Spreadsheet Link™ EX 3
User's Guide

MATLAB®

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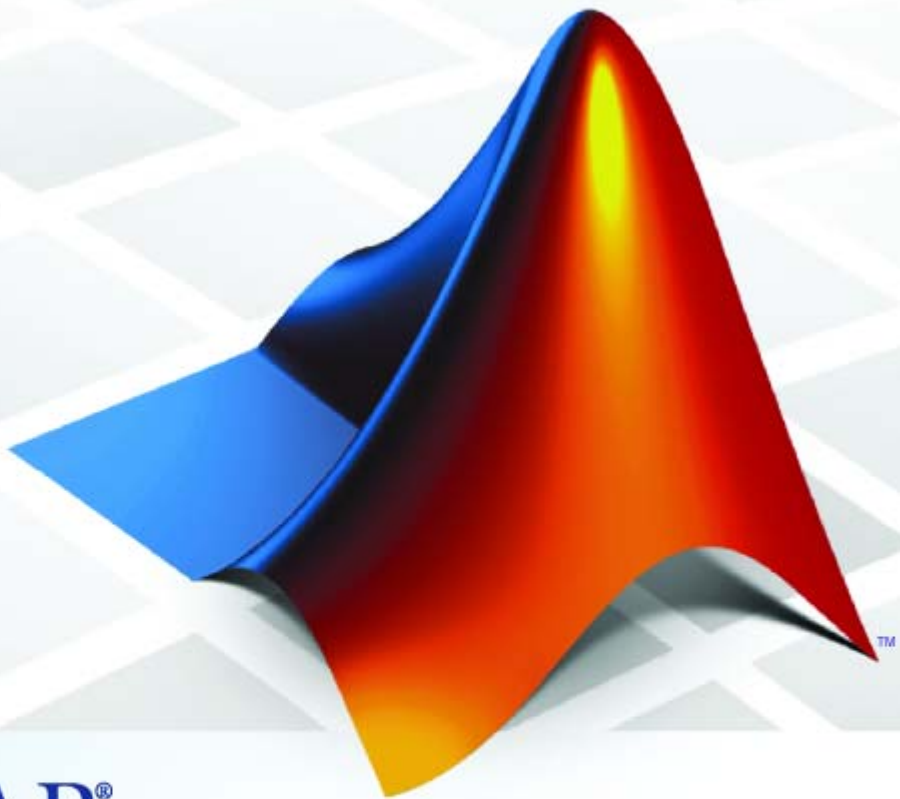
How to Contact The MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Spreadsheet Link™ EX User's Guide

© COPYRIGHT 1996–2008 by The MathWorks, Inc.

Trademarks

Patents

Contents

Solving Problems with the Spreadsheet Link EX Software

2

Function Reference

3

4 Functions — Alphabetical List

A Error Messages and Troubleshooting

B Examples

Index

1

Getting Started

Product Overview

The Spreadsheet Link™ EX software 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.
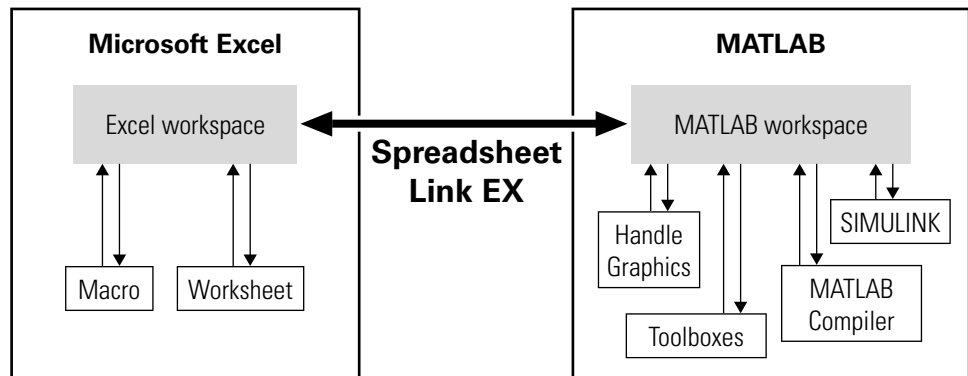
Installing the Spreadsheet Link EX Software

In this section...
"System Requirements" on page 1-3
"Product Installation" on page 1-3
"Files and Directories Created by the Installation" on page 1-3
"Modifying Your System Path" on page 1-4

System Requirements

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

The Spreadsheet Link EX product requires the MATLAB for Microsoft
Windows software. For best results with MATLAB figures and graphics, set
the color palette of your display to a value greater than 256 colors:

1 Click **Start > Settings > Control Panel > Display**.

2 Click the **Settings** tab. Choose an appropriate entry from the **Color
Palette** menu.

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.

Files and Directories Created by the Installation

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

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

- excllink.xla: The Spreadsheet Link EX add-in

- ExliSamp.xls: Spreadsheet Link EX example files described in this documentation

The installation also creates a Spreadsheet Link EX initialization file, exlink.ini, in the appropriate Windows directory (for example, C:\Winnt).

The Spreadsheet Link EX software uses Kernel32.dll, which should already be in the appropriate Windows system directory (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.

Configuring the Spreadsheet Link EX Software

In this section...

"Configuring Version 2003 and Earlier Versions of the Microsoft® Excel Software" on page 1-5

"Configuring Version 2007 of the Microsoft® Excel Software" on page 1-7

"Setting Spreadsheet Link EX Preferences" on page 1-10

Configuring Version 2003 and Earlier Versions of the Microsoft Excel Software

1 Start Microsoft Excel.

2 Enable the Spreadsheet Link EX add-in.

    a Click **Tools > Add-Ins**. The Add-Ins dialog box appears.

    b Click **Browse**.

    c Select *matlabroot*\toolbox\exlink\excllink.xla.

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

    d Click **OK**.

    In the Add-Ins dialog box, the **Spreadsheet Link EX 3.0.1 for use with MATLAB** check box is now selected.

    e 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.
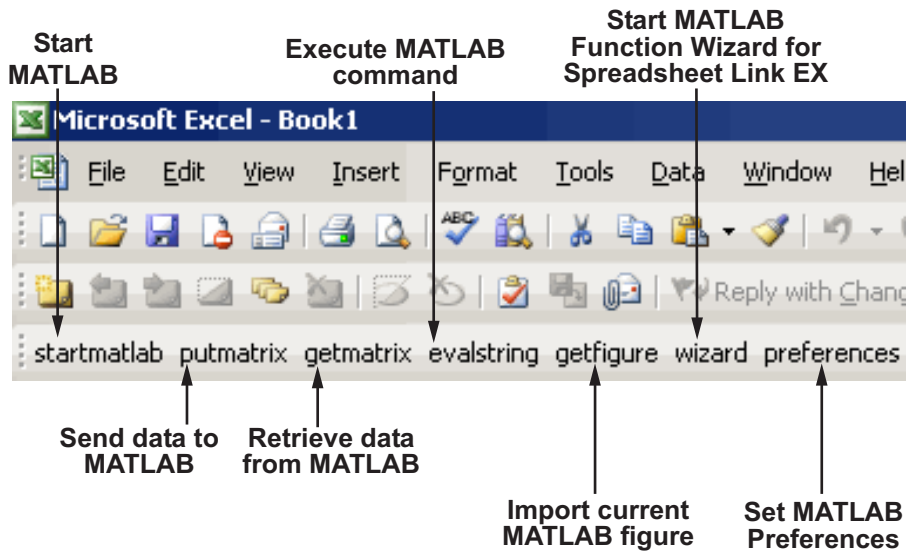
The Spreadsheet Link EX software is now ready for use.

