for the Spreadsheet Link EX software allows you to browse MATLAB folders and run functions from within the Excel interface.



You can use this wizard to:

- 1** Display a list of all MATLAB working folders and function categories.

All folders or categories in the current MATLABPATH display in the **Select a category** field. Click an entry in the list to select it. Each entry in the list displays as a folder path and a description read from the `Contents.m` file in that folder. If no `Contents.m` file is found, the folder or category display notifies you as follows:

```
finance\finsupport -(No table of contents file)
```

To refresh the folder/category list, click the **Update** button.

- 2** Select a particular folder or category, and list functions available for that folder or category.

After you select a folder or category, the **Select a function** field displays available functions for that folder or category. Click a function name to select it.

Tip The Function Wizard does not allow you to access MATLAB constructors and methods. You can write a wrapper function for a method or a constructor and access that wrapper. See “Using the Function Wizard to Access Custom MATLAB Functions” on page 1-28.

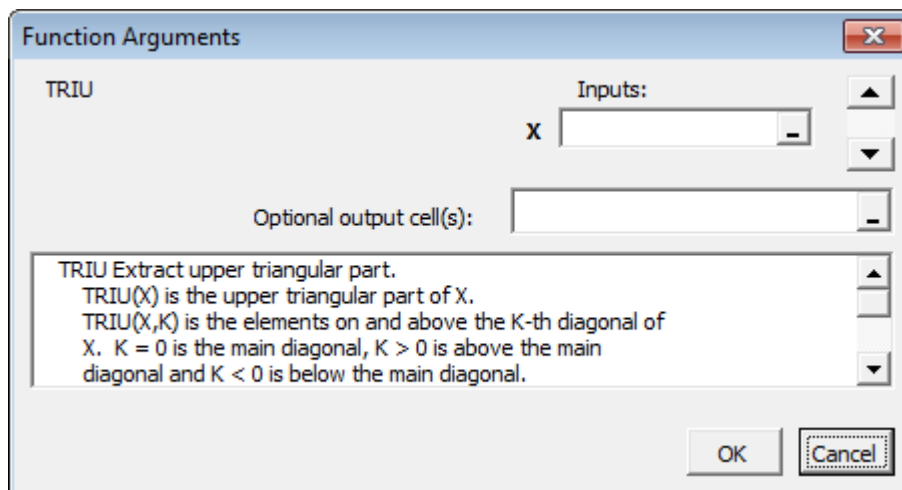
- 3** Select a function signature and enter a formula into the current spreadsheet cell.

After you select a function, the **Select a function signature** field displays available signatures for that function. Click a function signature to select it.

- 4** View help for the selected function.

The **Function Help** field displays help for the selected function.

When you click a function signature, the **Function Arguments** dialog box appears.



This dialog box allows you to specify the cells that contain input arguments and the cells where to display outputs. By default, the output of the selected function appears in the current spreadsheet cell using the Spreadsheet Link EX function `matlabfcn`. In the following example, the output displays in the current spreadsheet cell and generates a MATLAB figure:

```
=matlabfcn("plot", Sheet1!$B$2:$D$4)
```

Specifying a target range of cells using the **Optional output cell(s)** field in the Function Arguments dialog box causes the selected function to appear in the current spreadsheet cell as an argument of the `matlabsub` function. In addition, `matlabsub` includes an argument that indicates where to write the function's output. In the following example, the data from A2 is input to the `rand` function, whose target cell is B2:

```
=matlabsub("rand", "Sheet1!$B$2", Sheet1!$A$2)
```

Tip Although the Function Wizard lets you specify multiple output cells, it does not return multiple outputs. If you specify a range of output cells, the wizard returns the first output argument starting in the first output cell.

For example, if a function returns two separate elements **a** and **b**, and you specify **A1:A2** as output cells, the Function Wizard displays element **a** in cell **A1**. It discards element **b**. If an output is a matrix, the Function Wizard displays all elements of that matrix starting in the first output cell.

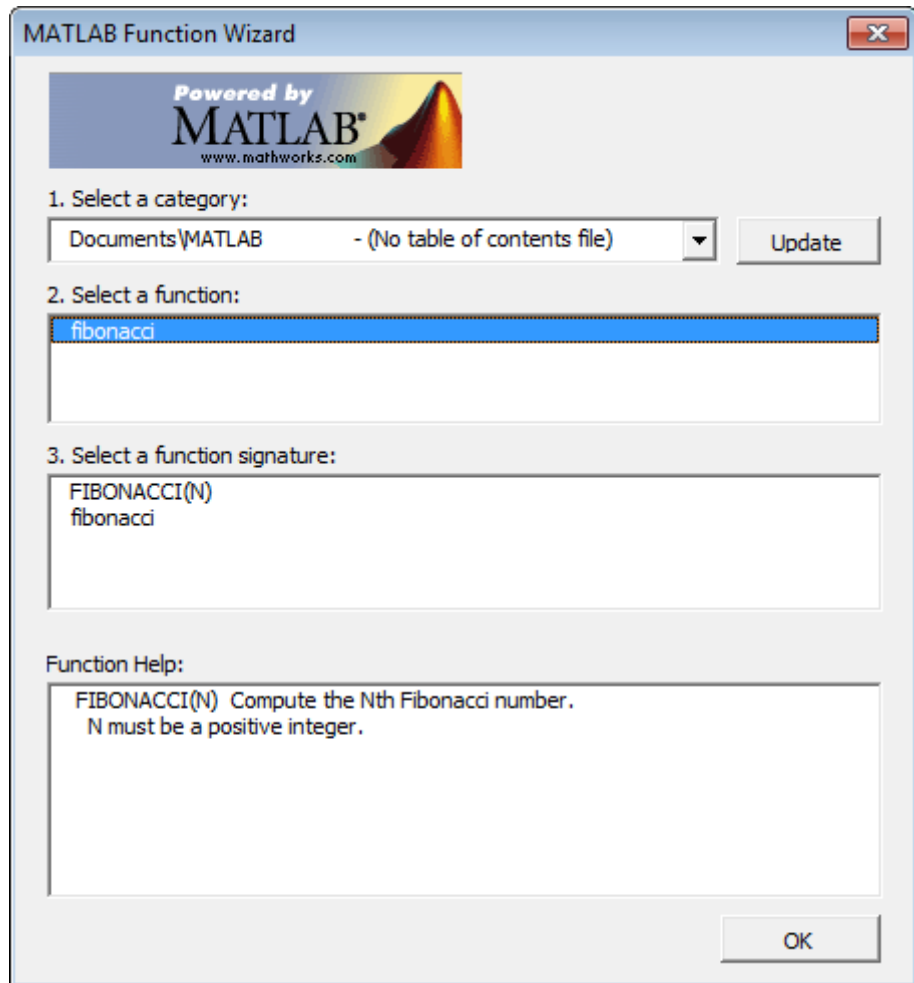
Using the Function Wizard to Access Custom MATLAB Functions

To access your custom MATLAB functions from the Function Wizard:

- 1** In MATLAB, create and save your function. For example, write the function that computes the Fibonacci numbers and save it in the folder `Documents\MATLAB`:

```
function f = fibonacci(n)
%FIBONACCI(N) Compute the Nth Fibonacci number.
% N must be a positive integer.
if n < 0
    error('Invalid number.')
elseif n == 0
    f = 0;
elseif n == 1
    f = 1;
else
    f = fibonacci(n - 1) + fibonacci(n - 2);
end;
end
```

- 2** Add the folder where you saved the function to the MATLAB search path. To add the folder to the search path, use the `pathtool` function or select **File > Set Path** in the MATLAB desktop.
- 3** In Excel, open the MATLAB Function Wizard and select the folder where you saved your function.



The Function Wizard does not allow you to access MATLAB constructors and methods. To be able to access a method or a constructor from the Function Wizard, write a wrapper function for that method or constructor. For example, to access the `timeseries(DATA)` constructor from the Function Wizard, write the following wrapper function:

```
function TS = timeseries_wrapper(DATA)
% timeseries_wrapper(DATA) is a wrapper function
% for TIMESERIES(DATA)
% TS = TIMESERIES(DATA) creates a time series object TS using
% data DATA. By default, the time vector ranges from 0 to N-1,
% where N is the number of samples, and has an interval of 1
% second. The default name of the TS object is 'unnamed'.
T = timeseries(DATA);
TS = T.data;
end
```

Examples: Using Spreadsheet Link EX Functions in Macros

About the Examples

This section contains examples that show how to manipulate MATLAB data using Spreadsheet Link EX.

- For an example of how to exchange data between the MATLAB and Excel workspaces, see “Importing and Exporting Data Between the Microsoft® Excel Interface and the MATLAB Workspace” on page 1-33.
- For an example of how to export data from the MATLAB workspace and display it in an Excel worksheet, see “Sending MATLAB Data to an Excel Worksheet and Displaying the Results” on page 1-30.

Sending MATLAB Data to an Excel Worksheet and Displaying the Results

In this example, you run MATLAB commands using VBA, send MATLAB data to the Excel software, and display the results in an Excel dialog box.

- 1** Start an Excel session.
- 2** Initialize the MATLAB session by clicking the **startmatlab** button in the Spreadsheet Link EX toolbar or by running the `matlabinit` function.
- 3** If the Spreadsheet Link EX Add-In is not enabled, enable it.

- For instructions on enabling this Add-In for the Excel 2003 software, see “Configuring Microsoft® Excel 2003 and Earlier Versions” on page 1-7.
 - For instructions on enabling this Add-In for the Excel 2007 software, see “Configuring Microsoft® Excel 2007 and 2010” on page 1-9.
- 4** Enable the Spreadsheet Link EX software as a Reference in the Microsoft Visual Basic Editor. For instructions, see “Working with the Microsoft® Visual Basic Editor” on page 1-15.
 - 5** In the Visual Basic Editor, create a module.
 - a** Right-click the **Microsoft Excel Objects** folder in the **Project — VBAProject** browser.
 - b** Select **Insert > Module**.
 - 6** Enter the following code into the module window:

```
Option Base 1
Sub Method1()

    MLShowMatlabErrors "yes"

    '''To MATLAB:
    Dim Vone(2, 2) As Double      'Input
    Vone(1, 1) = 1
    Vone(1, 2) = 2
    Vone(2, 1) = 3
    Vone(2, 2) = 4

    MLPutMatrix "a", Range("A1:B2")
    MLPutVar "b", Vone
    MLEvalString ("c = a*b")
    MLEvalString ("d = eig(c)")

    '''From MATLAB:
    Dim Vtwo As Variant          'Output
    MLGetVar "c", Vtwo
    MsgBox "c is " & Vtwo(1, 1)
```

```
MLGetMatrix "b", Range("A7:B8").Address
MatlabRequest
MLGetMatrix "c", "Sheet1!A4:B5"
MatlabRequest

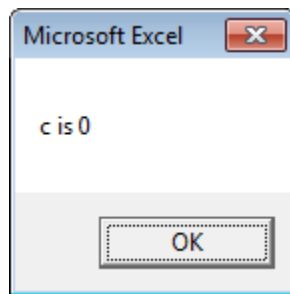
Sheets("Sheet1").Select
Range("A10").Select
MLGetMatrix "d", ActiveCell.Address
MatlabRequest

End Sub
```

Tip Copy and paste this code into the Visual Basic Editor from the HTML version of the documentation.

7 Run the code. Press **F5** or click **Run > Run Sub/UserForm**.

The following dialog box appears.



8 Click **OK** to close the dialog box.

Note Do not include `MatlabRequest` in a macro function unless the macro function is called from a subroutine.

Tip In macros, leave a space between the function name and the first argument; do not use parentheses.

Importing and Exporting Data Between the Microsoft Excel Interface and the MATLAB Workspace

- This example uses `MLGetMatrix` in a macro subroutine to export data from the MATLAB matrix `A` into the Excel worksheet `Sheet1`.

```
Sub Test1()  
    MLGetMatrix "A", "Sheet1!A5"  
    MatlabRequest  
End Sub
```

Note The `MatlabRequest` function initializes internal Spreadsheet Link EX variables and enables `MLGetMatrix` to function in the subroutine.

- This example uses `MLPutMatrix` in a macro subroutine to import data into the MATLAB matrix `A`, from a specified cell range in the Excel worksheet `Sheet1`.

```
Sub Test2()  
    Set myRange = Range("A1:C3")  
    MLPutMatrix "A", myRange  
End Sub
```

Working with Dates

Default Microsoft Excel date numbers represent the number of days that have passed since January 1, 1900. For example, January 1, 1950 is represented as 18264 in the Excel software.

However, MATLAB date numbers represent the number of days that have passed since January 1, 0000, so January 1, 1950 is represented as 712224 in the MATLAB software. Therefore, the difference in dates between the

Excel software and the MATLAB software is a constant, 693960 (712224 minus 18264).

To use date numbers in MATLAB calculations, apply the 693960 constant as follows:

- Add it to Excel date numbers that are read into the MATLAB software.
- Subtract it from MATLAB date numbers that are read into the Excel software.

Note If you use the optional Excel 1904 date system, the constant is 695422.

Dates are stored internally in the Excel software as numbers and are unaffected by locale.

Localization Information

This document uses the Microsoft Excel software with an English (United States) Microsoft Windows regional setting for illustrative purposes. If you use the Spreadsheet Link EX software with a non-English (United States) Windows desktop environment, certain syntactical elements may not work as illustrated. For example, you may have to replace the comma (,) delimiter within Spreadsheet Link EX commands with a semicolon (;) or other operator.

Please consult your Windows documentation to determine which regional setting differences exist among non-U.S. versions.

