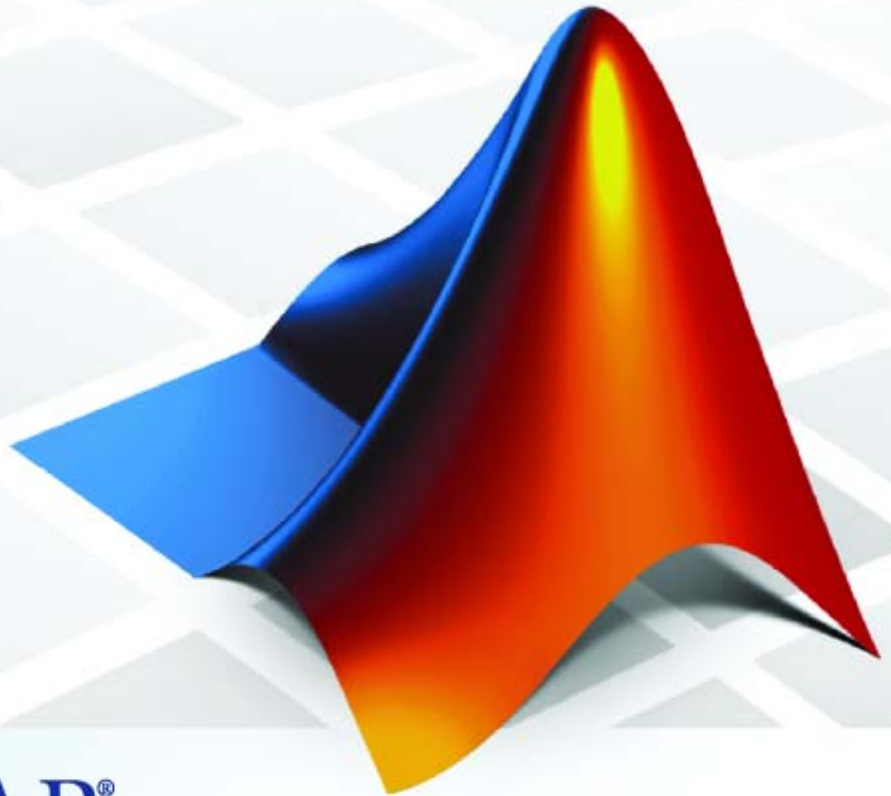


Excel® Link 3 User's Guide



MATLAB®

How to Contact The MathWorks



www.mathworks.com
comp.soft-sys.matlab
www.mathworks.com/contact_TS.html

Web
Newsgroup
Technical Support



suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
service@mathworks.com
info@mathworks.com

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
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.

Excel® Link User's Guide

© COPYRIGHT 1996–2007 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, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, SimBiology, SimHydraulics, SimEvents, and xPC TargetBox are registered trademarks and The MathWorks, the L-shaped membrane logo, Embedded MATLAB, and PolySpace are trademarks of The MathWorks, Inc.

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

Patents

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

Revision History

May 1996	First printing	New for Version 1.0
May 1997	Second printing	Updated for Version 1.0.3
January 1999	Third printing	Updated for Version 1.0.8 (Release 11)
September 2000	Fourth printing	Updated for Version 1.1.2
April 2001	Fifth printing	Updated for Version 1.1.3
July 2002	Sixth printing	Updated for Version 2.0 (Release 13)
September 2003	Online only	Updated for Version 2.1 (Release 13SP1)
June 2004	Online only	Updated for Version 2.2 (Release 14)
September 2005	Online only	Updated for Version 2.3 (Release 14SP3)
March 2006	Online only	Updated for Version 2.3.1 (Release 2006a)
September 2006	Online only	Updated for Version 2.4 (Release 2006b)
September 2006	Seventh printing	Updated for Version 2.4 (Release 2006b)
March 2007	Online only	Updated for Version 2.5 (Release 2007a)
September 2007	Online only	Updated for Version 3.0 (Release 2007b)