Configuring Version 2007 of the Microsoft Excel Software

1 Click **My Computer > Properties**. The System Properties pane appears.

2 Click the **Advanced** tab.

3 Click **Environment Variables**.

4 Under **System variables**, double-click the PATH variable. The Edit System Variable window appears.

5 Add *matlabroot*\bin\win32 to the **Variable value** field.

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

---

6 Click **OK** on all dialog boxes until you exit System Properties.

7 Right-click the **MATLAB** shortcut icon and choose **Properties** from its context menu. The Properties pane appears.

8 In the **Target** field, enter /regserver.

9 Click **Apply**.

10 Start a Microsoft Excel session.

11 Enable the Spreadsheet Link EX add-in.

   a Click , the Microsoft Office Button.

   b Click **Excel Options**. The Excel Options dialog box appears.

   c Click **Add-Ins**.

   d From the **Manage** selection list, choose **Excel Add-Ins**.

   e Click **Go**. The Add-Ins dialog box appears.

   f Click **Browse**.

g Select *matlabroot*\toolbox\exlink\excllink.xla.

h Click **Open**.

i In the Add-Ins dialog box, the **Spreadsheet Link EX 3.0.1 for use with MATLAB** check box is now selected.
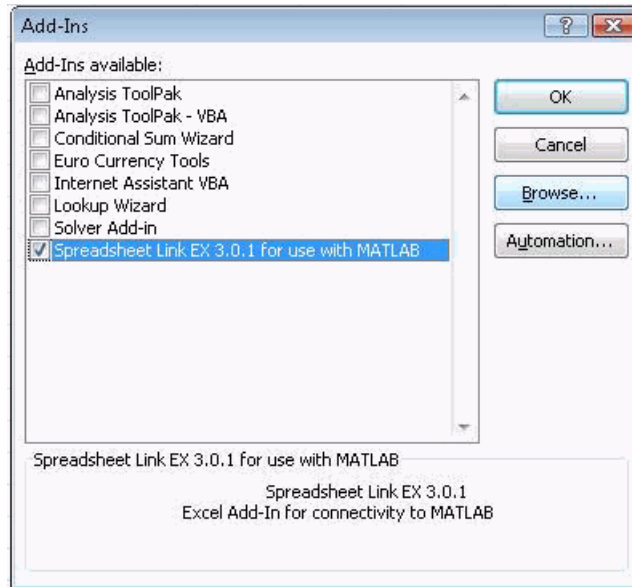
j Click **OK** to close the Add-Ins dialog box.

k 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.
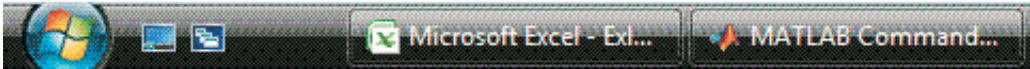
The Spreadsheet Link EX toolbar appears on your Excel worksheet.
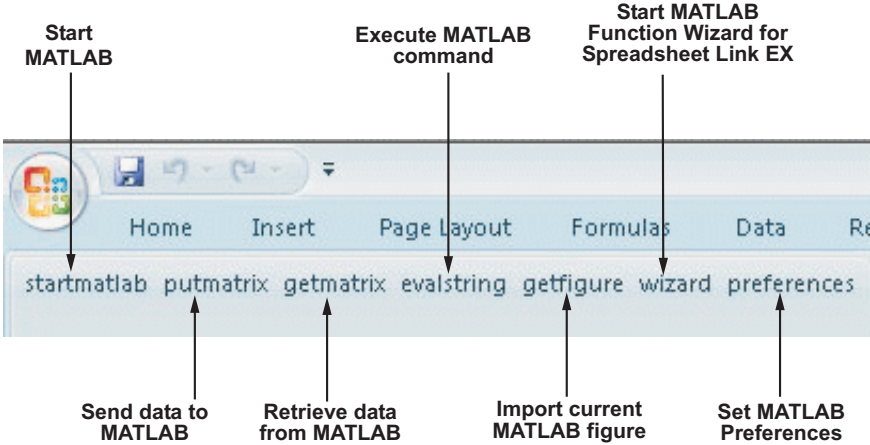
The Spreadsheet Link EX software is now ready for use.

Setting Spreadsheet Link EX Preferences

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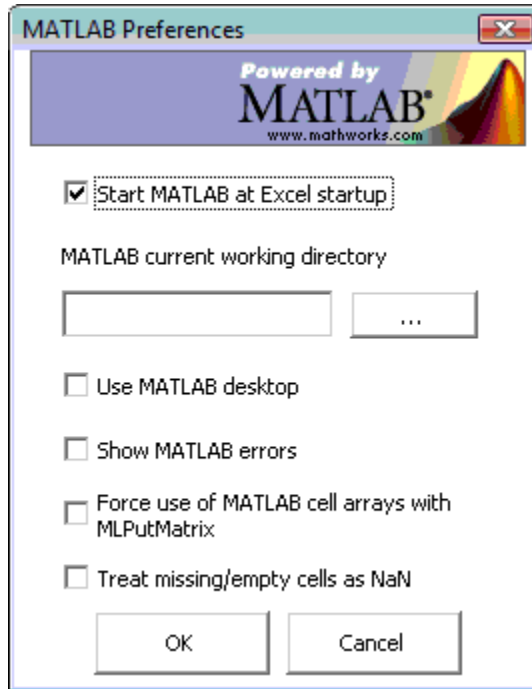
Preferences include:

- **Start MATLAB when Excel starts (enabled by default)** starts a MATLAB session automatically when an Excel session starts.

- **MATLAB current working directory** enables you to specify the current working directory for your MATLAB session at startup.

- **Use MATLAB desktop** starts the MATLAB desktop, including the current directory, workspace, command history, and Command Window panes, when an Excel session starts.

- **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.

Starting and Stopping the Spreadsheet Link EX Software

In this section...
"Automatically Starting the Spreadsheet Link EX Software" on page 1-12
"Manually Starting the Spreadsheet Link EX Software" on page 1-12
"Connecting to an Existing MATLAB Session" on page 1-13
"Stopping the Spreadsheet Link EX Software" on page 1-13

Automatically Starting the Spreadsheet Link EX Software

When installed and configured according to the instructions in "Configuring the Spreadsheet Link EX Software" on page 1-5, 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**.