Solving Problems with the Spreadsheet Link EX Software

- “Modeling Data Sets Using Data Regression and Curve Fitting” on page 2-2
- “Interpolating Data” on page 2-11
- “Pricing Stock Options Using the Binomial Model” on page 2-15
- “Calculating and Plotting the Efficient Frontier of Financial Portfolios” on page 2-19
- “Mapping Time and Bond Cash Flows” on page 2-24

Modeling Data Sets Using Data Regression and Curve Fitting

In this section...
“Using Worksheets” on page 2-2
“Using Macros” on page 2-6

Regression techniques and curve fitting attempt to find functions that describe the relationship among variables. In effect, they attempt to build mathematical models of a data set. MATLAB matrix operators and functions simplify this task.

This example shows both data regression and curve fitting. It also executes the same example in a worksheet version and a macro version. The example uses Microsoft Excel worksheets to organize and display the data. Spreadsheet Link EX functions copy the data to the MATLAB workspace, and then executes MATLAB computational and graphic functions. The macro version also returns output data to an Excel worksheet.

This example is included in the Spreadsheet Link EX product. To run it:

- 1 Start Excel, Spreadsheet Link EX, and MATLAB sessions.
- 2 Navigate to the folder *matlabroot\toolbox\exlink*.
- 3 Open the file *ExliSamp.xls*
- 4 Execute the example as needed.

Using Worksheets

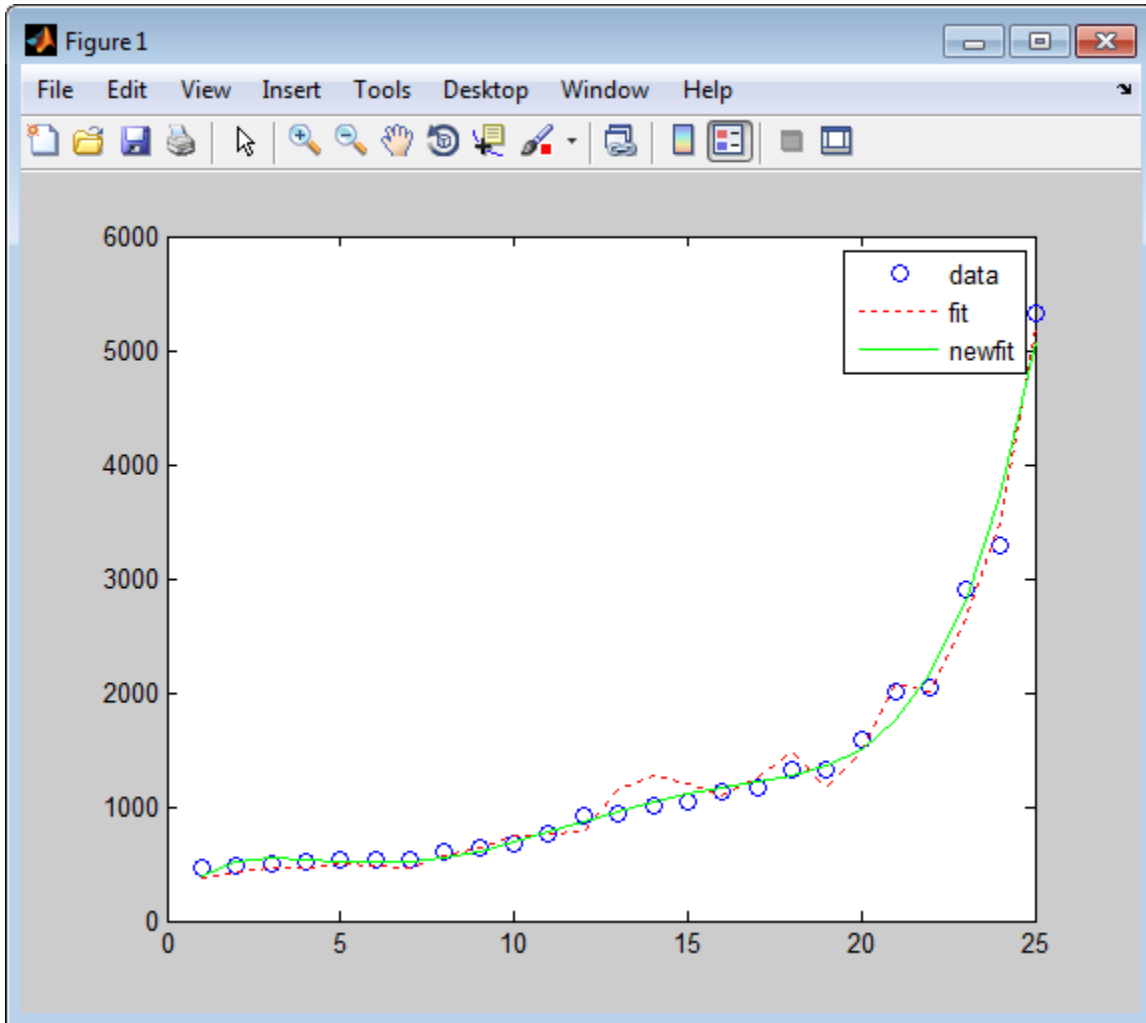
- 1 Click the **Sheet1** tab on the *ExliSamp.xls* window. The worksheet for this example appears.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Regression and Curve Fitting											
2												
3		DATA			Spreadsheet Link EX Functions							
4	35	207	1325		1. Transfer the data to MATLAB.							
5	17	90	533		#MATLAB? <== MLPutMatrix("data",DATA)							
6	43	180	1013									
7	41	187	1163		2. Set up data for regression.							
8	177	552	5326		#MATLAB? <== MLEvalString("y = data(:,3)")							
9	57	354	2043		#MATLAB? <== MLEvalString("e = ones(length(data),1)")							
10	20	101	602		#MATLAB? <== MLEvalString("A = [e data(:, 1:2)]")							
11	18	91	532									
12	17	86	543		3. Compute regression coefficients.							
13	35	180	1134		#MATLAB? <== MLEvalString("beta = A\y")							
14	25	136	766									
15	17	84	495		4. Calculate regressed result.							
16	23	102	635		#MATLAB? <== MLEvalString("fit = A*beta")							
17	24	148	913									
18	40	292	1591		5. Compare original data with regression results.							
19	25	126	671		#MATLAB? <== MLEvalString("[y,k] = sort(y)")							
20	17	88	521		#MATLAB? <== MLEvalString("fit = fit(k)")							
21	46	235	1319		#MATLAB? <== MLEvalString("n = size(data,1)")							
22	37	204	1038									
23	15	68	458		6. Use MATLAB's polynomial solving functions for another curve fit.							
24	85	363	2904		#MATLAB? <== MLEvalString("[p,S] = polyfit(1:n,y',5)")							
25	66	300	2006		#MATLAB? <== MLEvalString("newfit = polyval(p,1:n,S)")							
26	39	161	938									
27	111	459	3282		7. Plot curves and add legend							
28	16	80	476		#MATLAB? <== MLEvalString("plot(1:n,y,'bo',1:n,fit,'r',1:n,newfit,'g'); legend('data','fit','newfit')")							

The worksheet contains one named range: A4:C28 is named DATA and contains the data set for this example.

- 2 Make E5 the active cell. Press **F2**; then press **Enter** to execute the Spreadsheet Link EX function that copies the sample data set to the MATLAB workspace. The data set contains 25 observations of three variables. There is a strong linear dependence among the observations; in fact, they are close to being scalar multiples of each other.
- 3 Move to cell E8 and press **F2**; then press **Enter**. Repeat with cells E9 and E10. These Spreadsheet Link EX functions regress the third column of data on the other two columns, and create the following:
 - A single vector y containing the third-column data.
 - A three-column matrix A , that consists of a column of ones followed by the rest of the data.

- 4 Execute the function in cell E13. This function computes the regression coefficients by using the MATLAB back slash (`\`) operation to solve the (overdetermined) system of linear equations, $A \cdot \text{beta} = y$.
- 5 Execute the function in cell E16. MATLAB matrix-vector multiplication produces the regressed result (`fit`).
- 6 Execute the functions in cells E19, E20, and E21. These functions do the following:
 - a Compare the original data with `fit`.
 - b Sort the data in increasing order and apply the same permutation to `fit`.
 - c Create a scalar for the number of observations.
- 7 Execute the functions in cells E24 and E25. Often it is useful to fit a polynomial equation to data. To do so, you would ordinarily have to set up a system of simultaneous linear equations and solve for the coefficients. The MATLAB `polyfit` function automates this procedure, in this case for a fifth-degree polynomial. The `polyval` function then evaluates the resulting polynomial at each data point to check the goodness of fit (`newfit`).
- 8 Execute the function in cell E28. The MATLAB `plot` function graphs the original data (blue circles), the regressed result `fit` (dashed red line), and the polynomial result (solid green line). It also adds a legend.



Since the data is closely correlated but not exactly linearly dependent, the `fit` curve (dashed line) shows a close, but not an exact, fit. The fifth-degree polynomial curve, `newfit`, is a more accurate mathematical model for the data.

When you finish this version of the example, close the figure window.

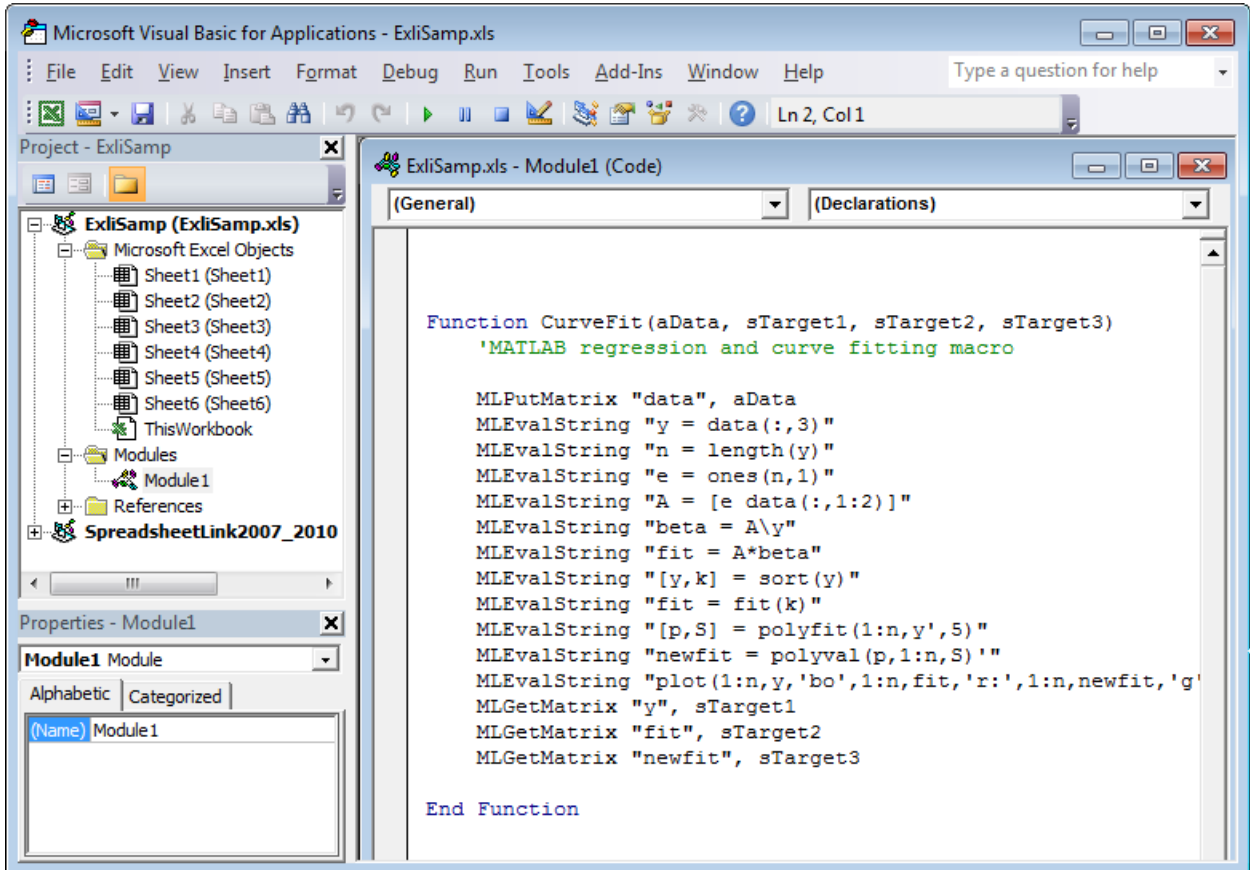
Using Macros

- 1 Click the **Sheet2** tab on Ex1iSamp.xls. The worksheet for this example appears.

	A	B	C	D
1	Regression and Curve Fitting Macro			
2	(See Module 1)			
3				
4			0 <== CurveFit(DATA,"A7","B7","C7")	
5				
6	y	fit	newfit	
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26				
27				
28				
29				
30				
31				


- 2 Make cell A4 the active cell, but do not execute it yet.