Getting Started

1

What Is Excel Link?	1-2
Installing Excel Link	1-3
System Requirements	1-3
Installing Excel Link	1-3
Files and Directories Created by the Installation	1-3
Modifying Your System Path	1-4
Configuring Excel Link	1-5
Configuring Excel to Work with Excel Link	1-5
Setting Excel Link Preferences	1-6
Starting and Stopping Excel Link	1-8
Automatically Starting Excel Link	1-8
Manually Starting Excel Link	1-8
Connecting to an Existing MATLAB Session	1-8
Stopping Excel Link	1-9
About Functions	1-10
How Excel Link Functions Differ from Microsoft Excel Functions	1-10
Types of Excel Link Functions	1-10
Using Worksheets	1-11
Working with Arguments in Excel Link Functions	1-13
Using the MATLAB Function Wizard for Excel Link	1-13
Create Macros for Excel Link Functions	1-17
Dates	1-20
Information for International Users	1-22

Solving Problems with Excel Link

2

About the Examples	2-2
Data Regression and Curve Fitting	2-3
Worksheet Version	2-3
Macro Version	2-6
Data Interpolation	2-9
Stock Option Pricing Using the Binomial Model	2-13
Calculating and Plotting the Efficient Frontier of Financial Portfolios	2-16
Bond Cash Flow and Time Mapping	2-20

Functions — By Category

3

Link Management Functions	3-2
Data Management Functions	3-3

4

Error Messages and Troubleshooting

A

Excel Cell Error Messages	A-2
Error Messages	A-6
Audible Error Signals	A-10
Data Errors	A-11
Matrix Data Errors	A-11
Errors When Opening Saved Worksheets	A-11

Examples

B

Macro Examples	B-2
Financial Examples	B-2

Index

Getting Started

What Is Excel Link? (p. 1-2)

How Excel Link works with both
MATLAB® and Microsoft Excel®

Installing Excel Link (p. 1-3)

How to install Excel Link

Configuring Excel Link (p. 1-5)

How to configure Excel and MATLAB
to work with Excel Link

Starting and Stopping Excel Link
(p. 1-8)

How to start and stop Excel Link

About Functions (p. 1-10)

Discusses the two kinds of Excel
Link functions: Link Management
and Data Management

Dates (p. 1-20)

How dates are represented in Excel
vs. Excel Link

Information for International Users
(p. 1-22)

About Windows regional settings

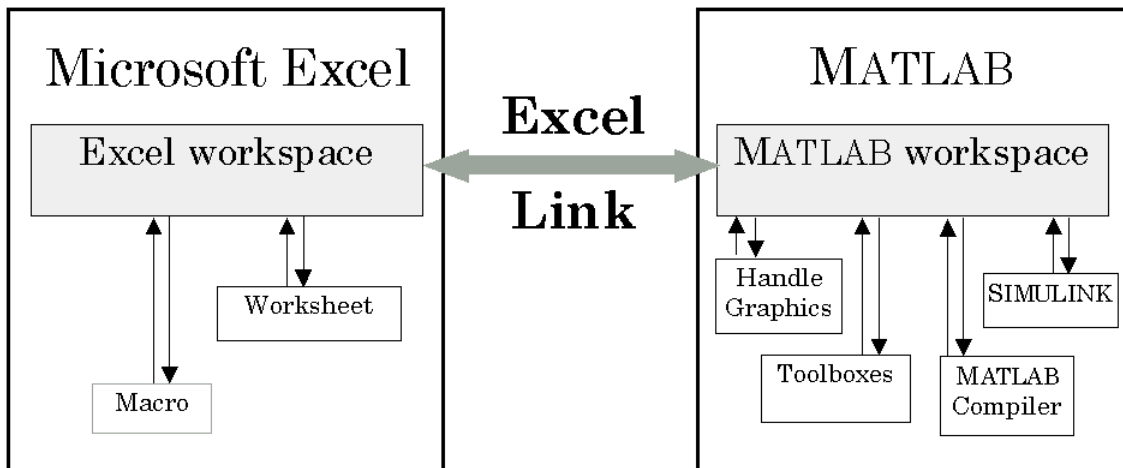
What Is Excel Link?

Excel® Link is a software add-in that integrates Microsoft Excel and MATLAB in a Microsoft Windows®-based computing environment. It positions Excel as a front end to MATLAB. By connecting Excel with the MATLAB workspace, you can access the numerical, computational, and graphical power of MATLAB from Excel worksheet and macro programming tools.

Excel Link lets you use Excel Link functions from 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, Excel Link is powerful in its simplicity.

Note The terms *worksheet* and *spreadsheet* are used interchangeably throughout this document.

Excel Link 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 Excel Link

In this section...

“System Requirements” on page 1-3

“Installing Excel Link” 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 Excel Link, see <http://www.mathworks.com/products/excellink/requirements.html>.