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

   For more information about the matlabinit function, see Chapter 3, "Function Reference".

3 Click **Run**.

   The MATLAB Command Window button appears on the Microsoft Windows taskbar.
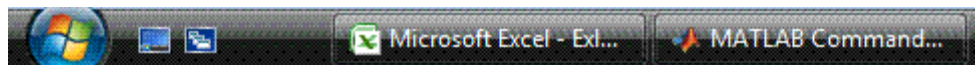
Connecting to an Existing MATLAB Session

To connect a new Excel session to an existing MATLAB session, start MATLAB with the `spredsheetautomation` command-line option. This option starts MATLAB as an automation server. The Command Window is minimized, and the Desktop is not running.

1 Right-click your MATLAB shortcut icon.

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

- To stop both the Spreadsheet Link EX and MATLAB software, stop 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.

About Functions

In this section...

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. Standard MATLAB function names are always lowercase; for example, plot(f).

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 theExcel 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 3, "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 1-17.

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 directories and run functions from within the Excel interface.

List functions available for specified directory/category

Display list of MATLAB working directories and available function categories
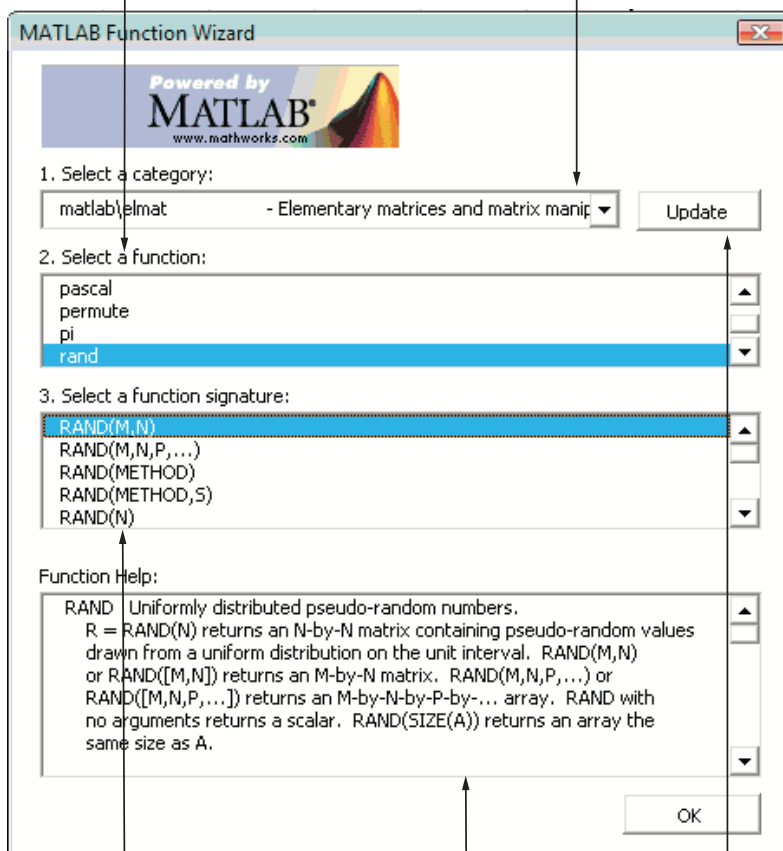
Select function signature and enter formula into specified spreadsheet cell

Display help for given function signature

Refresh directory/category list

You can use this wizard to:

1 Display a list of all MATLAB working directories and function categories

All directories 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 directory path plus a description read from the Contents.m file in that directory. If no Contents.m file is found, the directory/category display notifies you as follows:

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

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

2 Choose a particular directory or category, and list functions available for that directory or category

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

3 Parse a specified function signature and enter a formula into the current spreadsheet cell

After you select a function, available function signatures for the specified function display in the **Select a function signature** field. Click a function signature to display the Function Arguments pane.

Specify cell for function output (optional)

Scroll through list of function input arguments

MATLAB Function Wizard

Powered by
MATLAB®
www.mathworks.com

1. Select a category:

matlab\elmat   - Elementary matrices and matrix manip ▼   Update

Function Arguments

RAND   Inputs:
                                              M

                           Optional output cell(s):

RAND   Uniformly distributed pseudo-random numbers.
  R = RAND(N) returns an N-by-N matrix containing pseudo-random values
  drawn from a uniform distribution on the unit interval. RAND(M,N)
  or RAND([M,N]) returns an M-by-N matrix. RAND(M,N,P,...) or
  RAND([M,N,P,...]) returns an M-by-N-by-P-by-... array. RAND with

                                                        OK   Cancel

  drawn from a uniform distribution on the unit interval. RAND(M,N)
  or RAND([M,N]) returns an M-by-N matrix. RAND(M,N,P,...) or
  RAND([M,N,P,...]) returns an M-by-N-by-P-by-... array. RAND with
  no arguments returns a scalar. RAND(SIZE(A)) returns an array the
  same size as A.

                                                        OK

Double-click function signature to display Function Arguments pane ...

Enter function arguments

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)
```

4 Display online help headers for functions

After you select a function signature from the **Select a function signature** field, its help header appears in the **Function Help** field.

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 1-23.

- 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 1-23.

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
```

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.

1-23

- For instructions on enabling this add-in for the Excel 2003 software, see "Configuring Version 2003 and Earlier Versions of the Microsoft® Excel Software" on page 1-5.

- For instructions on enabling this add-in for the Excel 2007 software, see "Configuring Version 2007 of the Microsoft® Excel Software" on page 1-7.

4 Enable the Spreadsheet Link EX software as a Reference in the Microsoft® Visual Basic® editor.

  a 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 software, click the Visual Basic button, 📝, or press **Alt+F11**.

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

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

  d Click **OK**.

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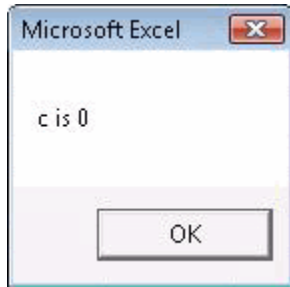
8 Click **OK** to close the dialog box.

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.

Working with Dates

Default Microsoft Excel date numbers represent the number of days that have passed since January 1, 1900; for example, May 15, 1996 is represented as 35200 in the Excel software.

However, MATLAB date numbers represent the number of days that have passed since January 1, 0000, so May 15, 1996 is represented as 729160 in the MATLAB software. Therefore, the difference in dates between the Excel software and the MATLAB software is a constant, 693960 (729160 minus 35200).

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.

2

Solving Problems with the Spreadsheet Link EX Software

- "Running the Examples" on page 2-2
- "Modeling Data Sets Using Data Regression and Curve Fitting" on page 2-3
- "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 2-23

Note For other applications, see "Using Spreadsheet Link EX with Bioinformatic Data".