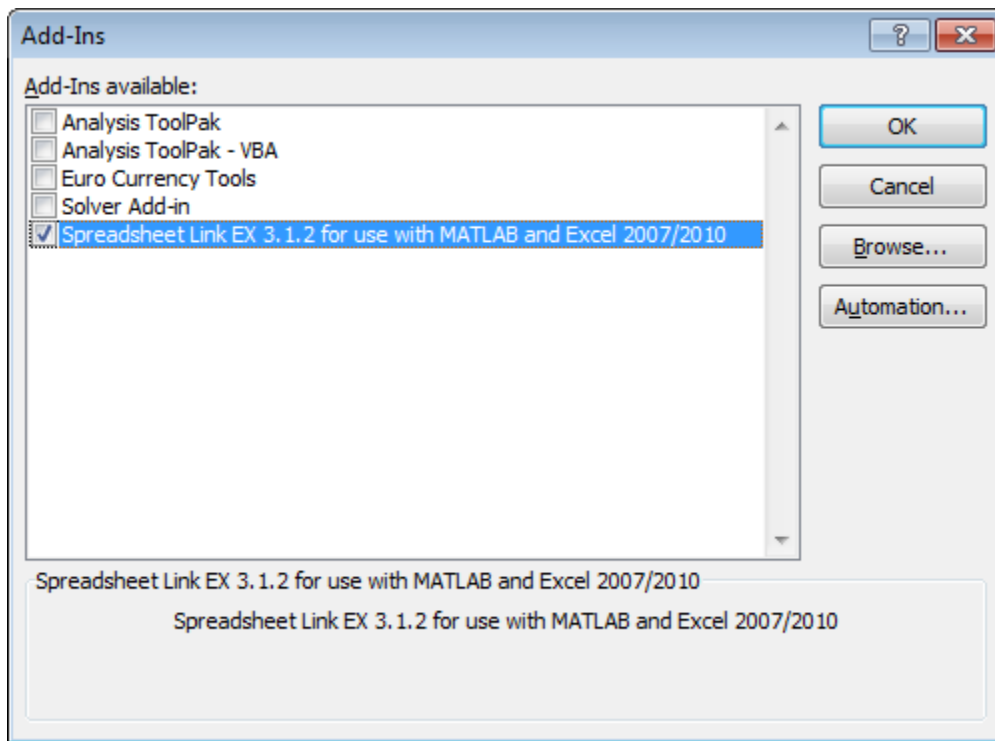
Cell A4 calls the macro CurveFit, which you can examine in the Microsoft Visual Basic environment.



- 3 While this module is open, make sure that the Spreadsheet Link EX add-in is enabled.

- If you are using the Excel 2003 software:
 - Click **Tools > References**.

- b** In the References dialog box, make sure that the `excllink.xla` check box is selected. If not, select it.
- c** Click **OK**.
- If you are using the Excel 2007 software:
 - d** Click the Microsoft Office Button, .
 - e** Click **Options**. The Excel Options pane appears.
 - f** Click **Add-Ins**.
 - g** From the Manage selection list, choose **Excel Add-Ins**.
 - h** Click **Go**. The Add-Ins pane appears.
 - i** Make sure that the **Spreadsheet Link EX for use with MATLAB** check box is selected. If not, select it.



- j** Click **OK** to close the Add-Ins pane.
 - k** Click **OK** to close the Excel Options pane.
- 4** In cell A4 of **Sheet2**, press **F2**; then press **Enter** to execute the CurveFit macro. The macro does the following:
 - a** Runs the same functions as the worksheet example (in a slightly different order), including plotting the graph.
 - b** Calls the `MLGetMatrix` function in the CurveFit macro. This macro copies to the worksheet the original data `y` (sorted), the corresponding regressed data `fit`, and the polynomial data `newfit`.

	A	B	C	D
1	Regression and Curve Fitting Macro			
2	(See Module 1)			
3				
4		0 <= CurveFit(DATA,"A7","B7","C7")		
5				
6	y	fit	newfit	
7	1325	379.0475	402.008	
8	533	430.3099	515.8528	
9	1013	462.4722	549.7114	
10	1163	472.0222	543.0184	
11	5326	501.7971	524.5499	
12	2043	476.7973	513.775	
13	602	467.2472	522.2081	
14	532	570.8968	554.761	
15	543	641.1212	611.0947	
16	1134	743.6461	686.9715	
17	766	767.5211	775.6072	
18	495	773.5589	869.023	
19	635	1143.781	959.3974	
20	913	1279.593	1040.419	
21	1591	1201.219	1108.636	
22	671	1098.695	1164.812	
23	521	1251.081	1215.276	
24	1319	1478.743	1273.275	
25	1038	1163.157	1360.322	
26	458	1479.157	1507.557	
27	2904	2086.177	1757.09	
28	2006	2011.592	2163.358	
29	938	2666.224	2794.475	
30	3282	3483.345	3733.586	
31	476	5197.796	5080.215	

Interpolating Data

Interpolation is a process for estimating values that lie between known data points. It is important for applications such as signal and image processing and data visualization. MATLAB interpolation functions let you balance the smoothness of data fit with execution speed and efficient memory use.

This example is included in the Spreadsheet Link EX product. To run it:

- 1 Start Excel, Spreadsheet Link EX, and MATLAB sessions.
- 2 Navigate to the folder `matlabroot\toolbox\exlink\`.
- 3 Open the file `ExliSamp.xls`
- 4 Execute the example as needed.

This example uses a two-dimensional data-gridding interpolation function on thermodynamic data, where volume has been measured for time and temperature values. It finds the volume values underlying the two-dimensional, time-temperature function for a new set of time and temperature coordinates.

The example uses a Microsoft Excel worksheet to organize and display the original data and the interpolated output data. You use Spreadsheet Link EX functions to copy the data to and from the MATLAB workspace, and then execute the MATLAB interpolation function. Finally, you invoke MATLAB graphics to display the interpolated data in a three-dimensional color surface.

- 1 Click the **Sheet3** tab on `ExliSamp.xls`. The worksheet for this example appears.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Data Interpolation																			
2																				
3	Original Data					Interpolated Values														
4	Time	Temp	Volume																	
5	0.025	68.00	2504.08																	
6	0.050	68.05	2535.07		Time	68.0	68.5	69.0	69.5	70.0	70.5	71.0	71.5	72.0	72.5	73.0	73.5	74.0	74.5	75.0
7	0.075	68.07	2562.91																	
8	0.100	68.09	2575.74																	
9	0.125	68.20	2606.16																	
10	0.150	68.50	2628.58																	
11	0.175	68.85	2681.38																	
12	0.200	69.22	2712.06																	
13	0.225	70.08	2767.52																	
14	0.250	70.33	2815.54																	
15	0.275	70.59	2824.37																	
16	0.300	70.85	2873.65																	
17	0.325	71.11	2882.20																	
18	0.350	71.44	2896.49																	
19	0.375	71.82	2902.07																	
20	0.400	72.33	2920.04																	
21	0.425	72.65	2929.35																	
22	0.450	73.46	2934.23																	
23	0.475	73.85	2938.55																	
24	0.500	74.22	3012.93																	
25	0.525	74.37	3039.12																	
26	0.550	74.55	3130.01																	
27	0.575	74.67	3179.24																	
28	0.600	74.72	3180.71																	
29	0.625	75.00	3184.15																	
30																				
31	Spreadsheet Link EX Functions																			
32	1. Transfer original data to MATLAB.																			
33	#MATLAE <= MLPutMatrix("Labels", A4:C4)																			
34	#MATLAE <= MLPutMatrix("X", A5:A29)																			
35	#MATLAE <= MLPutMatrix("T", B5:B29)																			
36	#MATLAE <= MLPutMatrix("V", C5:C29)																			
37																				
38	2. Transfer interpolation data points to MATLAB.																			
39	#MATLAE <= MLPutMatrix("Xa", E7:E30)																			
40	#MATLAE <= MLPutMatrix("Ta", F6:T6)																			
41																				
42	3. Execute MATLAB data interpolation function.																			
43	#MATLAE <= MLEvalString("[X], Tl, V] = griddata(X,T,V,Xa,Ta,'inwdist')")																			
44																				
45	4. Transpose output data matrix and transfer data to Excel.																			
46	#MATLAE <= MLEvalString("IV = VI;")																			
47	#MATLAE <= MLGetMatrix("IV", sheet3!F7")																			
48																				
49	5. Plot interpolated data and label the figure.																			
50	#MATLAE <= MLEvalString("surf(Xl, Tl, Vl), title('Interpolated Data'); xlabel(Labels{1}); ylabel(Labels{2}); zlabel(Labels{3}); grid on")																			

The worksheet contains the measured thermodynamic data in cells A5:A29, B5:B29, and C5:C29. The time and temperature values for interpolation are in cells E7:E30 and F6:T6, respectively.

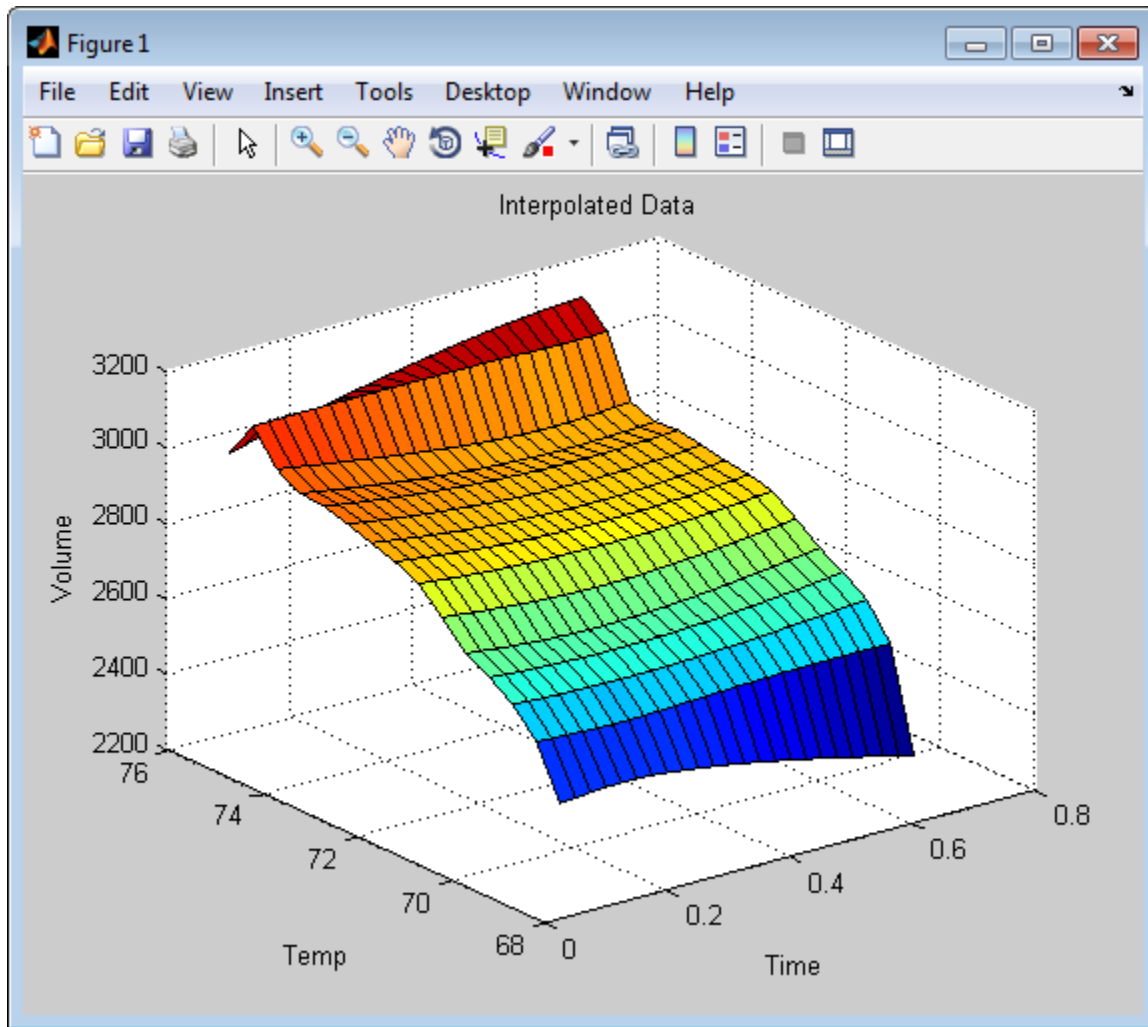
- 2** Make A33 the active cell. Press **F2**; then press **Enter** to execute the Spreadsheet Link EX function that passes the Time, Temp, and Volume labels to the MATLAB workspace.
- 3** Make A34 the active cell. Press **F2**; then press **Enter** to execute the Spreadsheet Link EX function that copies the original time data to the MATLAB workspace. Move to cell A35 and execute the function to copy the

original temperature data. Execute the function in cell A36 to copy the original volume data.

- 4 Move to cell A39 and press **F2**; then press **Enter** to copy the interpolation time values to the MATLAB workspace. Execute the function in cell A40 to copy the interpolation temperature values.
- 5 Execute the function in cell A43. `griddata` is the MATLAB two-dimensional interpolation function that generates the interpolated volume data using the inverse distance method.
- 6 Execute the functions in cells A46 and A47 to transpose the interpolated volume data and copy it to the Excel worksheet. The data fills cells F7:T30, which are enclosed in a border.