Excel Link requires MATLAB for Windows. 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.

Installing Excel Link

Install Windows and Excel *before* you install MATLAB and Excel Link.

To install Excel Link, follow the instructions in the MATLAB installation documentation. Select the **Excel Link** check box when you select MATLAB components to install.

Files and Directories Created by the Installation

The Excel Link installation program creates the subdirectory under *matlabroot/toolbox/*, where *matlabroot* is the directory where MATLAB is installed on your system. The *exlink* directory contains the following files:

- `exclink.xla`: The Excel Link add-in
- `ExlISamp.xls`: Excel Link samples described in this manual

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

Excel Link 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

For Excel Link to function properly, you must add the following directories to your system path. For more information on how to do this, consult your Windows documentation or your system administrator.

- On all supported operating systems, add `C:\MATLAB\bin` to your path.
- On Windows 2000, add `C:\MATLAB\bin`, `C:\Winnt\system`, and `C:\Winnt\system32` to your path.

Configuring Excel Link

In this section...

“Configuring Excel to Work with Excel Link” on page 1-5

“Setting Excel Link Preferences” on page 1-6

Configuring Excel to Work with Excel Link

After you have installed Excel Link, you are ready to configure Excel. You need to do these steps only once.

Note These instructions are for Excel 2003 and earlier versions. For instructions on how to configure Excel 2007 to Work with Excel Link, see The MathWorks Support Web site.

- 1 Start Microsoft Excel.
- 2 Click **Tools > Add-Ins > Browse**.
- 3 Find and select the Excel Link add-in `excllink.xla` under `matlabroot/toolbox/exlink`.

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

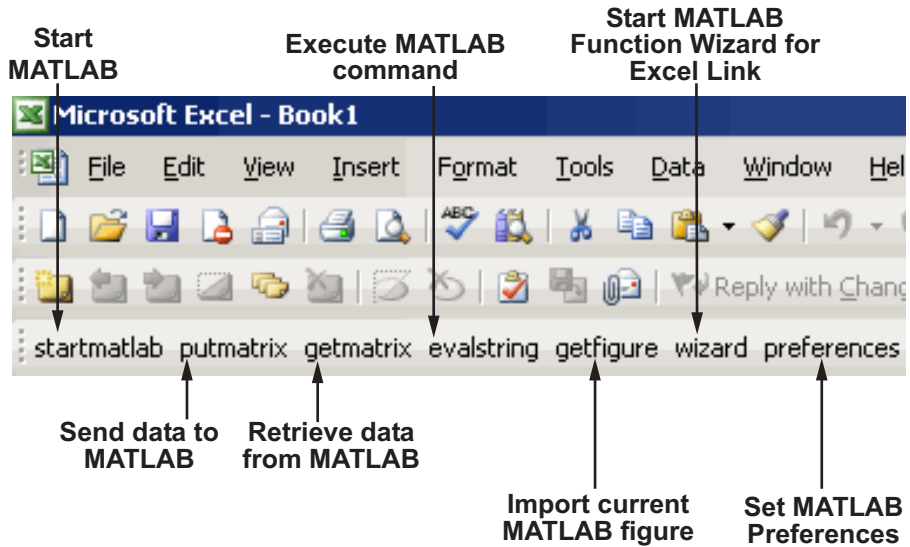
- 4 Click **OK**.

The Excel Link add-in loads now and with each subsequent invocation of Excel.

Note the **MATLAB Command Window** button on the Windows taskbar.



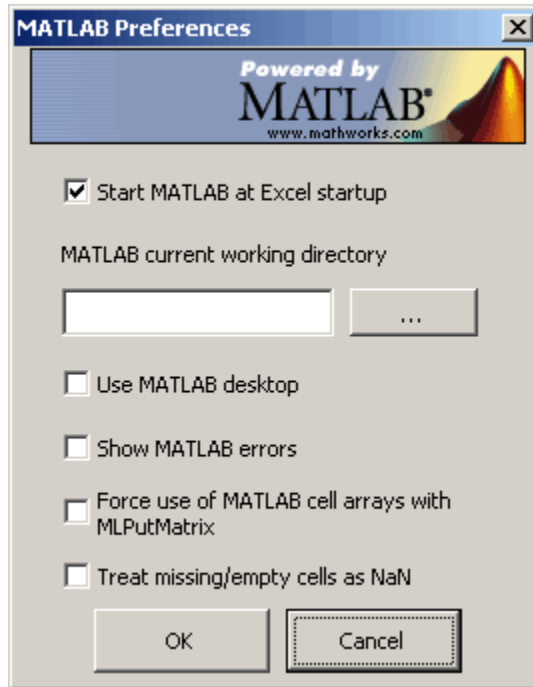
Note the Excel Link toolbar on your Excel worksheet.