Running the Examples

The following sections show how the Microsoft Excel, Spreadsheet Link EX, and MATLAB software work together to solve real-world problems.

These examples are included with the Spreadsheet Link EX product. To run them:

1 Start Excel, Spreadsheet Link EX, and MATLAB sessions.

2 Navigate to the directory *matlabroot*\toolbox\exlink\.

3 Open the file ExliSamp.xls

4 Execute the examples as needed.

Note Examples 1 and 2 use MATLAB functions only. Examples 3, 4, and 5 use Financial Toolbox™ functions. The Financial Toolbox software requires the Statistics Toolbox™ and Optimization Toolbox™ products.

Modeling Data Sets Using Data Regression and Curve Fitting

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.

Using Worksheets

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

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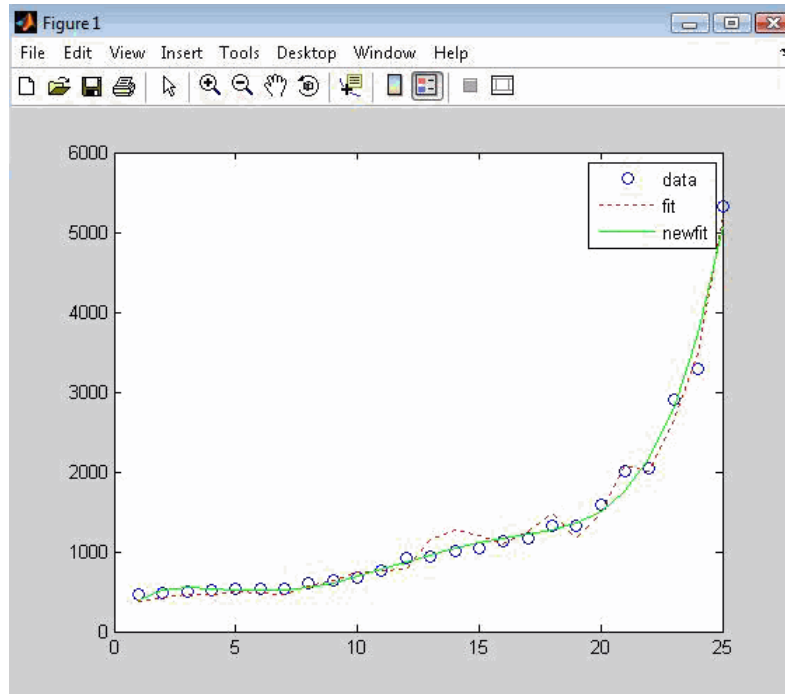
Since the data is closely correlated but not exactly linearly dependent, the fit curve (dashed line) shows a close, but not an exact, fit. The fifth-degree polynomial curve, newfit, is a more accurate mathematical model for the data.

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

Using Macros

1 Click the **Sheet2** tab on ExliSamp.xls. The worksheet for this example appears.
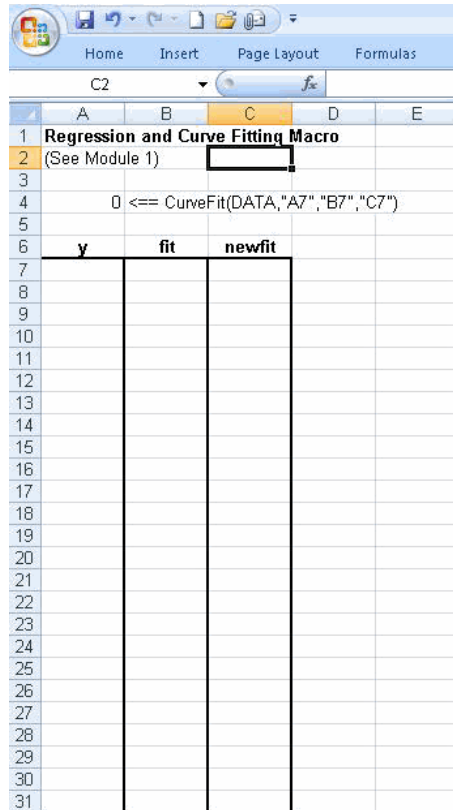
2 Make cell A4 the active cell, but do not execute it yet.

Cell A4 calls the macro CurveFit, which you can examine in the Microsoft Visual Basic environment.
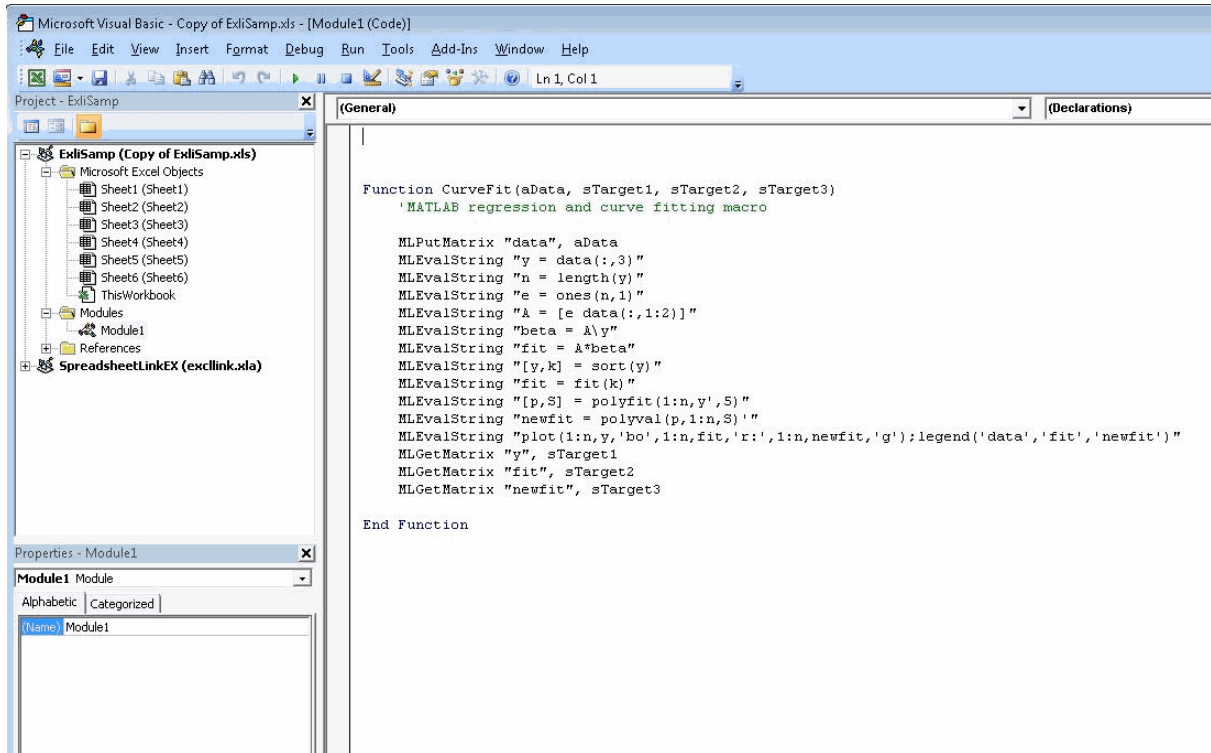
3 While this module is open, make sure that the Spreadsheet Link EX add-in is enabled.