Interpolated Values															
	Temp														
Time	68.0	68.5	69.0	69.5	70.0	70.5	71.0	71.5	72.0	72.5	73.0	73.5	74.0	74.5	75.0
0.025	2504.08	2638.15	2707.32	2750.09	2784.91	2851.19	2911.62	2940.67	2961.40	2983.17	3000.06	3006.32	3041.01	3125.78	3026.85
0.05	2507.26	2635.76	2704.79	2746.66	2779.96	2846.35	2907.00	2934.98	2955.07	2976.69	2993.64	2999.35	3034.49	3126.43	3036.68
0.075	2510.83	2633.45	2702.58	2743.62	2775.40	2841.84	2902.75	2929.64	2949.08	2970.51	2987.50	2992.60	3027.98	3126.97	3046.32
0.1	2513.93	2631.34	2700.70	2740.99	2771.27	2837.66	2898.88	2924.66	2943.43	2964.66	2981.67	2986.08	3021.49	3127.39	3055.77
0.125	2515.14	2629.60	2699.17	2738.77	2767.61	2833.83	2895.40	2920.07	2938.14	2959.14	2976.16	2979.83	3015.06	3127.71	3065.02
0.15	2514.31	2628.58	2698.02	2736.99	2764.49	2830.38	2892.31	2915.87	2933.23	2953.97	2970.99	2973.86	3008.70	3127.95	3074.08
0.175	2511.84	2628.88	2697.25	2735.66	2762.00	2827.31	2889.59	2912.08	2928.72	2949.17	2966.17	2968.21	3002.47	3128.11	3082.93
0.2	2508.10	2629.91	2696.87	2734.79	2760.22	2824.68	2887.26	2908.72	2924.62	2944.75	2961.71	2962.89	2996.39	3128.21	3091.57
0.225	2503.37	2631.32	2696.88	2734.37	2759.24	2822.57	2885.29	2905.80	2920.96	2940.73	2957.65	2957.93	2990.50	3128.25	3099.99
0.25	2497.84	2632.93	2697.28	2734.42	2759.10	2821.05	2883.68	2903.34	2917.76	2937.13	2953.97	2953.36	2984.86	3128.24	3108.19
0.275	2491.66	2634.64	2698.05	2734.91	2759.76	2820.23	2882.43	2901.33	2915.02	2933.97	2950.71	2949.20	2979.52	3128.18	3116.14
0.3	2484.92	2636.35	2699.18	2735.85	2761.12	2820.16	2881.55	2899.79	2912.78	2931.26	2947.88	2945.48	2974.53	3128.07	3123.83
0.325	2477.71	2638.00	2700.64	2737.22	2763.09	2820.81	2881.06	2898.72	2911.04	2929.03	2945.47	2942.21	2969.96	3127.90	3131.26
0.35	2470.07	2639.54	2702.41	2739.01	2765.59	2822.11	2880.97	2898.13	2909.82	2927.29	2943.52	2939.43	2965.89	3127.66	3138.38
0.375	2462.06	2640.93	2704.45	2741.19	2768.54	2823.98	2881.29	2898.00	2909.13	2926.05	2942.01	2937.16	2962.39	3127.30	3145.19
0.4	2453.70	2642.15	2706.75	2743.75	2771.89	2826.33	2882.03	2898.34	2908.97	2925.33	2940.96	2935.42	2959.55	3126.79	3151.66
0.425	2445.03	2643.15	2709.26	2746.67	2775.62	2829.13	2883.20	2899.16	2909.34	2925.14	2940.37	2934.25	2957.45	3126.07	3157.75
0.45	2436.07	2643.94	2711.97	2749.92	2779.68	2832.32	2884.78	2900.44	2910.23	2925.48	2940.24	2933.67	2956.16	3125.09	3163.42
0.475	2426.82	2644.48	2714.84	2753.48	2784.06	2835.88	2886.78	2902.19	2911.63	2926.34	2940.57	2933.71	2955.74	3123.85	3168.63
0.5	2417.31	2644.77	2717.84	2757.32	2788.73	2839.78	2889.19	2904.40	2913.52	2927.71	2941.36	2934.34	2956.22	3122.46	3173.31
0.525	2407.54	2644.80	2720.95	2761.44	2793.67	2844.01	2891.99	2907.04	2915.89	2929.57	2942.61	2935.55	2957.60	3121.27	3177.39
0.55	2397.51	2644.56	2724.14	2765.79	2798.87	2848.55	2895.19	2910.11	2918.72	2931.90	2944.30	2937.30	2959.85	3120.88	3180.74
0.575	2387.24	2644.05	2727.39	2770.37	2804.31	2853.38	2898.77	2913.60	2921.99	2934.68	2946.43	2939.57	2962.89	3121.69	3183.21
0.6	2376.71	2643.25	2730.67	2775.14	2809.97	2858.49	2902.71	2917.48	2925.67	2937.89	2948.99	2942.35	2966.66	3123.41	3184.53

- 7 Execute the function in cell A50. The MATLAB software plots and labels the interpolated data on a three-dimensional color surface, with the color proportional to the interpolated volume data.



When you finish the example, close the figure window.

Pricing Stock Options Using the Binomial Model

The Financial Toolbox™ product provides functions that compute prices, sensitivities, and profits for portfolios of options or other equity derivatives. This example uses the binomial model to price an option. The binomial model assumes that the probability of each possible price over time follows a binomial distribution. That is, prices can move to only two values, one up or one down, over any short time period. Plotting these two values over time is known as building a *binomial tree*.

This example organizes and displays input and output data using a Microsoft Excel worksheet. Spreadsheet Link EX functions copy data to a MATLAB matrix, calculate the prices, and return data to the worksheet.

This example is included in the Spreadsheet Link EX product. To run it:

- 1 Start Excel, Spreadsheet Link EX, and MATLAB sessions.
- 2 Navigate to the folder `matlabroot\toolbox\exlink\`.
- 3 Open the file `ExliSamp.xls`
- 4 Execute the example as needed.

Note This example requires Financial Toolbox, Statistics Toolbox™, and Optimization Toolbox™.

- 1 Click the **Sheet4** tab on `ExliSamp.xls` to open the worksheet for this example.

	A	B	C	D	E	F	G	H	I	J	K
1	Binomial Option Pricing										
2											
3		bindata		Spreadsheet Link EX Functions							
4	Asset price, so	\$ 52.00		1. Transfer data to MATLAB.							
5	Option exercise price, x	\$ 50.00		#MATLAB' <== MLPutMatrix("b", bindata)							
6	Risk-free interest rate, r	10%		2. Execute MATLAB Financial Toolbox binomial option pricing function.							
7	Time to maturity, t (yrs)	0.416667	=5/12	#MATLAB' <== MLEvalString("[p, o]=binprice(b(1), b(2), b(3), b(4), b(5), b(6), b(7))")							
8	Time increment, dt	0.083333	=1/12								
9	Volatility, sig	0.4		3. Transfer output data to Excel.							
10	Call (1) or put (0), flag	0		#MATLAB' <== MLGetMatrix("p", "asset_tree")							
11				#MATLAB' <== MLGetMatrix("o", "value_tree")							
12											
13											
14			Start	Period 1	Period 2	Period 3	Period 4	Period 5			
15	Asset price tree, p (\$)										
16											
17											
18											
19											
20											
21											
22											
23	Option value tree, o (\$)										
24											
25											
26											
27											
28											

The worksheet contains three named ranges:

- B4:B10 named bindata. Two cells in bindata contain formulas:
 - B7 contains =5/12
 - B8 contains =1/12
- B15 named asset_tree.
- B23 named value_tree.

2 Make D5 the active cell. Press **F2**; then press **Enter** to execute the Spreadsheet Link EX function that copies the asset data to the MATLAB workspace.

3 Move to D8 and execute the function that computes the binomial prices.

- 4 Execute the functions in D11 and D12 to copy the price data to the Excel worksheet.

The worksheet looks as follows.

	A	B	C	D	E	F	G	H	I	J	K
1	Binomial Option Pricing										
2											
3		bindata		Spreadsheet Link EX Functions							
4	Asset price, so	\$ 52.00		1. Transfer data to MATLAB.							
5	Option exercise price, x	\$ 50.00		0 <== MLPutMatrix("b", bindata)							
6	Risk-free interest rate, r	10%									
7	Time to maturity, t (yrs)	0.416667	=5/12	2. Execute MATLAB Financial Toolbox binomial option pricing function.							
8	Time increment, dt	0.083333	=1/12	0 <== MLEvalString("[p, o]=binprice(b(1), b(2), b(3), b(4), b(5), b(6), b(7))")							
9	Volatility, sig	0.4									
10	Call (1) or put (0), flag	0		3. Transfer output data to Excel.							
11				0 <== MLGetMatrix("p", "asset_tree")							
12				0 <== MLGetMatrix("o", "value_tree")							
13											
14		Start	Period 1	Period 2	Period 3	Period 4	Period 5				
15	Asset price tree, p (\$)	52.000	58.365	65.509	73.527	82.527	92.628				
16		0	46.329	52.000	58.365	65.509	73.527				
17		0	0	41.277	46.329	52.000	58.365				
18		0	0	0	36.776	41.277	46.329				
19		0	0	0	0	32.765	36.776				
20		0	0	0	0	0	29.192				
21											
22											
23	Option value tree, o (\$)	3.728	1.664	0.428	0	0	0				
24		0	5.918	2.964	0.876	0	0				
25		0	0	9.060	5.164	1.793	0				
26		0	0	0	13.224	8.723	3.671				
27		0	0	0	0	17.235	13.224				
28		0	0	0	0	0	20.808				

Read the asset price tree as follows:

- Period 1 shows the up and down prices.
- Period 2 shows the up-up, up-down, and down-down prices.
- Period 3 shows the up-up-up, up-up, down-down, and down-down-down prices.
- And so on.

Ignore the zeros. The option value tree gives the associated option value for each node in the price tree. The option value is zero for prices significantly above the exercise price. Ignore the zeros that correspond to a zero in the price tree.

- 5 Try changing the data in B4:B10, and then executing the Spreadsheet Link EX functions again.

Note If you increase the time to maturity (B7) or change the time increment (B8), you may need to enlarge the output tree areas.

- 6 When you finish the example, close the figure window.

Calculating and Plotting the Efficient Frontier of Financial Portfolios

MATLAB and Financial Toolbox functions compute and plot risks, variances, rates of return, and the efficient frontier of portfolios. Efficient portfolios have the lowest aggregate variance, or risk, for a given return. Microsoft Excel and the Spreadsheet Link EX software let you set up data, execute financial functions and MATLAB graphics, and display numeric results.

This example analyzes three portfolios, using rates of return for six time periods. In actual practice, these functions can analyze many portfolios over many time periods, limited only by the amount of computer memory available.

This example is included in the Spreadsheet Link EX product. To run it:

- 1 Start Excel, Spreadsheet Link EX, and MATLAB sessions.
- 2 Navigate to the folder `matlabroot\toolbox\exlink\`.
- 3 Open the file `ExliSamp.xls`
- 4 Execute the example as needed.

Note This example requires Financial Toolbox, Statistics Toolbox, and Optimization Toolbox.

- 1 Click the **Sheet5** tab on `ExliSamp.xls`. The worksheet for this example appears.

	A	B	C	D	E	F	G	H	I	J
1	Portfolio Efficient Frontier									
2										
3	Rates of return	Global	Corp. Bnd	Small Cap		Risk	ROR	Global Weights	Corp. Bnd	Small Cap
4	Nov-91	7.125%	4.125%	8.375%						
5	Nov-92	5.125%	5.125%	3.875%						
6	Nov-93	-1.375%	5.750%	10.500%						
7	Nov-94	7.750%	6.000%	14.750%						
8	Nov-95	8.250%	6.375%	-3.625%						
9	Nov-96	12.625%	6.125%	9.125%						
10										
11										
12										
13	Spreadsheet Link EX Functions									
14	1. Transfer data to MATLAB.									
15	#MATLAB?	<== MLPutMatrix("Labels", F3:G3)								
16	#MATLAB?	<== MLPutMatrix("retseries", B4:D9)								
17										
18	2. Execute MATLAB Financial Toolbox functions.									
19	#MATLAB?	<== MLEvalString("[ret, cov] = ewstats(retseries)")								
20	#MATLAB?	<== MLEvalString("[risk, ror, weights] = portopt(ret, cov, 20)")								
21										
22	3. Transfer output data to Excel.									
23	#MATLAB?	<== MLGetMatrix("risk", "sheet5!F4")								
24	#MATLAB?	<== MLGetMatrix("ror", "sheet5!G4")								
25	#MATLAB?	<== MLGetMatrix("weights", "sheet5!H4")								
26										
27	4. Plot efficient frontier data and label the figure.									
28	#MATLAB?	<== MLEvalString("portopt(ret, cov, 20); grid on; xlabel(Labels{1}); ylabel(Labels{2})")								

2 Make A15 the active cell. Press **F2**; then press **Enter**. The Spreadsheet Link EX function transfers the labels that describe the output that the MATLAB software computes.

3 Make A16 the active cell to copy the portfolio return data to the MATLAB workspace.

4 Execute the functions in A19 and A20 to compute the Financial Toolbox efficient frontier function for 20 points along the frontier.

5 Execute the Spreadsheet Link EX functions in A23, A24, and A25 to copy the output data to the Excel worksheet.

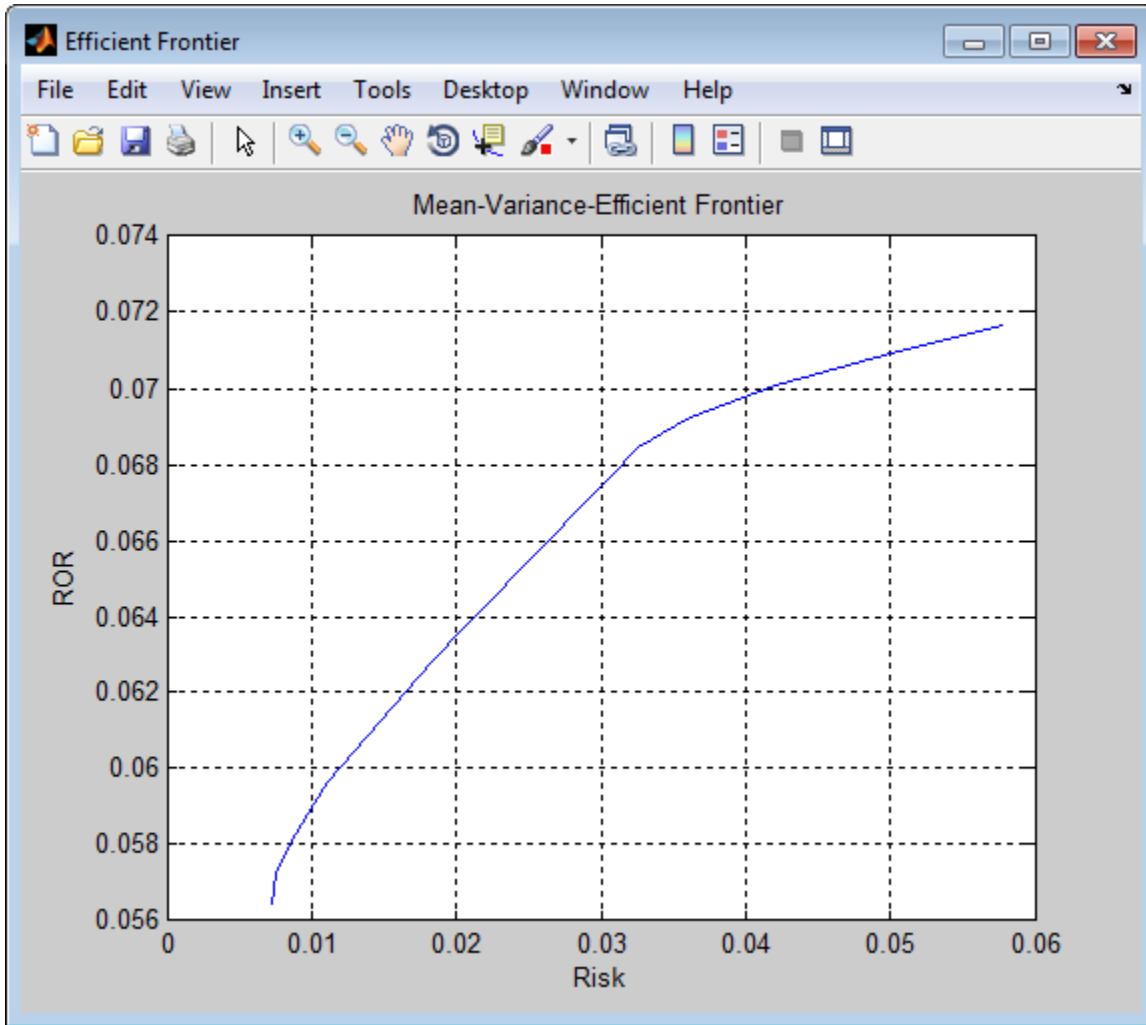
The worksheet looks as follows.