Excel Link is now ready for your use.

Setting Excel Link Preferences

Use the Preferences dialog box to set Excel Link 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 MATLAB automatically when Excel starts.
- **MATLAB current working directory** enables you to specify the current working directory for your MATLAB session at startup.
- **Use MATLAB desktop** starts the entire MATLAB desktop, including the current directory, workspace, command history and command window panes, when Excel starts.
- **Force use of MATLAB cell arrays with MLPutMatrix** enables you to set the MLPutMatrix function to use cell arrays for transfer of data between Excel Link and MATLAB.
- **Treat missing/empty cells as NaN** sets data in missing or empty cells to NaN or zero.

Starting and Stopping Excel Link

In this section...

“Automatically Starting Excel Link” on page 1-8

“Manually Starting Excel Link” on page 1-8

“Connecting to an Existing MATLAB Session” on page 1-8

“Stopping Excel Link” on page 1-9

Automatically Starting Excel Link

When installed and configured according to the instructions in “Configuring Excel Link” on page 1-5, Excel Link and MATLAB automatically start when you start Excel.

Manually Starting Excel Link

To start Excel Link and MATLAB manually from Excel:

- 1 Click **Tools > Macro**.
- 2 Enter `matlabinit` into the **Macro Name/Reference** box.

For more information about the `matlabinit` function, see Chapter 3, “Functions — By Category”.

- 3 Click **Run**.

Watch for the MATLAB Command Window button on the Windows taskbar.



Connecting to an Existing MATLAB Session

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

To add the /automation option to the command line:

- 1** Right-click your shortcut to MATLAB.
- 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**.

Stopping Excel Link

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

About Functions

In this section...
“How Excel Link Functions Differ from Microsoft Excel Functions” on page 1-10
“Types of Excel Link Functions” on page 1-10
“Using Worksheets” on page 1-11
“Working with Arguments in Excel Link Functions” on page 1-13
“Using the MATLAB Function Wizard for Excel Link” on page 1-13
“Create Macros for Excel Link Functions” on page 1-17

How Excel Link Functions Differ from Microsoft Excel Functions

- Excel Link functions *perform an action*, while Microsoft Excel functions *return a value*. Keep this distinction in mind as you use Excel Link.
- Excel Link function names *are not* case sensitive; that is, `MLPutMatrix` and `mputmatrix` 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 Excel Link functions.

Types of Excel Link Functions

Excel Link provides functions to manage the link and to manipulate data between Excel and MATLAB without your ever needing to leave the Excel environment. You can invoke functions as worksheet cell formulas or in macros. With Excel Link, Microsoft Excel becomes an easy-to-use data-storage and application-development front end for MATLAB, which is a powerful computational and graphical processor.

Excel Link provides two types of functions: link management functions and data management functions.

Link management functions initialize, start, and stop Excel Link and MATLAB. Any link management function other than `matlabinit` can be invoked as a worksheet cell formula or in macros. This function must be invoked from the Excel **Tools Macro** menu or in macro subroutines.

Data management functions copy data between Excel and MATLAB and execute MATLAB commands from Excel. Any data management function other than `MLPutVar` and `MLGetVar` can be invoked as a worksheet cell formula or in macros. These functions can be invoked only in macros.

For more information about Excel Link functions, see Chapter 3, “Functions — By Category”.

Using Worksheets

Entering Functions into Worksheet Cells

Excel Link functions expect A1-style worksheet cell references; that is, columns designated with letters and rows with numbers. This is 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 Excel Link 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 Excel Link functions, see “Working with Arguments in Excel Link Functions” on page 1-13.

Note Do not use the Excel Function Wizard, as it can generate unpredictable results.

After an Excel Link function successfully executes as a worksheet formula, the cell contains the value 0. While a function is executing, the cell may 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 provides a useful confirmation for lengthy operations.

Automatic Calculation Mode Vs. Manual Calculation Mode

Excel Link functions are most effective in automatic calculation mode. To “automate” the recalculation of an Excel Link function, add to it some 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 not to create endless recalculation loops.