- If you are using the Excel 2003 software:

  1 Click **Tools > References**.

  2 In the **References** dialog box, make sure that the excllink.xla check box is selected. If not, select it.

  3 Click **OK**.

- If you are using the Excel 2007 software:

  1 Click the Microsoft Office Button, .

  2 Click **Options**. The Excel Options pane appears.
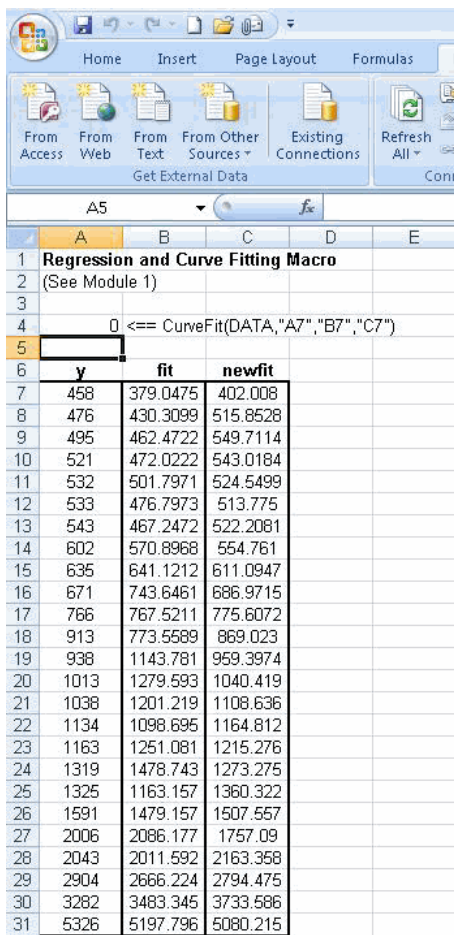
  3 Click **Add-Ins**.

4 From the **Manage** selection list, choose **Excel Add-Ins**.

5 Click **Go**. The Add-Ins pane appears.

6 Make sure that the **Spreadsheet Link EX 3.0.1 for use with MATLAB** check box is selected. If not, select it.
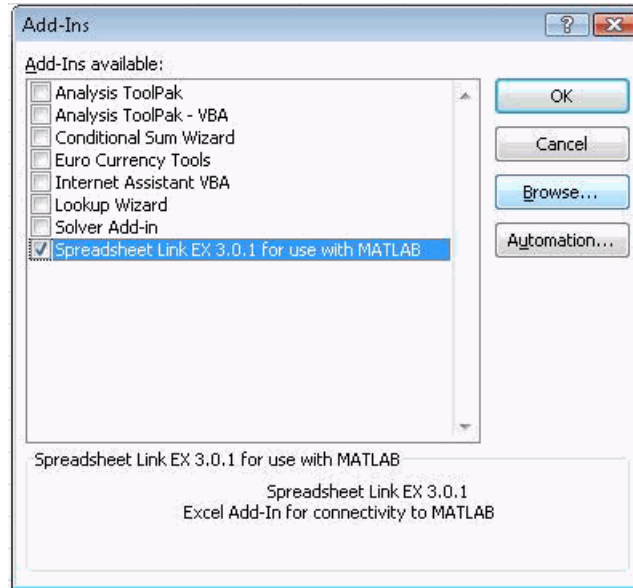
7 Click **OK** to close the Add-Ins pane.

8 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.

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 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.

Formula bar: A33 — `=MLPutMatrix("Labels", A4:C4)`

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Data Interpolation																			
3	Original Data					Interpolated Values														
4	Time	Temp	Volume																	
5	0.025	68.00	2504.08			Temp														
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	458.000	379.05	402.01		0.025															
8	476.000	430.31	515.85		0.05															
9	495.000	462.47	549.71		0.075															
10	521.000	472.02	543.02		0.1															
11	532.000	501.80	524.55		0.125															
12	533.000	476.80	513.77		0.15															
13	543.000	467.25	522.21		0.175															
14	602.000	570.90	554.76		0.2															
15	635.000	641.12	611.09		0.225															
16	671.000	743.65	686.97		0.25															
17	766.000	767.52	775.61		0.275															
18	913.000	773.56	869.02		0.3															
19	938.000	1143.78	959.40		0.325															
20	1013.000	1279.59	1040.42		0.35															
21	1038.000	1201.22	1108.64		0.375															
22	1134.000	1098.69	1164.81		0.4															
23	1163.000	1251.08	1215.28		0.425															
24	1319.000	1478.74	1273.27		0.45															
25	1325.000	1163.16	1360.32		0.475															
26	1591.000	1479.16	1507.56		0.5															
27	2006.000	2086.18	1757.09		0.525															
28	2043.000	2011.59	2163.36		0.55															
29	2904.000	2666.22	2794.48		0.575															
30	3282	3483.345	3733.586		0.6															
31	5326	5197.796	5080.215																	
32	1. Transfer original data to MATLAB.																		
33	0	<== MLPutMatrix("Labels", A4:C4)																		
34	0	<== MLPutMatrix("X",A5:A29)																		
35	0	<== MLPutMatrix("T",B5:B29)																		
36	0	<== MLPutMatrix("V",C5:C29)																		
38	2. Transfer interpolation data points to MATLAB.																			
39	0	<== MLPutMatrix("Xa",E7:E30)																		
40	0	<== MLPutMatrix("Ta",F6:T6)																		
42	3. Execute MATLAB data interpolation function.																			
43	#COMMA	<== MLEvalString("[Xl, Tl, Vl] = griddata(X,T,V,Xa,Ta, 'invdist')")																		
45	4. Transpose output data matrix and transfer data to Excel.																			
46	#COMMA	<== MLEvalString("IV = Vl';")																		
47	#NONEXIS	<== MLGetMatrix("IV","F7")																		
49	5. Plot interpolated data and label the figure.																			
50	#COMMA	<== 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