	A	B	C	D	E	F	G	H	I	J
1	Portfolio Efficient Frontier									
2										
3	Rates of return	Global	Corp. Bnd	Small Cap		Risk	ROR	Global Weights	Corp. Bnd	Small Cap
4	Nov-91	7.125%	4.125%	8.375%		0.730%	5.643%	0.3%	96.1%	3.5%
5	Nov-92	5.125%	5.125%	3.875%		0.760%	5.723%	4.0%	89.7%	6.3%
6	Nov-93	-1.375%	5.750%	10.500%		0.844%	5.803%	7.7%	83.3%	9.0%
7	Nov-94	7.750%	6.000%	14.750%		0.968%	5.883%	11.3%	76.9%	11.8%
8	Nov-95	8.250%	6.375%	-3.625%		1.118%	5.964%	15.0%	70.5%	14.5%
9	Nov-96	12.625%	6.125%	9.125%		1.287%	6.044%	18.7%	64.0%	17.3%
10						1.466%	6.124%	22.3%	57.6%	20.0%
11						1.653%	6.204%	26.0%	51.2%	22.8%
12						1.846%	6.284%	29.7%	44.8%	25.5%
13	Spreadsheet Link EX Functions					2.042%	6.365%	33.3%	38.4%	28.3%
14	1. Transfer data to MATLAB.									
15		0	<== MLPutMatrix("Labels", F3:G3)							
16		0	<== MLPutMatrix("retseries", B4:D9)							
17						2.646%	6.605%	44.3%	19.1%	36.6%
18	2. Execute MATLAB Financial Toolbox functions.									
19		0	<== MLEvalString("[ret, cov] = ewstats(retseries)")							
20		0	<== MLEvalString("[risk, ror, weights] = portopt(ret, cov, 20)")							
21						3.055%	6.766%	51.6%	6.3%	42.1%
22	3. Transfer output data to Excel.									
23		0	<== MLGetMatrix("risk", "sheet5IF4")							
24		0	<== MLGetMatrix("ror", "sheet5IG4")							
25		0	<== MLGetMatrix("weights", "sheet5IH4")							
26						4.955%	7.086%	13.8%	0.0%	86.2%
27	4. Plot efficient frontier data and label the figure.									
28	#MATLAB?	<== MLEvalString("portopt(ret, cov, 20); grid on; xlabel(Labels{1}); ylabel(Labels{2})")								

The data describes the efficient frontier for these three portfolios: that set of points representing the highest rate of return (ROR) for a given risk. For each of the 20 points along the frontier, the weighted investment in each portfolio (Weights) would achieve that rate of return.

- Now move to A28 and press **F2**; then press **Enter** to execute the Financial Toolbox function that plots the efficient frontier for the same portfolio data.

The following figure appears.



The light blue line shows the efficient frontier. Note the change in slope above a 6.8% return because the Corporate Bond portfolio no longer contributes to the efficient frontier.

- 7 To try running this example using different data, close the figure window and change the data in cells B4:D9. Then execute all the Spreadsheet Link

EX functions again. The worksheet then shows the new frontier data, and the MATLAB software displays a new efficient frontier graph.

When you finish this example, close the figure window.

Mapping Time and Bond Cash Flows

This example shows how to use the Financial Toolbox and Spreadsheet Link EX software to compute a set of cash flow amounts and dates, given a portfolio of five bonds with known maturity dates and coupon rates. It is included in the Spreadsheet Link EX product. To run it:

- 1** Start Excel, Spreadsheet Link EX, and MATLAB sessions.
- 2** Navigate to the folder *matlabroot\toolbox\exlink*.
- 3** Open the file *ExliSamp.xls*
- 4** Execute the example as needed.

Note This example requires Financial Toolbox, Statistics Toolbox, and Optimization Toolbox.

- 1** Click the **Sheet6** tab on *ExliSamp.xls*. The worksheet for this example appears.

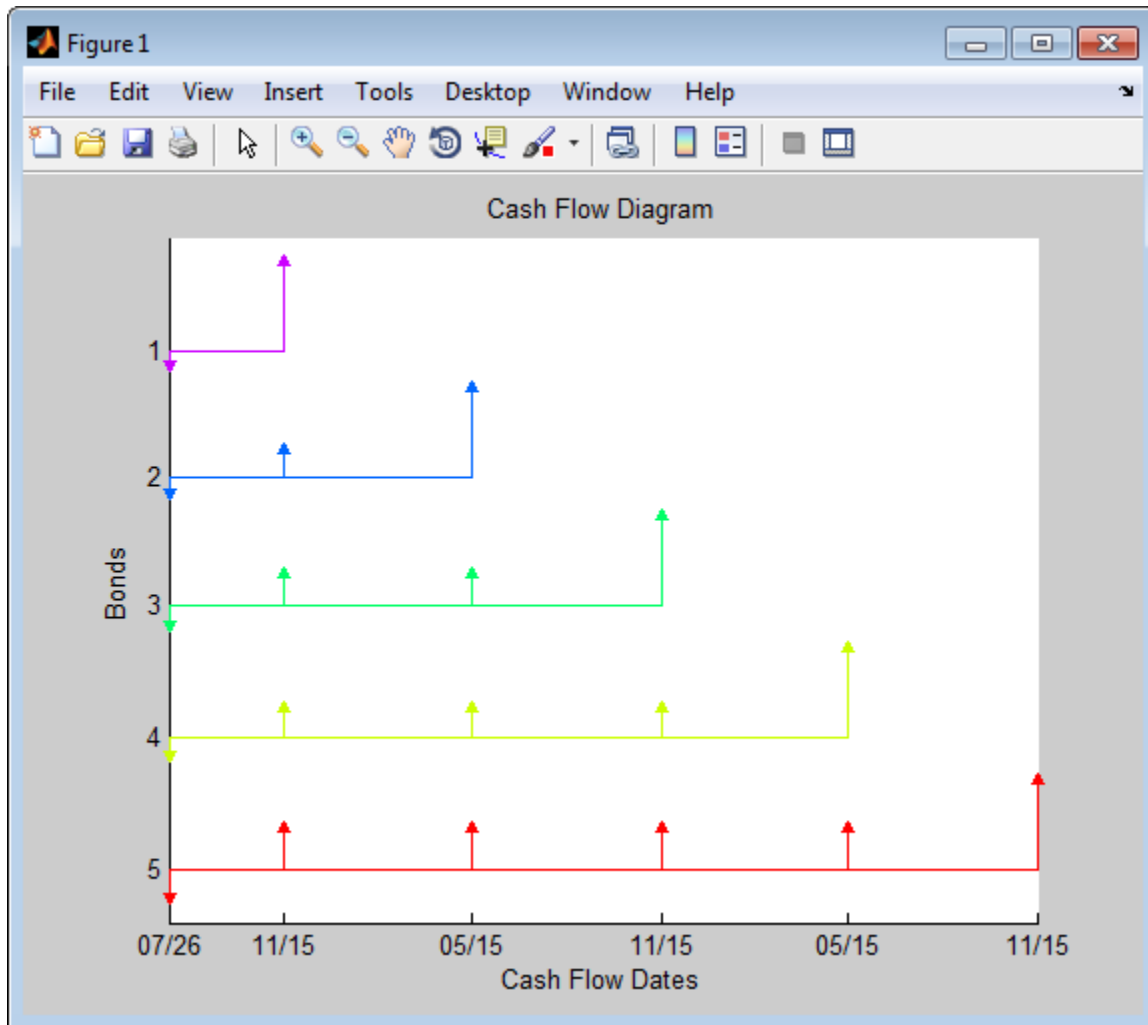
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
1	Cash Flow and Time Mapping for a Portfolio of Bonds														
2											Cash Flow Dates				
3	Settlement Date		26-Jul-99					Bond1							
4							Bond2								
5		Bond Data					Bond3								
6							Bond4								
7		Maturity	Coupon Rate				Bond5								
8	Bond1	15-Nov-99	0.05875						Cash Flow Amounts						
9	Bond2	15-May-00	0.06375												
10	Bond3	15-Nov-00	0.08500												
11	Bond4	15-May-01	0.08000												
12	Bond5	15-Nov-01	0.15750												
13															
14								Bond1							
15							Bond2								
16	Spreadsheet Link EX Functions							Bond3							
17	1. Transfer data to MATLAB.							Bond4							
18	#MATLAB' <== MLPutMatrix("maturity", 'Maturity')							Bond5							
19	#MATLAB' <== MLPutMatrix("cpnrate", 'CpnRate')														
20	#MATLAB' <== MLPutMatrix("sd", C3)														
21															
22	2. Execute MATLAB Financial Toolbox Cash flow and Time mapping function.														
23	#MATLAB' <== MLEvalString("md = x2mdate(maturity,0); sdm = x2mdate(sd,0)")														
24	#MATLAB' <== MLEvalString("[cfa, cfd] = cfamounts(cpnrate, sdm, md, 2)")														
25															
26	3. Transform date numbers to string cell array.														
27	#MATLAB' <== MLEvalString("i = find(isnan(cfd)); zcfd = cfd; zcfd(i) = 0; scfd=datestr(zcfd,2);")														
28	#MATLAB' <== MLEvalString("ccfd = num2cell(scfd,2); ccf(i) = {'N/A'}; ccfd = reshape(ccfd, size(cfd));")														
29	#MATLAB' <== MLEvalString("ccfa = cfa; ccfa(i) = 0; alldates = ccfd(end, :);")														
30															
31	4. Transfer output data to Excel.														
32	#MATLAB' <== MLGetMatrix("ccfd", "sheet6!i3")														
33	#MATLAB' <== MLGetMatrix("alldates", "sheet6!i13")														
34	#MATLAB' <== MLGetMatrix("ccfa", "sheet6!i14")														
35															
36	5. Plot the cash flow diagram.														
37	#MATLAB' <== MLEvalString("cfplot(cfd, cfa); dtaxis('x',6, sdm,50);title('Cash Flow Diagram');xlabel('Cash Flow Dates');ylabel('Bonds');")														

- 2** Make A18 the active cell. Press **F2**, then **Enter** to execute the Spreadsheet Link EX function that transfers the column vector **Maturity** to the MATLAB workspace.
- 3** Make A19 the active cell to transfer the column vector **Coupon Rate** to the MATLAB workspace.
- 4** Make A20 the active cell to transfer the settlement date to the MATLAB workspace.
- 5** Execute the functions in cells A23 and A24 to enable the Financial Toolbox software to compute cash flow amounts and dates.