To use `MLGetMatrix` in manual calculation mode, enter the function into a cell, press **F2**; then press **Enter** to execute it.

Note Pressing **F9** to recalculate a worksheet affects only Excel functions. It does not operate on Excel Link functions.

If you use explicit cell addresses in a function and then later insert or delete rows or columns, or move or copy the function to another cell, you need to edit the function arguments to reference the new cell address. Excel Link does not automatically adjust cell addresses in functions.

Working with Arguments in Excel Link Functions

This section describes tips for managing variable-name arguments and data-location arguments in Excel Link Functions.

Variable-name Arguments

- You can *directly* or *indirectly* specify a variable-name argument in most Excel Link 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.

Data-location Arguments

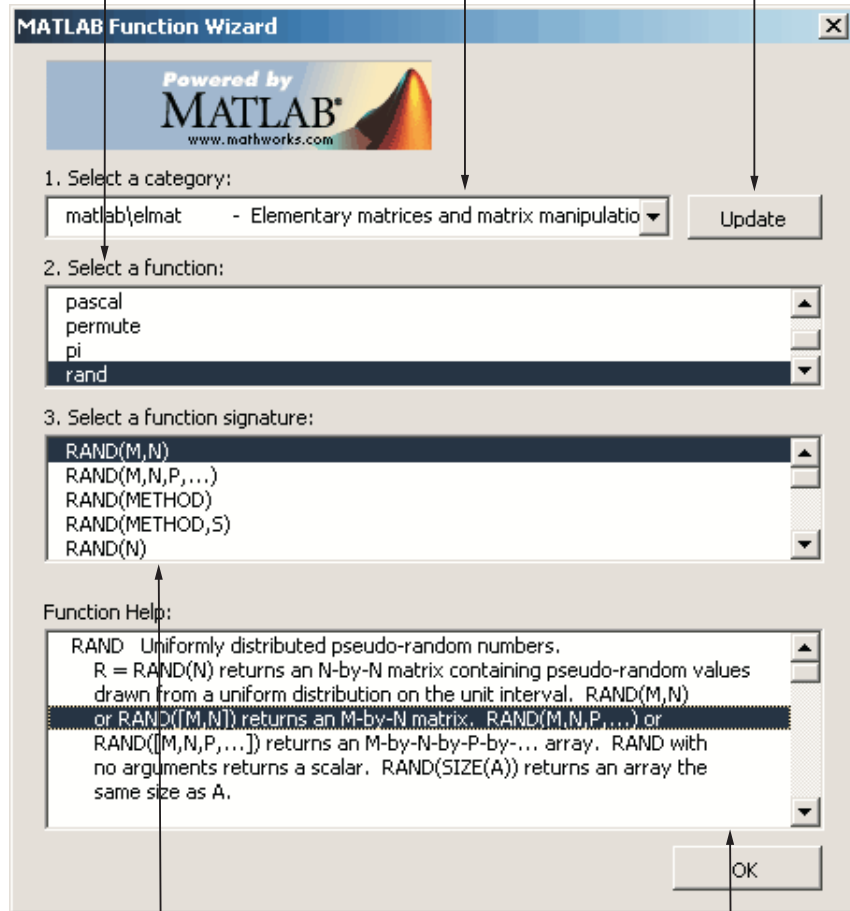
- A data-location argument must be a worksheet cell address or range name.
- Do not enclose a data-location argument in quotation marks (except in `MLGetMatrix`, which has unique argument conventions).
- A data-location argument can include a worksheet number; for example, `Sheet3!B1:C7` or `Sheet2!OUTPUT`.

Note You can use virtually any special character as part of a worksheet name if you embed the sheet name within single quotation marks (' ') when referencing it in `MLGetMatrix` or `MLPutMatrix`.

Using the MATLAB Function Wizard for Excel Link

The MATLAB Function Wizard for Excel Link allows you to browse MATLAB directories and run functions from within Excel.