Time	Temp 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	2700.39	2859.95	3006.62	3144.98	3277.18	3404.45	3527.63	3647.28	3763.86	3877.69	3989.04	4098.13	4205.14	4310.24
0.05	2509.72	2704.37	2863.51	3009.93	3148.12	3280.18	3407.35	3530.43	3650.01	3766.52	3880.28	3991.57	4100.61	4207.58	4312.63
0.075	2515.27	2708.29	2867.04	3013.22	3151.25	3283.18	3410.24	3533.23	3652.73	3769.17	3882.87	3994.10	4103.09	4210.01	4315.02
0.1	2520.45	2712.14	2870.53	3016.49	3154.36	3286.16	3413.11	3536.02	3655.44	3771.81	3885.45	3996.63	4105.57	4212.44	4317.40
0.125	2525.29	2715.91	2873.99	3019.74	3157.45	3289.13	3415.98	3538.80	3658.14	3774.44	3888.02	3999.15	4108.03	4214.86	4319.78
0.15	2529.89	2719.62	2877.41	3022.97	3160.53	3292.09	3418.84	3541.57	3660.84	3777.07	3890.59	4001.66	4110.50	4217.28	4322.15
0.175	2534.31	2723.25	2880.81	3026.18	3163.59	3295.03	3421.68	3544.33	3663.53	3779.69	3893.15	4004.17	4112.96	4219.69	4324.52
0.2	2538.58	2726.80	2884.16	3029.36	3166.63	3297.96	3424.52	3547.08	3666.21	3782.31	3895.71	4006.67	4115.41	4222.10	4326.89
0.225	2542.72	2730.29	2887.48	3032.52	3169.66	3300.88	3427.34	3549.82	3668.88	3784.91	3898.26	4009.17	4117.86	4224.50	4329.25
0.25	2546.77	2733.71	2890.77	3035.66	3172.67	3303.78	3430.16	3552.56	3671.54	3787.51	3900.80	4011.66	4120.30	4226.90	4331.61
0.275	2550.74	2737.07	2894.02	3038.77	3175.66	3306.67	3432.96	3555.28	3674.19	3790.11	3903.34	4014.14	4122.73	4229.29	4333.96
0.3	2554.62	2740.36	2897.23	3041.86	3178.64	3309.55	3435.75	3558.00	3676.84	3792.69	3905.87	4016.62	4125.17	4231.68	4336.30
0.325	2558.44	2743.59	2900.41	3044.93	3181.60	3312.41	3438.53	3560.70	3679.48	3795.27	3908.39	4019.09	4127.59	4234.06	4338.64
0.35	2562.20	2746.77	2903.56	3047.97	3184.54	3315.26	3441.29	3563.40	3682.11	3797.84	3910.90	4021.56	4130.01	4236.44	4340.98
0.375	2565.90	2749.89	2906.67	3050.99	3187.46	3318.10	3444.05	3566.08	3684.73	3800.40	3913.41	4024.02	4132.42	4238.81	4343.31
0.4	2569.55	2752.97	2909.74	3053.98	3190.37	3320.92	3446.80	3568.76	3687.34	3802.96	3915.92	4026.47	4134.83	4241.17	4345.64
0.425	2573.16	2756.00	2912.78	3056.96	3193.26	3323.73	3449.53	3571.42	3689.95	3805.50	3918.41	4028.92	4137.24	4243.54	4347.96
0.45	2576.72	2759.00	2915.79	3059.90	3196.13	3326.52	3452.25	3574.08	3692.54	3808.04	3920.90	4031.36	4139.63	4245.89	4350.28
0.475	2580.24	2761.95	2918.76	3062.83	3198.98	3329.30	3454.96	3576.72	3695.13	3810.58	3923.38	4033.80	4142.03	4248.24	4352.59
0.5	2583.72	2764.87	2921.70	3065.73	3201.81	3332.06	3457.66	3579.36	3697.71	3813.10	3925.86	4036.23	4144.41	4250.59	4354.90
0.525	2587.17	2767.76	2924.61	3068.61	3204.63	3334.82	3460.34	3581.98	3700.27	3815.62	3928.33	4038.65	4146.79	4252.93	4357.20
0.55	2590.58	2770.61	2927.49	3071.46	3207.43	3337.55	3463.02	3584.60	3702.84	3818.13	3930.79	4041.06	4149.17	4255.26	4359.50
0.575	2593.96	2773.44	2930.34	3074.29	3210.21	3340.28	3465.68	3587.21	3705.39	3820.63	3933.24	4043.48	4151.54	4257.59	4361.79
0.6	2597.31	2776.24	2933.16	3077.10	3212.98	3342.98	3468.33	3589.80	3707.93	3823.12	3935.69	4045.88	4153.90	4259.92	4364.08

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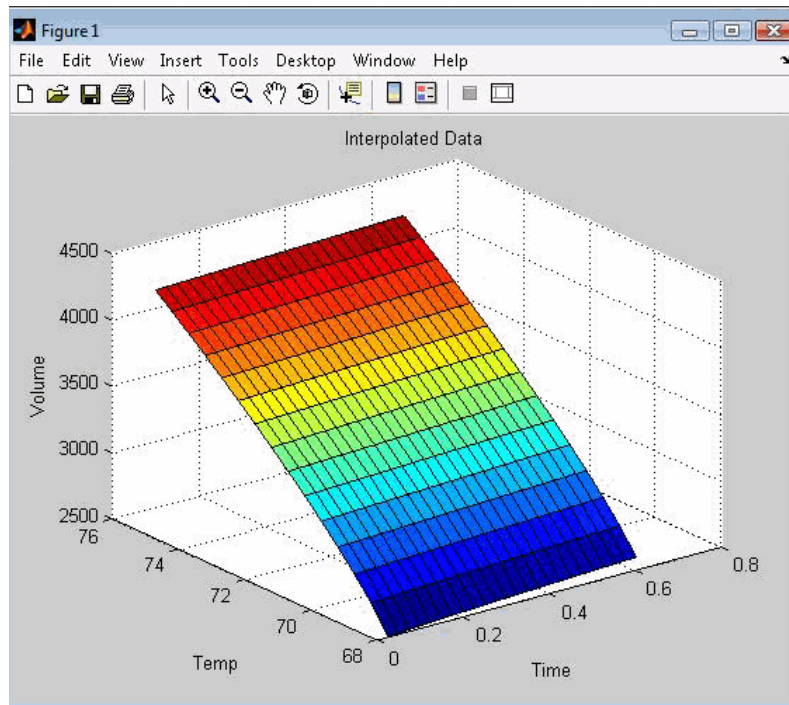
When you finish the example, close the figure window.

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.

Note This example requires the Financial Toolbox software.