- 6 Now execute the functions in cells A27 through A29 to transform the dates into string form contained in a cell array.
- 7 Execute the functions in cells A32 through A34 to transfer the data to the Excel worksheet.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Cash Flow and Time Mapping for a Portfolio of Bonds													
2											Cash Flow Dates			
3	Settlement Date		26-Jul-99					Bond1	7/26/1999	11/15/1999	N/A	N/A	N/A	N/A
4								Bond2	7/26/1999	11/15/1999	5/15/2000	N/A	N/A	N/A
5			Bond Data					Bond3	7/26/1999	11/15/1999	5/15/2000	11/15/2000	N/A	N/A
6								Bond4	7/26/1999	11/15/1999	5/15/2000	11/15/2000	5/15/2001	N/A
7			Maturity	Coupon Rate				Bond5	7/26/1999	11/15/1999	5/15/2000	11/15/2000	5/15/2001	11/15/2001
8	Bond1	15-Nov-99	0.05875											
9	Bond2	15-May-00	0.06375											
10	Bond3	15-Nov-00	0.08500											
11	Bond4	15-May-01	0.08000											
12	Bond5	15-Nov-01	0.15750											
13									Cash Flow Amounts					
14								Bond1	-1.1495	102.9375	0	0	0	0
15								Bond2	-1.2473	3.1875	103.1875	0	0	0
16	Spreadsheet Link EX Functions													
17	1. Transfer data to MATLAB.													
18	0	<== MLPutMatrix("maturity","Maturity")												
19	0	<== MLPutMatrix("cpnrate","CpnRate")												
20	0	<== MLPutMatrix("sd",C3)												
21														
22	2. Execute MATLAB Financial Toolbox Cash flow and Time mapping function.													
23	0	<== MLEvalString("md = x2mdate(maturity,0); sdm = x2mdate(sd,0)")												
24	0	<== MLEvalString("[cfa, cfd] = cfamounts(cpnrate, sdm, md, 2)")												
25														
26	3. Transform date numbers to string cell array.													
27	0	<== MLEvalString("i = find(isnan(cfd)); zcfd = cfd; zcfd(i) = 0; scfd=datestr(zcfd,2);")												
28	0	<== MLEvalString("ccfd = num2cell(scfd,2); ccfd(i) = {'N/A'}; ccfd = reshape(ccfd, size(cfd));")												
29	0	<== MLEvalString("ccfa = cfa; ccfa(i) = 0; alldates = ccfd(end, :);")												
30														
31	4. Transfer output data to Excel.													
32	0	<== MLGetMatrix("ccfd", "sheet6li3")												
33	0	<== MLGetMatrix("alldates", "sheet6li13")												
34	0	<== MLGetMatrix("ccfa", "sheet6li14")												
35														
36	5. Plot the cash flow diagram.													
37	#MATLAB	<== MLEvalString("cfplot(cfd, cfa); dtaxis('x',6,sdm,50);title('Cash Flow Diagram');xlabel('Cash Flow Dates');ylabel('Bonds');")												

- 8 Finally, execute the function in cell A37 to display a plot of the cash flows for each portfolio item.



9 When you finish the example, close the figure window.

Error Messages and Troubleshooting

- “Worksheet Cell Errors” on page 3-2
- “Microsoft® Excel Errors” on page 3-5
- “Data Errors” on page 3-8
- “Startup Errors” on page 3-10
- “Audible Error Signals” on page 3-10

Worksheet Cell Errors

You may see these error messages displayed in a worksheet cell.

The first column of the following table contains worksheet cell error messages. The error messages begin with the number sign (#). Most end with an exclamation point (!) or with a question mark (?).

Worksheet Cell Error Messages

Worksheet Cell Error Message	Meaning	Solution
#COLS>#MAXCOLS!	Your MATLAB variable exceeds the Microsoft Excel limit of #MAXCOLS! columns.	This is a limitation in the Excel product. Try the computation with a variable containing fewer columns.
#COMMAND!	The MATLAB software does not recognize the command in an MLEvalString function. The command may be misspelled.	Verify the spelling of the MATLAB command. Correct typing errors.
#DIMENSION!	You used MLAppendMatrix and the dimensions of the appended data do not match the dimensions of the matrix you want to append.	Verify the matrix dimensions and the appended data dimensions, and correct the argument. For more information, see the MLAppendMatrix reference page.
#INVALIDNAME!	You entered an illegal variable name.	Make sure to use legal MATLAB variable names. MATLAB variable names must start with a letter followed by up to 30 letters, digits, or underscores.
#INVALIDTYPE!	You have specified an illegal MATLAB data type with MLGetVar or MLGetMatrix.	For a list of supported MATLAB data types, see “Classes (Data Types)” in the MATLAB Programming Fundamentals documentation.

Worksheet Cell Error Messages (Continued)

Worksheet Cell Error Message	Meaning	Solution
#MATLAB?	You used a Spreadsheet Link EX function and no MATLAB software session is running.	Start the Spreadsheet Link EX and MATLAB software. See “Starting and Stopping the Spreadsheet Link EX Software” on page 1-17.
#NAME?	The function name is unrecognized. The <code>excllink.xla</code> add-in is not loaded, or the function name may be misspelled.	Be sure the <code>excllink.xla</code> add-in is loaded. See “Configuring the Spreadsheet Link EX Software” on page 1-7. Check the spelling of the function name. Correct typing errors.
#NONEXIST!	You referenced a nonexistent matrix in an <code>MLGetMatrix</code> or <code>MLDeleteMatrix</code> function. The matrix name may be misspelled.	Verify the spelling of the MATLAB matrix. Use the MATLAB <code>whos</code> command to display existing matrices. Correct typing errors.
#ROWS>#MAXROWS!	Your MATLAB variable exceeds the Excel limit of #MAXROWS! rows.	This is a limitation in the Excel product. Try the computation with a variable containing fewer rows.
#SYNTAX?	You entered a Spreadsheet Link EX function with incorrect syntax. For example, you did not specify double quotation marks (") , or you specified single quotation marks (') instead of double quotation marks.	Verify and correct the function syntax. For more information, see Chapter 5, “Functions — Alphabetical List”.

Worksheet Cell Error Messages (Continued)

Worksheet Cell Error Message	Meaning	Solution
#VALUE!	An argument is missing from a function, or a function argument is the wrong type.	Supply the correct number of function arguments, of the correct type.
#VALUE!	A macro subroutine uses <code>MLGetMatrix</code> followed by <code>MatlabRequest</code> , which is correct standard usage. A macro function calls that subroutine, and you execute that function from a worksheet cell. The function works correctly, but this message appears in the cell.	Since the function works correctly, ignore the message. Or, in this special case, remove <code>MatlabRequest</code> from the subroutine.

Note When you open an Excel worksheet that contains Spreadsheet Link EX functions, the Excel software tries to execute the functions from the bottom up and right to left. Excel may generate cell error messages such as `#COMMAND!` or `#NONEXIST!`. This is expected behavior. Do the following:

- 1** Ignore the messages.
 - 2** Close MATLAB figure windows.
 - 3** Reexecute the cell functions one at a time in the correct order by pressing **F2**, and then **Enter**.
-

Microsoft Excel Errors

The Excel software may display one of the following error messages.

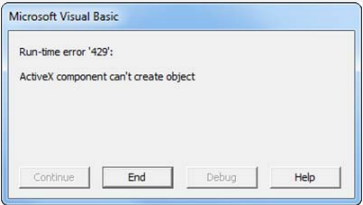
Excel Error Messages

Error Message	Cause of Error	Solution
Error in formula	You entered a formula incorrectly. Common errors include a space between the function name and the left parenthesis; or missing, extra, or mismatched parentheses.	Check entry and correct typing errors.
Can't find project or library	You tried to execute a macro and the location of <code>excllink.xla</code> is incorrect.	Click OK . The References window opens. Remove the check from MISSING: excllink.xla . Find <code>excllink.xla</code> in its correct location, select its check box in the References window, and click OK .
Run-time error '1004': Cells method of Application class failed	You used <code>MLGetMatrix</code> and the matrix is larger than the space available in the worksheet. This error destabilizes the Spreadsheet Link EX software session and changes worksheet calculation mode to manual.	Click OK . Reset worksheet calculation mode to automatic, and save your worksheet as needed. Restart the Excel, Spreadsheet Link EX, and MATLAB software sessions.

Excel Error Messages (Continued)

Error Message	Cause of Error	Solution
MATLAB failed to check out a license of Spreadsheet Link EX or does not have a valid installation of Spreadsheet Link EX	You entered an invalid license passcode or did not install Spreadsheet Link EX properly.	Check that you entered the license passcode properly. Reinstall the Spreadsheet Link EX add-on. (See “Installing the Spreadsheet Link EX Software” on page 1-4.) If you followed the installation guidelines, used a proper passcode and you are still unable to start the Spreadsheet Link EX software, contact your MathWorks representative.
Datasource: Excel; prompt for user name and password	This message appears when an attempt to connect to the Excel software from the Database Toolbox™ software fails.	Make sure that the Excel spreadsheet referenced by the data source exists, then retry the connection.

Excel Error Message Boxes

Error Message Box	Cause of Error	Solution
	<p>This error appears when you start the automation server from the Excel interface, and multiple versions of the MATLAB software are installed on your desktop.</p>	<p>To correct this error, perform the following:</p> <ol style="list-style-type: none"> 1 Shut down all MATLAB and Excel instances. 2 Open a Command Prompt window, and using <code>cd</code>, change to the <code>bin\win32</code> subfolder of the MATLAB installation folder. 3 Type the command: <pre>.\matlab /regserver</pre> 4 When the MATLAB session starts, close it. Using <code>/regserver</code> fixes the registry entries. 5 Start an Excel session. The Spreadsheet Link EX add-in now loads properly. 6 Verify that the Spreadsheet Link EX software is working by entering the following command from the MATLAB Command Window: <pre>a = 3.14159</pre> 7 Enter the following formula in cell A1 of the open Excel worksheet: <pre>=mlgetmatrix("a", "a1")</pre> 8 The value 3.14159 appears in cell A1.

Data Errors

In this section...
“Matrix Data Errors” on page 3-8
“Errors When Opening Saved Worksheets” on page 3-8

Matrix Data Errors

Data in the MATLAB or Microsoft Excel workspaces may produce the following errors.

Data Errors

Data Error	Cause	Solution
MATLAB matrix cells contain zeros (0).	Corresponding Excel worksheet cells are empty.	Excel worksheet cells must contain only numeric or string data.
MATLAB matrix is a 1-by-1 zero matrix.	You used quotation marks around the data-location argument in <code>MLPutMatrix</code> or <code>MLAppendMatrix</code> .	Correct the syntax to remove quotation marks.
MATLAB matrix is empty ([]).	You referenced a nonexistent VBA variable in <code>MLPutVar</code> .	Correct the macro; you may have typed the variable name incorrectly.
VBA matrix is empty.	You referenced a nonexistent MATLAB variable in <code>MLGetVar</code> .	Correct the macro; you may have typed the variable name incorrectly.

Errors When Opening Saved Worksheets

This section describes errors that you may encounter when opening saved worksheets.

- When you open an Excel worksheet that contains Spreadsheet Link EX functions, the Excel software tries to execute the functions from the bottom

up and right to left. Excel may generate cell error messages such as #COMMAND! or #NONEXIST!. This is expected behavior. Do the following:

- 1 Ignore the messages.
 - 2 Close MATLAB figure windows.
 - 3 Reexecute the cell functions one at a time in the correct order by pressing **F2**, and then **Enter**.
- If you save an Excel worksheet containing Spreadsheet Link EX functions, and then reopen it in an environment where the `excllink.xla` add-in is in a different location, you may see the message: This document contains links: Re-establish links?

To address this issue, do the following:

- 1 Click **No**.
- 2 Select **Edit > Links**.
- 3 In the Links dialog box, click **Change Source**.
- 4 In the Change Links dialog box, select `matlabroot\toolbox\exlink\excllink.xla`.
- 5 Click **OK**.

The Excel software executes each function as it changes its link. You may see MATLAB figure windows and hear error beeps as the links change and functions execute; ignore them.

- 6 In the Links dialog box, click **OK**.

The worksheet now connects to the Spreadsheet Link EX add-in.

Or, instead of using the **Links** menu, you can manually edit the link location in each affected worksheet cell to show the correct location of `excllink.xla`.

Startup Errors

If you have enabled `MLAutoStart`, double-clicking an `xls` file in the MATLAB Current Folder browser and choosing **Open Outside MATLAB** causes a Microsoft Excel error to appear. To open the file successfully, click **End** in the error window.

To avoid this issue, disable `MLAutoStart`. Start MATLAB sessions from the Excel interface by clicking the **startmatlab** button in the Excel menu bar.

Audible Error Signals

You may hear audible errors while passing data to the MATLAB workspace using `MLPutMatrix` or `MLAppendMatrix`. These errors usually indicate that you have insufficient computer memory to carry out the operation. Close other applications or clear unnecessary variables from the MATLAB workspace and try again. If the error signal reoccurs, you probably have insufficient physical memory in your computer for this operation.

Function Reference

Link Management (p. 4-2)

Work with link management
functions

Data Management (p. 4-3)

Work with data management
functions

Link Management

<code>matlabinit</code>	Initialize Spreadsheet Link EX software and start MATLAB process
<code>MLAutoStart</code>	Automatically start MATLAB process
<code>MLClose</code>	Stop MATLAB process
<code>MLOpen</code>	Start MATLAB process
<code>MLUseCellArray</code>	Toggle <code>MLPutMatrix</code> to use MATLAB cell arrays

Data Management

<code>matlabfcn</code>	Evaluate MATLAB command given Microsoft Excel data
<code>matlabsub</code>	Evaluate MATLAB command given Microsoft Excel data and designate output location
<code>MLAppendMatrix</code>	Create or append MATLAB matrix with data from Microsoft Excel worksheet
<code>MLDeleteMatrix</code>	Delete MATLAB matrix
<code>MLEvalString</code>	Evaluate command in MATLAB software
<code>MLGetFigure</code>	Import current MATLAB figure into Microsoft Excel spreadsheet
<code>MLGetMatrix</code>	Write contents of MATLAB matrix to Microsoft Excel worksheet
<code>MLGetVar</code>	Write contents of MATLAB matrix in Microsoft Excel VBA variable
<code>MLMissingDataAsNaN</code>	Set empty cells to NaN or 0
<code>MLPutMatrix</code>	Create or overwrite MATLAB matrix with data from Microsoft Excel worksheet
<code>MLPutVar</code>	Create or overwrite MATLAB matrix with data from Microsoft Excel VBA variable
<code>MLShowMatlabErrors</code>	Return standard Spreadsheet Link EX errors or full MATLAB errors using <code>MLEvalString</code>

`MLStartDir`

Specify MATLAB current working folder after startup

`MLUseFullDesktop`

Specify whether to use full MATLAB desktop or MATLAB Command Window

Functions — Alphabetical List

matlabfcn

Purpose

Evaluate MATLAB command given Microsoft Excel data

Syntax

Worksheet: `matlabfcn(command, inputs)`
command MATLAB command to evaluate. Embed the command in double quotation marks; for example, "command".
inputs Variable length input argument list passed to a MATLAB command. The argument list may contain a range of worksheet cells that contain input data.