List functions available for specified directory/category Display list of MATLAB working directories and available function categories Refresh directory/category list



Select function signature and enter formula into specified spreadsheet cell

Display help for given function signature

Use The MATLAB Function Wizard for Excel Link to do the following:

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

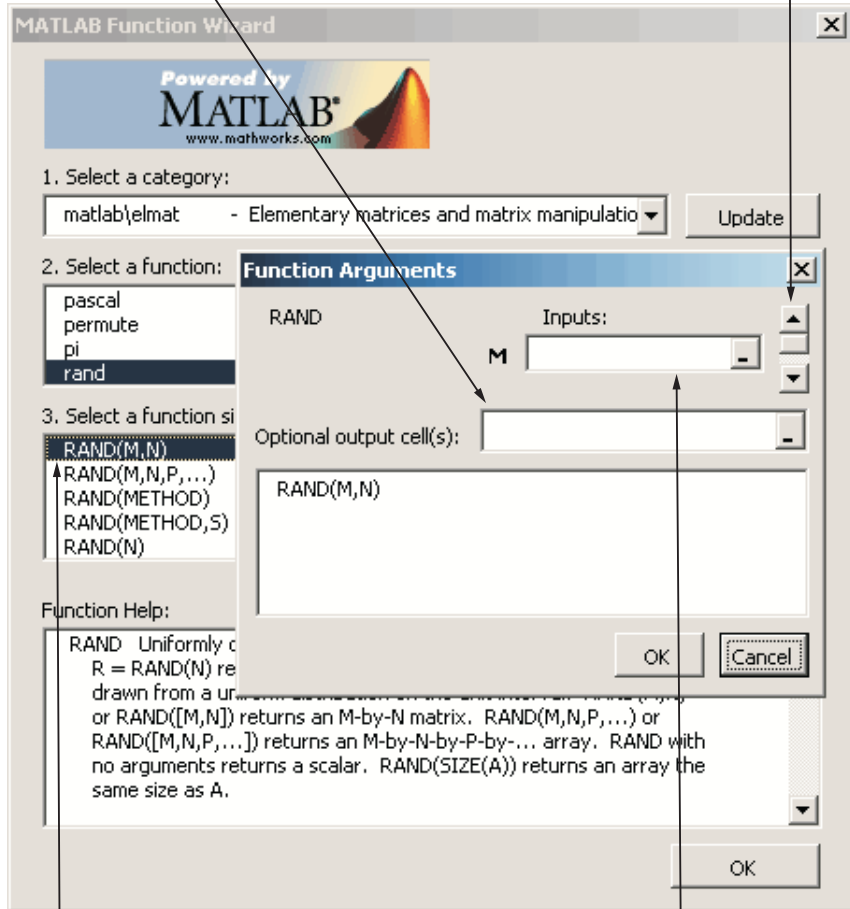
After you select a directory or category, available functions for that directory or category display in the **Select a function** field. Click a function name to select it.

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



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 Excel Link 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)
```

If you specify a target range of cells using the **Optional output cell(s)** field in the Function Arguments dialog box, the selected function appears in the current spreadsheet cell using `MATLABSUB`, with an additional argument to indicate where to write the function's output. In the following example, the target cell is B2 and the input to the rand function comes from A2:

```
=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, the help header for the specified function signature appears in the **Function Help** field.

Create Macros for Excel Link Functions

To create macros that use Excel Link functions, you must first configure Excel to reference the functions from the Excel Link add-in. Use the following steps to do this.

- 1 From the Visual Basic environment, click **Insert > Module**.
- 2 When the Module pane opens, click **Tools > References**.
- 3 In the References dialog box, select the **ExcelLink** check box and click **OK**.

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

Example: Using MLGetMatrix in a Macro Subroutine

To use `MLGetMatrix` in a macro subroutine, enter `MatlabRequest` on the line after `MLGetMatrix`. `MatlabRequest` initializes internal Excel Link variables and enables `MLGetMatrix` to function in a subroutine. For example:

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

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

Example: Running Excel Link Functions from the Visual Basic Editor

In this example, you use VBA code to execute MATLAB commands, send MATLAB data to Excel, and display the results in an Excel dialog box.

Follow these steps to run this example:

- 1** Start Excel.
- 2** Initialize MATLAB by clicking the **startmatlab** button in the Excel Link toolbar or by running the `matlabinit` function.
- 3** Enable the Excel Link Add-in:
 - Click **Tools > Add-ins**.
 - Select the **Excel Link 3.0 for use with MATLAB** check box.
 - Click **OK**.
- 4** Enable Excel Link as a Reference in the Excel Visual Basic Editor:
 - Click **Tools > Macro > Visual Basic Editor**.
 - In the Visual Basic Editor window, click **Tools > References**.