---

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

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.
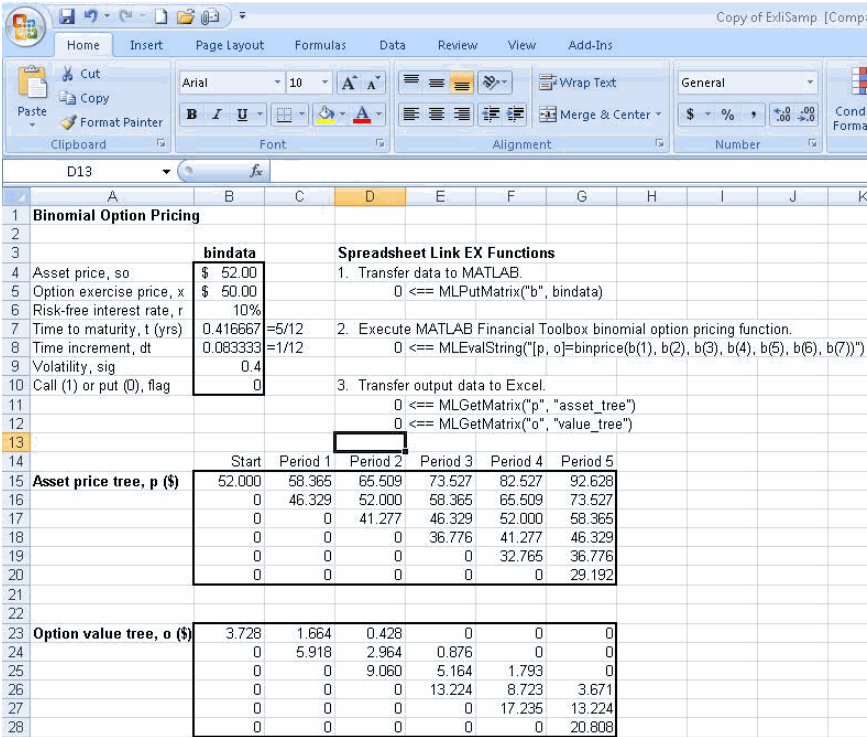
Read the asset price tree as follows:

- 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 reexecuting the Spreadsheet Link EX functions.

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.

Note This example requires the Financial Toolbox software.

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

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.

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.

6 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.
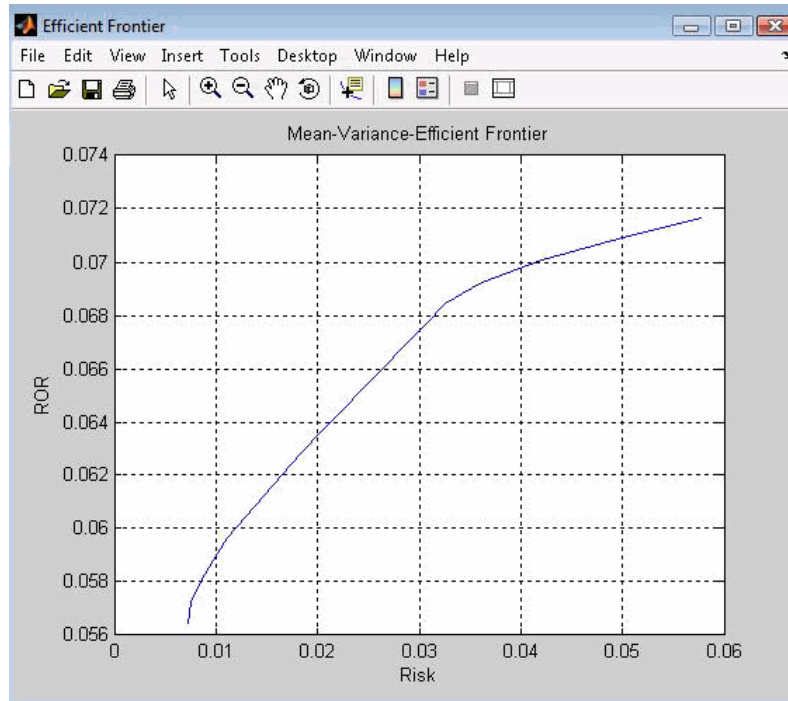
The light blue line shows the efficient frontier. Note the change in slope above a 6.8% return because the Corporate Bond portfolio no longer contributes to the efficient frontier.

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

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

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.

The spreadsheet contains the following data:

Cash Flow and Time Mapping for a Portfolio of Bonds

Settlement Date	26-Jul-99

Bond Data

	Maturity	Coupon Rate
Bond1	15-Nov-99	0.05875
Bond2	15-May-00	0.06375
Bond3	15-Nov-00	0.08500
Bond4	15-May-01	0.08000
Bond5	15-Nov-01	0.15750

Cash Flow Dates

Bond1	7/26/1999	11/15/1999	N/A	N/A	N/A	N/A
Bond2	7/26/1999	11/15/1999	5/15/2000	N/A	N/A	N/A
Bond3	7/26/1999	11/15/1999	5/15/2000	11/15/2000	N/A	N/A
Bond4	7/26/1999	11/15/1999	5/15/2000	11/15/2000	5/15/2001	N/A
Bond5	7/26/1999	11/15/1999	5/15/2000	11/15/2000	5/15/2001	11/15/2001

Cash Flow Amounts

	7/26/1999	11/15/1999	5/15/2000	11/15/2000	5/15/2001	11/15/2001
Bond1	-1.1495	102.9375	0	0	0	0
Bond2	-1.2473	3.1875	103.1875	0	0	0
Bond3	-1.6630	4.2500	4.2500	104.2500	0	0
Bond4	-1.5652	4.0000	4.0000	4.0000	104.0000	0
Bond5	-3.0815	7.8750	7.8750	7.8750	7.8750	107.8750

Spreadsheet Link EX Functions

1. Transfer data to MATLAB.

```
0 <== MLPutMatrix("maturity",'Maturity')
0 <== MLPutMatrix("cpnrate","CpnRate")
0 <== MLPutMatrix("sd",C3)
```

2. Execute MATLAB Financial Toolbox Cash flow and Time mapping function.

```
0 <== MLEvalString("md = x2mdate(maturity,0); sdm = x2mdate(sd,0)")
0 <== MLEvalString("[cfa, cfd] = cfamounts(cpnrate, sdm, md, 2)")
```

3. Transform date numbers to string cell array.

```
0 <== MLEvalString("i = find(isnan(cfd)); zcfd = cfd; zcfd(i) = 0; scfd=datestr(zcfd,2);")
0 <== MLEvalString("ccfd = num2cell(scfd,2); ccfd(i) = {'N/A'}; ccfd = reshape(ccfd, size(cfd));")
0 <== MLEvalString("ccfa = cfa; ccfa(i) = 0; alldates = ccfd(end, :);")
```

4. Transfer output data to Excel.

```
0 <== MLGetMatrix("ccfd", "i3")
0 <== MLGetMatrix("alldates", "i13")
0 <== MLGetMatrix("ccfa", "i14")
```