Description

Passes the command to the MATLAB workspace for evaluation, given the function input data. The function returns a single value or string depending upon the MATLAB output. The result is returned to the calling worksheet cell. This function is intended for use as an Excel worksheet function.

Examples

- 1 Add the data in worksheet cells B1 through B10, and then return the sum to the active worksheet cell:

```
matlabfcn("sum", B1:B10)
```

- 2 Plot the data in worksheet cells B1 through B10, using x as the marker type:

```
matlabfcn("plot", B1:B10, "x")
```

See Also

matlabsub

Purpose Initialize Spreadsheet Link EX software and start MATLAB process

Syntax matlabinit

Note To run matlabinit from the Microsoft Excel toolbar, click **Tools > Macro**. In the **Macro Name/Reference** box, enter matlabinit and click **Run**. Alternatively, you could include this function in a macro subroutine. You cannot run matlabinit as a worksheet cell formula or in a macro function.

Description Initializes the Spreadsheet Link EX software and starts MATLAB process. If the Spreadsheet Link EX software has been initialized and the MATLAB software is running, subsequent invocations do nothing. Use matlabinit to start Spreadsheet Link EX and MATLAB sessions manually when you have set MLAutoStart to no. If you set MLAutoStart to yes, matlabinit executes automatically.

See Also MLAutoStart | MLOpen

matlabsub

Purpose

Evaluate MATLAB command given Microsoft Excel data and designate output location

Syntax

Worksheet:	<code>matlabsub(command, edat, inputs)</code>
command	MATLAB command to evaluate. Enter the MATLAB command in double quotation marks, as "command".
edat	Worksheet location where the function writes the returned data. "edat" (in quotation marks) directly specifies the location and it must be a cell address or a range name. edat (without quotation marks) is an indirect reference: the function evaluates the contents of edat to get the location. edat must be a worksheet cell address or range name. Although you can specify a range of output cells, matlabsub does not support multiple outputs. Instead of returning multiple outputs, matlabsub returns the first output starting in the first cell from the specified range, and discards all other outputs.
inputs	Variable length input argument list passed to MATLAB command. This argument list can contain a range of worksheet cells that contain input data.

Description

Passes the specified command to the MATLAB workspace for evaluation, given the function input data. The function returns a single value or string depending upon the MATLAB output. This function is intended for use as an Excel worksheet function.

To return an array of data to the Microsoft Excel Visual Basic for Applications (VBA) workspace, see `MLEvalString` and `MLGetVar`.

Caution edat must not include the cell that contains the matlabsub function. In other words, be careful not to overwrite the function itself. Also make sure there is enough room in the worksheet to write the matrix contents. If there is insufficient room, the function generates a fatal error.

Examples

Sum the data in worksheet cells B1 through B10, and then return the output to cell A1:

```
matlabsub("sum", "A1", B1:B10)
```

See Also

matlabfcn

MLAppendMatrix

Purpose

Create or append MATLAB matrix with data from Microsoft Excel worksheet

Syntax

Worksheet: MLAppendMatrix(var_name, mdat)
Macro: MLAppendMatrix var_name, mdat
var_name Name of MATLAB matrix to which to append data. "var_name" (in quotation marks) directly specifies the matrix name. var_name (without quotation marks) is an indirect reference: the function evaluates the contents of var_name to get the matrix name, and var_name must be a worksheet cell address or range name

mdat Location of data to append to var_name. mdat (no quotation marks). Must be a worksheet cell address or range name.

If this argument is not initially an Excel Range data type and you call the function from a worksheet, MLAppendMatrix performs the necessary type coercion.

If this argument is not an Excel Range data type and you call the function from within a Microsoft Visual Basic macro, the call fails. The error message ByRef Argument Type Mismatch appears.

Description

Appends data in mdat to MATLAB matrix var_name. Creates var_name if it does not exist. The function checks the dimensions of var_name and mdat to determine how to append mdat to var_name. If the dimensions allow appending mdat as either new rows or new columns, it appends mdat to var_name as new rows. If the dimensions do not match, the function returns an error. mdat must contain either numeric data or string data. Data types cannot be combined within the range specified

in `mdat`. Empty `mdat` cells become MATLAB matrix elements containing zero if the data is numeric, and empty strings if the data is a string.

Examples

Example 1: Append data from a worksheet cell range to a MATLAB matrix

In this example, `B` is a 2-by-2 MATLAB matrix. Append the data in worksheet cell range `A1:A2` to `B`:

```
MlAppendMatrix("B", A1:A2)
```

		A1
		A2

`B` is now a 2-by-3 matrix with the data from `A1:A2` in the third column.

Example 2: Append data from a named worksheet cell range to a MATLAB matrix

`B` is a 2-by-2 MATLAB matrix. Cell `C1` contains the label (string) `B`, and `new_data` is the name of the cell range `A1:B2`. Append the data in cell range `A1:B2` to `B`:

```
MlAppendMatrix(C1, new_data)
```

`B` is now a 4-by-2 matrix with the data from `A1:B2` in the last two rows.

A1	B1
A2	B2

See Also

`MlPutMatrix`

MLAutoStart

Purpose Automatically start MATLAB process

Syntax

Worksheet: MLAutoStart("yes")
MLAutoStart("no")

Macro: MLAutoStart "yes"
MLAutoStart "no"

"yes" Automatically start the Spreadsheet Link EX and MATLAB software every time a Microsoft Excel session starts (default).

"no" Cancel automatic startup of the Spreadsheet Link EX and MATLAB software. If these products are running, it does not stop them.

Description

Sets automatic startup of the Spreadsheet Link EX and MATLAB software. When the Spreadsheet Link EX software is installed, the default is **yes**. A change of state takes effect the next time an Excel session starts.

Examples

Cancel automatic startup of the Spreadsheet Link EX and MATLAB software:

```
MLAutoStart("no")
```

These products do not start on subsequent Excel session invocations.

See Also

matlabinit | MLClose | MLOpen

Purpose Stop MATLAB process

Syntax

Worksheet:	MLClose()
Macro:	MLClose

Description Ends the MATLAB process, deletes all variables from the MATLAB workspace, and tells the Microsoft Excel software that the MATLAB software is no longer running. If no MATLAB process is running, nothing happens.

See Also MLOpen

MLDeleteMatrix

Purpose Delete MATLAB matrix

Syntax

Worksheet: MLDeleteMatrix(var_name)

Macro: MLDeleteMatrix var_name

var_name Name of MATLAB matrix to delete. "var_name" (in quotation marks) directly specifies the matrix name. var_name (without quotation marks) is an indirect reference: the function evaluates the contents of var_name to determine the matrix name, and var_name must be a worksheet cell address or range name.

Description Deletes the named matrix from the MATLAB workspace.

Examples Delete matrix A from the MATLAB workspace:

```
MLDeleteMatrix("A")
```


Purpose Evaluate command in MATLAB software

Syntax

Worksheet: MLEvalString(command)

Macro: MLEvalString command

command MATLAB command to evaluate. "command" (in quotation marks) directly specifies the command. command (without quotation marks) is an indirect reference: the function evaluates the contents of command to get the command, and command must be a worksheet cell address or range name.

Description

Passes a command string to the MATLAB software for evaluation. The specified action alters only the MATLAB workspace. It has no effect on the Microsoft Excel workspace.

Examples

Divide the MATLAB variable `b` by 2, and then plot it:

```
MLEvalString("b = b/2;plot(b)")
```

This command only modifies the MATLAB variable `b`. To update data in the Excel worksheet, use `MLGetMatrix`.

See Also

`MLGetMatrix`

MLGetFigure

Purpose Import current MATLAB figure into Microsoft Excel spreadsheet

Syntax

Worksheet: MLGetFigure(width,height)
Macro: MLGetFigure width, height
width Specify the width in normalized units of the MATLAB figure when imported into an Excel worksheet.
height Specify the height in normalized units of the MATLAB figure when imported into an Excel worksheet.

Description

Import the current MATLAB figure into an Excel worksheet, where the top-left corner of the figure is the current spreadsheet cell.

If worksheet calculation mode is automatic, `MLGetFigure` executes when you enter the formula in a cell. If worksheet calculation mode is manual, enter the `MLGetFigure` function in a cell, then press **F9** to execute it. However, pressing **F9** in this situation may also reexecute other worksheet functions and generate unpredictable results.

If you use `MLGetFigure` in a macro subroutine, enter `MatlabRequest` on the line after the `MLGetFigure`. `MatlabRequest` initializes internal Spreadsheet Link EX variables and enables `MLGetFigure` to function in a subroutine. Do not include `MatlabRequest` in a macro function unless the function is called from a subroutine.

Examples

Import the current MATLAB figure into an Excel worksheet. Adjust the width of the figure to be half that of the original figure, and the height to be a quarter that of the original figure:

```
MLGetFigure(.50,.25)
```

See Also

`MLGetMatrix` | `MLGetVar`

Purpose

Write contents of MATLAB matrix to Microsoft Excel worksheet

Syntax

Worksheet:	MLGetMatrix(var_name, edat)
Macro:	MLGetMatrix var_name, edat
var_name	Name of MATLAB matrix to access. "var_name" (in quotation marks) directly specifies the matrix name. var_name (without quotation marks) is an indirect reference: the function evaluates the contents of var_name to get the matrix name, and var_name must be a worksheet cell address or range name. var_name cannot be the MATLAB variable ans.
edat	Worksheet location where the function writes the contents of var_name. "edat" (in quotation marks) directly specifies the location and it must be a cell address or a range name. edat (without quotation marks) is an indirect reference: the function evaluates the contents of edat to get the location, and edat must be a worksheet cell address or range name.

Description

Writes the contents of MATLAB matrix `var_name` in the Excel worksheet, beginning in the upper-left cell specified by `edat`. If data exists in the specified worksheet cells, it is overwritten. If the dimensions of the MATLAB matrix are larger than that of the specified cells, the data overflows into additional rows and columns.

Caution

`edat` must not include the cell that contains the `MLGetMatrix` function. In other words, be careful not to overwrite the function itself. Also make sure there is enough room in the worksheet to write the matrix contents. If there is insufficient room, the function generates a fatal error.

MLGetMatrix function does not automatically adjust cell addresses. If edat is an explicit cell address, edit it to correct the address when you do either of the following:

- Insert or delete rows or columns.
- Move or copy the function to another cell.

If worksheet calculation mode is automatic, MLGetMatrix executes when you enter the formula in a cell. If worksheet calculation mode is manual, enter the MLGetMatrix function in a cell, and then press **F9** to execute it. However, pressing **F9** in this situation may also reexecute other worksheet functions and generate unpredictable results.

If you use MLGetMatrix in a macro subroutine, enter MatlabRequest on the line after the MLGetMatrix. MatlabRequest initializes internal Spreadsheet Link EX variables and enables MLGetMatrix to function in a subroutine. Do not include MatlabRequest in a macro function unless the function is called from a subroutine.

Examples

Example 1

Write the contents of the MATLAB matrix bonds starting in cell C10 of Sheet2. If bonds is a 4-by-3 matrix, fill cells C10..E13 with data:

```
MLGetMatrix("bonds", "Sheet2!C10")
```

Example 2

Access the MATLAB matrix named by the string in worksheet cell B12. Write the contents of the matrix to the worksheet starting at the location named by the string in worksheet cell B13:

```
MLGetMatrix(B12, B13)
```

Example 3

Write the contents of MATLAB matrix A to the worksheet, starting at the cell named by RangeA:

```
Sub Get_RangeA()  
MLGetMatrix "A", "RangeA"  
MatlabRequest  
End Sub
```

Example 4

In a macro, use the `Address` property of the range object returned by the VBA `Cells` function to specify where to write the data:

```
Sub Get_Variable()  
MLGetMatrix "X", Cells(3, 2).Address  
MatlabRequest  
End Sub
```

See Also

[MLAppendMatrix](#) | [MLPutMatrix](#)

MLGetVar

Purpose

Write contents of MATLAB matrix in Microsoft Excel VBA variable

Syntax

```
MLGetVar ML_var_name, VBA_var_name
```

ML_var_name Name of MATLAB matrix to access. "ML_var_name" (in quotation marks) directly specifies the matrix name. ML_var_name (without quotation marks) is an indirect reference: the function evaluates the contents of ML_var_name to get the matrix name, and ML_var_name must be a VBA variable containing the matrix name as a string. var_name cannot be the MATLAB variable ans. If defined, ML_var_name should be of type VARIANT. Any other type will give a "TYPE MISMATCH" error.

VBA_var_name Name of VBA variable where the function writes the contents of ML_var_name. Use VBA_var_name without quotation marks.

Description

Writes the contents of MATLAB matrix ML_var_name in the Excel Visual Basic for Applications (VBA) variable VBA_var_name. Creates VBA_var_name if it does not exist. Replaces existing data in VBA_var_name.

Examples

Write the contents of the MATLAB matrix J into the VBA variable DataJ:

```
Sub Fetch()  
MLGetVar "J", DataJ  
End Sub
```

See Also

MLPutVar

Purpose Set empty cells to NaN or 0

Syntax

Worksheet: MLMissingDataAsNaN("yes")
 MLMissingDataAsNaN("no") (Default)
Macro: MLMissingDataAsNaN "yes"
 MLMissingDataAsNaN "no" (Default)
"yes" Sets empty cells to use NaNs.
"no" Sets empty cells to use 0s. (Default)

Description

Sets empty cells to NaN or 0. When the Spreadsheet Link EX software is installed, the default is "no", so empty cells are handled as 0s. If you change the value of MLUseCellArray to "yes", the change remains in effect the next time a Microsoft Excel session starts.

Note A string in an Excel range always forces cell array output and empty cells as NaNs.