- In the References — VBAProject dialog box, select the **ExcelLink** check box.
 - Click **OK**.
- 5** In the Visual Basic Editor, create a new module. To do this, right-click the **Microsoft Excel Objects** folder in the **Project — VBAProject** tree browser and select **Insert > Module** from the context menu.
- 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

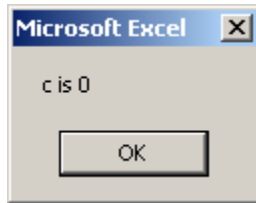
    Sheets("Sheet1").Select
    Range("A10").Select
    MLGetMatrix "d", ActiveCell.Address
```

```
MatlabRequest
```

```
End Sub
```

7 To run the code, press **F5** or click **Run > Run Sub/UserForm**.

The following dialog box appears.



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

Dates

Default 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 Excel.

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 MATLAB. Therefore, the difference in dates between Excel and MATLAB is a constant, 693960 (729160 minus 35200).

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

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

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

Note Dates are stored internally in Excel as numbers and are not affected by locale.

Information for International Users

This document uses Excel with an English (United States) Windows regional setting for illustrative purposes. If you use Excel Link 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 the Excel Link commands with a semicolon (;) or other operator.

Please consult your Microsoft Windows documentation to determine which regional setting differences exist among various international versions.

Solving Problems with Excel Link

About the Examples (p. 2-2)

How to run these examples online in Excel Link

Data Regression and Curve Fitting (p. 2-3)

Builds a mathematical model of a data set and uses an Excel worksheet to organize and display the data

Data Interpolation (p. 2-9)

Uses an Excel worksheet to organize and display input data and the interpolated output data

Stock Option Pricing Using the Binomial Model (p. 2-13)

Uses Excel Link, Financial Toolbox, and the binomial model to price options

Calculating and Plotting the Efficient Frontier of Financial Portfolios (p. 2-16)

Uses Excel Link and Financial Toolbox to analyze three portfolios, using rates of return for six time periods