5. Plot the cash flow diagram.

```
0 <== 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.
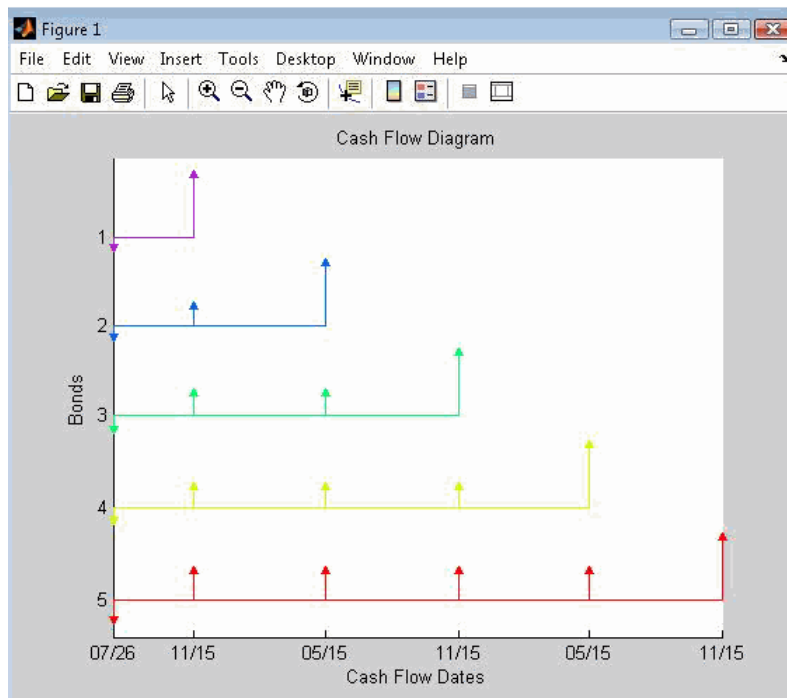
9 When you finish the example, close the figure window.

3

Function Reference

Link Management

matlabinit	Initialize Spreadsheet Link EX software and start MATLAB process
MLAutoStart	Automatically start MATLAB process
MLClose	End MATLAB process
MLOpen	Start MATLAB process
MLUseCellArray	Toggle MLPutMatrix to use MATLAB cell arrays

Data Management

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

MLStartDir   Specify MATLAB current working
                                    directory 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:	matlabsub(command, edat, inputs)
command	MATLAB command to evaluate. Enter the MATLAB command in double quotation marks, as `"command"`.
edat	Worksheet location where the function writes the returned date. `"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.
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 theMicrosoft 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   End 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.

Example   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

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`.
`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 NaNs 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:

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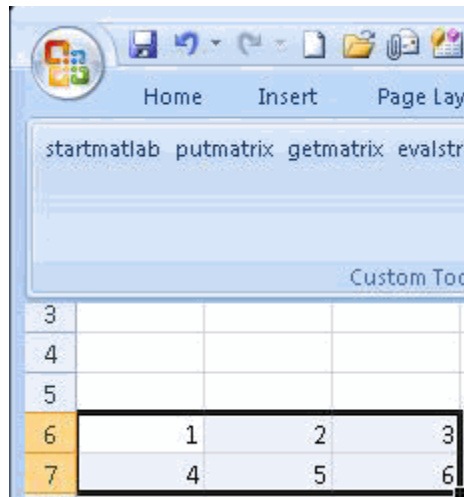
2 Click the **putmatrix** button on the Spreadsheet Link EX toolbar. A window appears that prompts you to specify the name of the MATLAB variable in which you want to store your data.
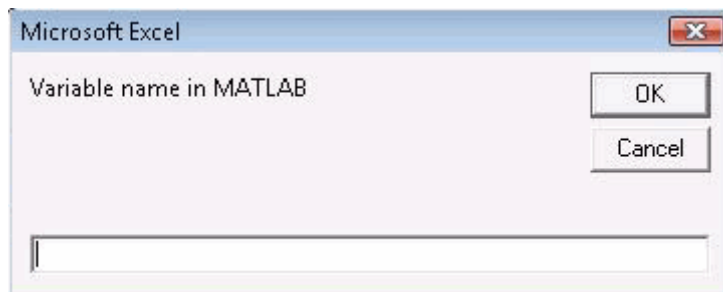
3 Enter newmatrix for the MATLAB variable name.

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

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
```

MLPutVar

```
        End Sub
```

See Also   MLGetVar

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`

MLStartDir

Purpose	Specify MATLAB current working directory after startup

Syntax

Worksheet:	MLStartDir(path)
Macro:	MLStartDir path

path	Specify the current MATLAB working directory after startup.

Description Sets the MATLAB working directory 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 directory.

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

Examples Set the MATLAB working directory to d:\work after startup:

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

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

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

See Also MLAutoStart

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`

MLUseFullDesktop

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

Syntax

Worksheet:	MLUseFullDesktop("yes")
	MLUseFullDesktop("no")
Macro:	MLUseFullDesktop "yes"
	MLUseFullDesktop "no"
"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

Error Messages and Troubleshooting

This appendix covers the following topics:

- "Worksheet Cell Errors" on page A-2
- "Microsoft® Excel Software Errors" on page A-5
- "Data Errors" on page A-8
- "Startup Errors" on page A-10
- "Audible Error Signals" on page A-11

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 1-12.
#NAME?	The function name is unrecognized. The excllink.xla add-in is not loaded, or the function name may be misspelled.	Be sure the excllink.xla add-in is loaded. See "Configuring the Spreadsheet Link EX Software" on page 1-5. Check the spelling of the function name. Correct typing errors.
#NONEXIST!	You referenced a nonexistent matrix in an MLGetMatrix or MLDeleteMatrix function. The matrix name may be misspelled.	Verify the spelling of the MATLAB matrix. Use the MATLAB whos 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 4, "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 MLGetMatrix followed by MatlabRequest, 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 MatlabRequest 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 Software Errors

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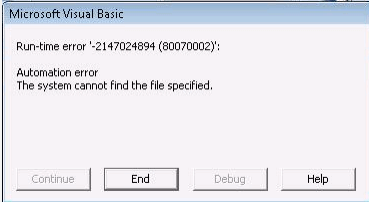
Excel Error Messages

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 `excllink.xla` is incorrect.	Click **OK**. The References window opens. Remove the check from `MISSING: excllink.xla`. Find `excllink.xla` 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 `MLGetMatrix` 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
`Spreadsheet Link EX license checkout failed!`	The license passcode that you entered was invalid.	Check that you entered the license passcode properly. If you 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

	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.	To correct this error, perform the following:
		
		1 Shut down all MATLAB and Excel instances.
		
		2 Open a Command Prompt window, and using `cd`, change to the `bin\win32` subdirectory of the MATLAB installation directory.
		
		3 Type the command:
		
		`.\matlab /regserver`
		
		4 When the MATLAB software session starts, close it. Using `/regserver` fixes the registry entries.
		
		5 Start an Excel software 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:
		
		`a = 3.14159`
		
		7 Enter the following formula in cell A1 of the open Excel worksheet:
		
		`=mlgetmatrix("a","a1")`
		
		8 The value `3.14159` appears in cell A1.

Data Errors

In this section...
"Matrix Data Errors" on page A-8
"Errors When Opening Saved Worksheets" on page A-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 `MLPutMatrix` or `MLAppendMatrix`.	Correct the syntax to remove quotation marks.
MATLAB matrix is empty ([ ]).	You referenced a nonexistent VBA variable in `MLPutVar`.	Correct the macro; you may have typed the variable name incorrectly.
VBA matrix is empty.	You referenced a nonexistent MATLAB variable in `MLGetVar`.	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:

9 Ignore the messages.

10 Close MATLAB figure windows.

11 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, and 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 Directory 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 software 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.

Examples

Use this list to find examples in the documentation.

Macro Examples

Financial Examples

Index