Examples

Cancel the use of the value NaN for empty cells:

```
MLMissingDataAsNaN('no')
```

See Also

MLPutMatrix

MLOpen

Purpose Start MATLAB process

Syntax

Worksheet: MLOpen()

Macro: MLOpen

Description Starts MATLAB process. If a MATLAB process has already started, subsequent calls to MLOpen do nothing. Use MLOpen to restart the MATLAB session after you have stopped it with MLClose in a given Microsoft Excel session.

Note We recommend using `matlabinit` rather than `MLOpen`, since `matlabinit` starts a MATLAB session and initializes the Spreadsheet Link EX software.

Examples Starts a MATLAB session:

`MLOpen()`

See Also `matlabinit` | `MLClose`

Purpose

Create or overwrite MATLAB matrix with data from Microsoft Excel worksheet

Syntax

Worksheet: `MLPutMatrix(var_name, mdat)`

Macro: `MLPutMatrix var_name, mdat`

`var_name` Name of MATLAB matrix to create or overwrite. "var_name" (in quotation marks) directly specifies the matrix name. `var_name` (without quotation marks) is an indirect reference: the function evaluates the contents of `var_name` to get the matrix name, and `var_name` must be a worksheet cell address or range name.

`mdat` Location of data to copy into `var_name`. `mdat` (no quotation marks). Must be a worksheet cell address or range name.

Description

Creates or overwrites matrix `var_name` in MATLAB workspace with specified data in `mdat`. Creates `var_name` if it does not exist. If `var_name` exists, this function replaces the contents with `mdat`. Empty numeric data cells within the range of `mdat` become numeric zeros within the MATLAB matrix identified by `var_name`.

If any element of `mdat` contains string data, `mdat` is exported as a MATLAB cell array. Empty string elements within the range of `mdat` become NaNs within the MATLAB cell array.

When using `MLPutMatrix` in a subroutine, indicate the source of the worksheet data using the Microsoft Excel macro `Range`. For example:

```
Sub test()  
  MLPutMatrix "a", Range("A1:A3")  
End Sub
```

If you have a named range in your worksheet, you can specify the name instead of the range; for example:

MLPutMatrix

```
Sub test()  
MLPutMatrix "a", Range("temp")  
End Sub
```

where temp is a named range in your worksheet.

Examples

Example 1 – Create or overwrite a matrix in the MATLAB workspace

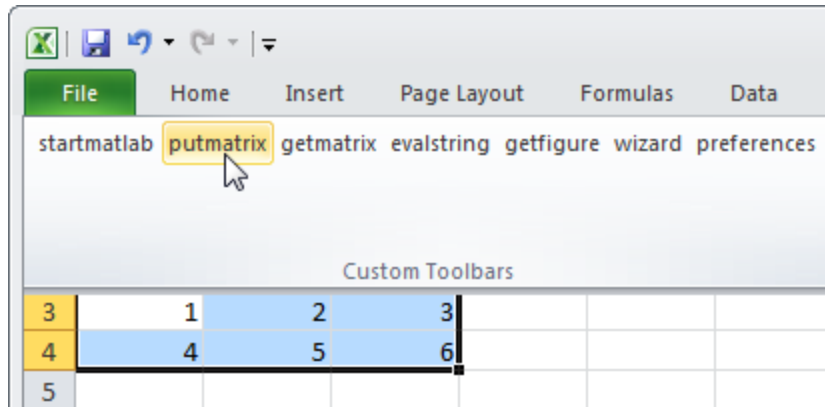
Create or overwrite matrix A in the MATLAB workspace with the data in the worksheet range A1:C3:

```
MLPutMatrix "A", Range("A1:C3")
```

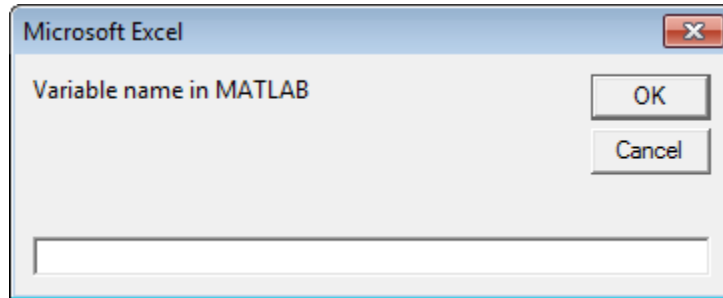
Example 2 – Use the putmatrix toolbar button to import data from a Microsoft Excel worksheet to the MATLAB workspace

Use the **putmatrix** toolbar button to import data from an Excel worksheet to the MATLAB workspace:

- 1 In the Excel worksheet, select the columns and/or rows you want to export to the MATLAB workspace.



- 2 Click the **putmatrix** button on the Spreadsheet Link EX toolbar. A window appears that prompts you to specify the name of the MATLAB variable in which you want to store your data.



- 3 Enter `newmatrix` for the MATLAB variable name.
- 4 Click **OK**.

Now you can manipulate `newmatrix` in the MATLAB Command Window.

```
newmatrix
newmatrix =

     1     2     3
     4     5     6
```

See Also

[MLAppendMatrix](#) | [MLGetMatrix](#)

MLPutVar

Purpose

Create or overwrite MATLAB matrix with data from Microsoft Excel VBA variable

Syntax

```
MLPutVar ML_var_name, VBA_var_name
```

ML_var_name Name of MATLAB matrix to create or overwrite. "ML_var_name" (in quotation marks) directly specifies the matrix name. ML_var_name (without quotation marks) is an indirect reference: the function evaluates the contents of ML_var_name to get the matrix name, and ML_var_name must be a VBA variable containing the matrix name as a string.

VBA_var_name Name of VBA variable whose contents are written to ML_var_name. Use VBA_var_name without quotation marks.

Description

Creates or overwrites matrix ML_var_name in MATLAB workspace with data in VBA_var_name. Creates ML_var_name if it does not exist. If ML_var_name exists, this function replaces the contents with data from VBA_var_name. Use MLPutVar only in a macro subroutine, not in a macro function or in a subroutine called by a function.

Empty numeric data cells within VBA_var_name become numeric zeros within the MATLAB matrix identified by ML_var_name.

If any element of VBA_var_name contains string data, VBA_var_name is exported as a MATLAB cell array. Empty string elements within VBA_var_name become NaNs within the MATLAB cell array.

Examples

Create (or overwrite) the MATLAB matrix K with the data in the Excel Visual Basic for Applications (VBA) variable DataK.

```
Sub Put()  
    MLPutVar "K", DataK
```

End Sub

See Also

MLGetVar

MLShowMatlabErrors

Purpose Return standard Spreadsheet Link EX errors or full MATLAB errors using MLEvalString

Syntax

Worksheet: MLShowMatlabErrors("yes")
 MLShowMatlabErrors("no") (Default)

Macro: MLShowMatlabErrors "yes"
 MLShowMatlabErrors "no" (Default)

"yes" Displays the full MATLAB error string upon MLEvalString failure.

"no" Displays the standard Spreadsheet Link EX errors upon MLEvalString failure.

Description Sets the Spreadsheet Link EX error display mode when executing MATLAB commands using MLEvalString.

Examples

- Cause MLEvalString failures to show standard Spreadsheet Link EX errors, such as #COMMAND.

```
MLShowMatlabErrors("no")
```

- Cause MLEvalString failures to show MATLAB error strings, such as ??? Undefined function or variable 'foo'.

```
MLShowMatlabErrors("yes")
```

See Also

MLEvalString

Purpose Specify MATLAB current working folder after startup

Syntax

Worksheet: MLStartDir(path)

Macro: MLStartDir path

path Specify the current MATLAB working folder after startup.

Description

Sets the MATLAB working folder after startup. This function does not work like the standard Microsoft Windows **Start In** setting, because it does not automatically run `startup.m` or `matlabrc.m` in the specified folder.

Note The working folder changes only if you run MATLAB *after* you run this function. Running this function while MATLAB is running does not change the working folder for the current session. In this case, MATLAB uses the specified folder as the working folder when it is restarted.

Examples

Set the MATLAB working folder to `d:\work` after startup:

```
MLStartDir ( d:\work )
```

If your folder path includes a space, embed the path in single quotation marks within double quotation marks. For example, to set the MATLAB working folder to `d:\my work`, run the command:

```
MLStartDir ( 'd:\my work' )
```

See Also

MLAutoStart

MLUseCellArray

Purpose Toggle MLPutMatrix to use MATLAB cell arrays

Syntax

Worksheet: MLUseCellArray("yes")
 MLUseCellArray("no")

Macro: MLUseCellArray "yes"
 MLUseCellArray "no"

"yes" Automatically uses cell arrays for transfer of data structures.

"no" Do not automatically use cell arrays for transfer of data (default).

Description

Using MLUseCellArray forces MLPutMatrix to use cell arrays for transfer of data (for example, dates). When the Spreadsheet Link EX software is installed, the default is "no". If you change the value of MLUseCellArray to "yes", the change remains in effect the next time a Microsoft Excel session starts.

Examples

Cancel automatic use of cell arrays for easy transfer of data:

```
MLUseCellArray("no")
```

See Also

MLPutMatrix

Purpose Specify whether to use full MATLAB desktop or MATLAB Command Window

Syntax

Worksheet:	<code>MLUseFullDesktop("yes")</code> <code>MLUseFullDesktop("no")</code>
Macro:	<code>MLUseFullDesktop "yes"</code> <code>MLUseFullDesktop "no"</code>
"yes"	Start full MATLAB desktop.
"no"	Start the MATLAB Command Window only.

Description Sets the MATLAB session to start with the full desktop or Command Window only. When the Spreadsheet Link EX software is installed, the default is "yes".

Examples Start only the MATLAB Command Window:

```
MLUseFullDesktop("no")
```

See Also `matlabinit` | `MLClose` | `MLOpen`

Examples

Use this list to find examples in the documentation.

Macro Examples

“Sending MATLAB Data to an Excel Worksheet and Displaying the Results” on page 1-30

“Importing and Exporting Data Between the Microsoft® Excel Interface and the MATLAB Workspace” on page 1-33

Financial Examples

“Modeling Data Sets Using Data Regression and Curve Fitting” on page 2-2

“Interpolating Data” on page 2-11

“Pricing Stock Options Using the Binomial Model” on page 2-15

“Calculating and Plotting the Efficient Frontier of Financial Portfolios” on page 2-19

“Mapping Time and Bond Cash Flows” on page 2-24

A

- add-in, Spreadsheet Link EX 3-3
- Add-In, Spreadsheet Link EX 1-7 1-9
- audible error signals 3-10
- /automation option 1-17

B

- beeps 3-10
- binomial tree 2-15

C

- calculation mode 3-5
- cash flow example 2-24
- COLS error 3-2
- COMMAND error 3-2
- computer memory errors 3-10
- curve fitting example 2-2

D

- data
 - matrix data errors 3-8
- data errors 3-8
- data interpolation example 2-11
- data types 1-2
- data-location argument 3-8 3-10
- date numbers 1-33
- date system 1-33
- dates 1-33
- DIMENSION error 3-2
- double quotation marks 3-3

E

- efficient frontier example 2-19
- empty matrix 3-8
- errors
 - Excel error message boxes 3-5
 - troubleshooting 3-1

- worksheet cell errors 3-2

examples

- cash flow 2-24
- efficient frontier 2-19
- interpolating data 2-11
- regression and curve fitting 2-2
- stock option 2-15

- excllink.xla 1-4
- excllink.xla add-in 3-5
- exlink.ini file 1-4
- ExliSamp.xls file
 - location 1-4
 - purpose 2-1

F

- file initialization 1-4
- Function Wizard for the Spreadsheet Link EX Software 1-25
- functions
 - about 1-20
 - arguments
 - working with 1-23
- MATLAB Function Wizard for the Spreadsheet Link EX Software 1-25
- Spreadsheet Link EX
 - types of 1-20
- Spreadsheet Link EX versus Microsoft Excel 1-20
- using in macros 1-30

I

- initialization file 1-4
- interpolating data 2-11
- INVALIDNAME error 3-2
- INVALIDTYPE error 3-2

K

- Kernel132.dll 1-4

L

license passcode 3-6
localization 1-35

M

macros
 creating 1-30
MATLAB error 3-3
MATLAB Function Wizard for the Spreadsheet
 Link EX Software 1-25
matlabfcn 5-2
matlabinit 5-3
matlabsub 5-4
matrix dimensions 3-2
MLAppendMatrix 5-6
MLAutoStart 5-8
MLClose 5-9
MLDeleteMatrix 5-10
MLEvalString 5-11
MLFullDesktop 5-27
MLGetFigure 5-12
MLGetMatrix 5-13
MLGetVar 5-16
MLMissingDataAsNaN 5-17
MLOpen 5-18
MLPutMatrix 5-19
MLPutVar 5-22
MLShowMatlabErrors 5-24
MLStartDir 5-25
MLUseCellArray 5-26

N

NAME error 3-3
NONEXIST error 3-3
nonexistent variable 3-8
non-U.S. users
 information for 1-35

P

passcode
 license 3-6
Preferences
 setting 1-14

R

regression and curve fitting 2-2
ROWS error 3-3

S

signals error 3-10
single quotation marks 3-3
spreadsheet formulas 1-21
Spreadsheet Link EX functions
 about 1-20
Spreadsheet Link EX software
 configuring
 for Excel 2003 and earlier versions 1-7
 for Excel 2007 1-9
 installing 1-4
 overview 1-2
 starting 1-17
 stopping 1-4 1-19
 using 2-1
spreadsheets 1-21
 using 1-21
startup error signals 3-10
stock option pricing example 2-15
SYNTAX error 3-3
system
 date 1-33
system path
 files on 1-4
system requirements 1-4

T

troubleshooting 3-1

V

VALUE error 3-4

W

worksheet formulas 1-21

worksheets 1-21

errors when opening 3-8

using 1-21

Z

zero matrix 3-8

zero matrix cells 3-8