Bond Cash Flow and Time Mapping (p. 2-20)

Uses Excel Link and Financial Toolbox to compute a set of cash flow amounts and dates, given a portfolio of five bonds

About the Examples

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

These examples are included with Excel Link. To run them:

- 1 Start Excel, Excel Link, and MATLAB.
- 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 functions in Financial Toolbox. Financial Toolbox in turn requires Statistics Toolbox and Optimization Toolbox.

Data Regression and Curve Fitting

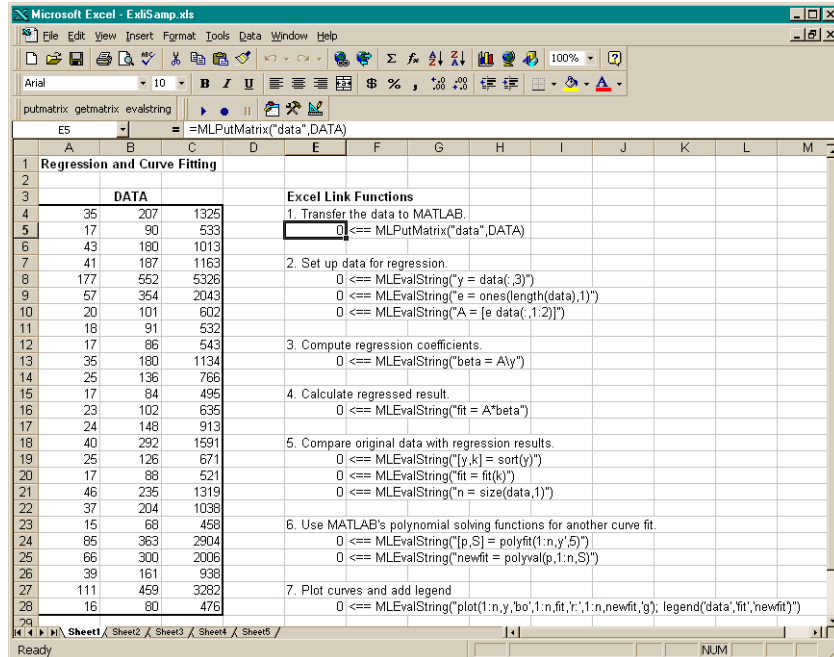
In this section...
“Worksheet Version” on page 2-3
“Macro Version” on page 2-6

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 provides many powerful yet easy-to-use matrix operators and functions to simplify the task.

This example demonstrates both data regression and curve fitting. It also executes the same example in a worksheet version and a macro version. The example uses Excel worksheets to organize and display the data. Excel Link functions copy the data to MATLAB and execute MATLAB computational and graphic functions. The macro version also returns output data to an Excel worksheet.

Worksheet Version

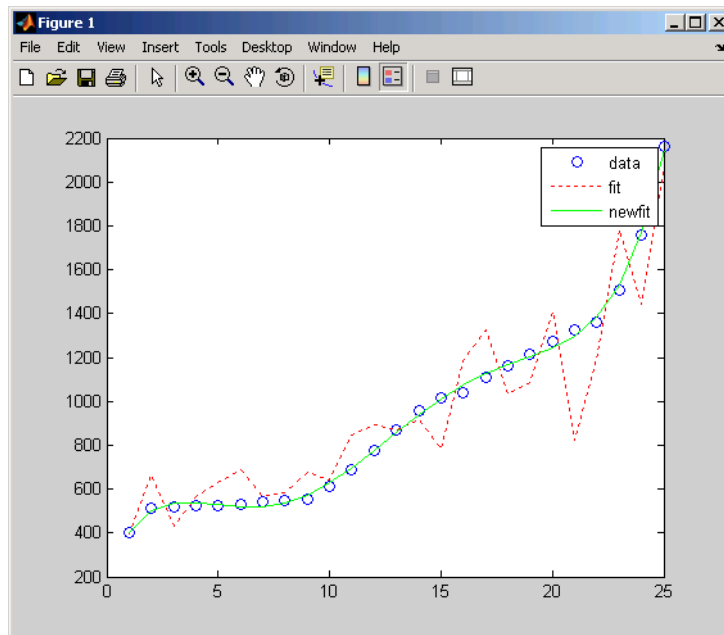
- 1 Click the **Sheet1** tab on the Ex1iSamp.xls window. The worksheet for this example appears.



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 Excel Link function that copies the sample data set to MATLAB. 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 Excel Link functions tell MATLAB to regress the third column of data on the other two columns. They create a single vector y containing the third-column data, and a new three-column matrix A consisting 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 backslash 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 compare the original data with fit; sort the data in increasing order and apply the same permutation to fit; and 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); and adds a legend. Data plots.

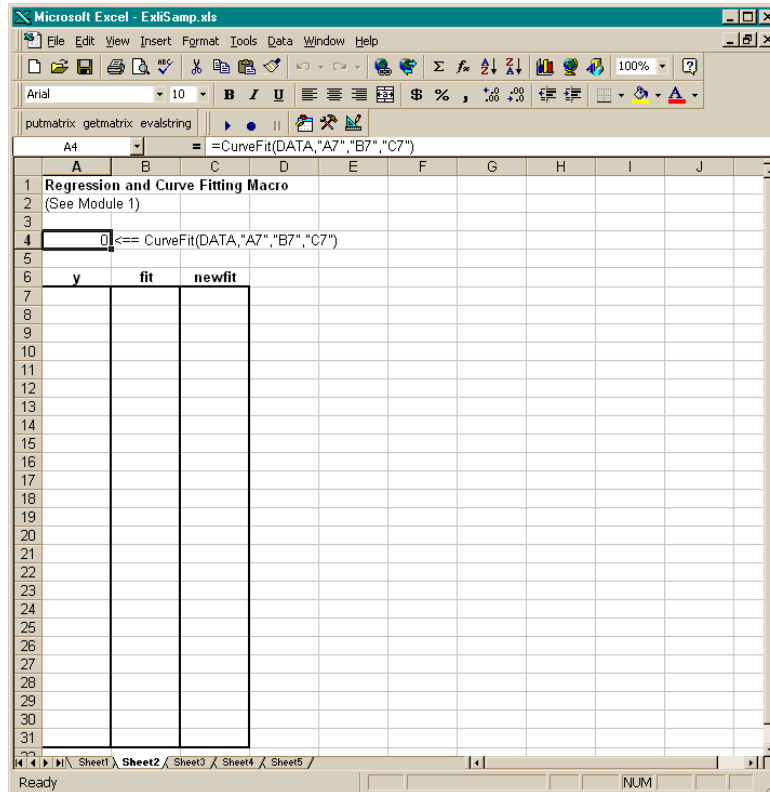


Since the data is closely correlated but not exactly linearly dependent, the fit curve (dashed line) shows a close, but not an exact, fit. The fifth-degree polynomial curve, `newfit`, represents a more accurate mathematical model for the data.

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

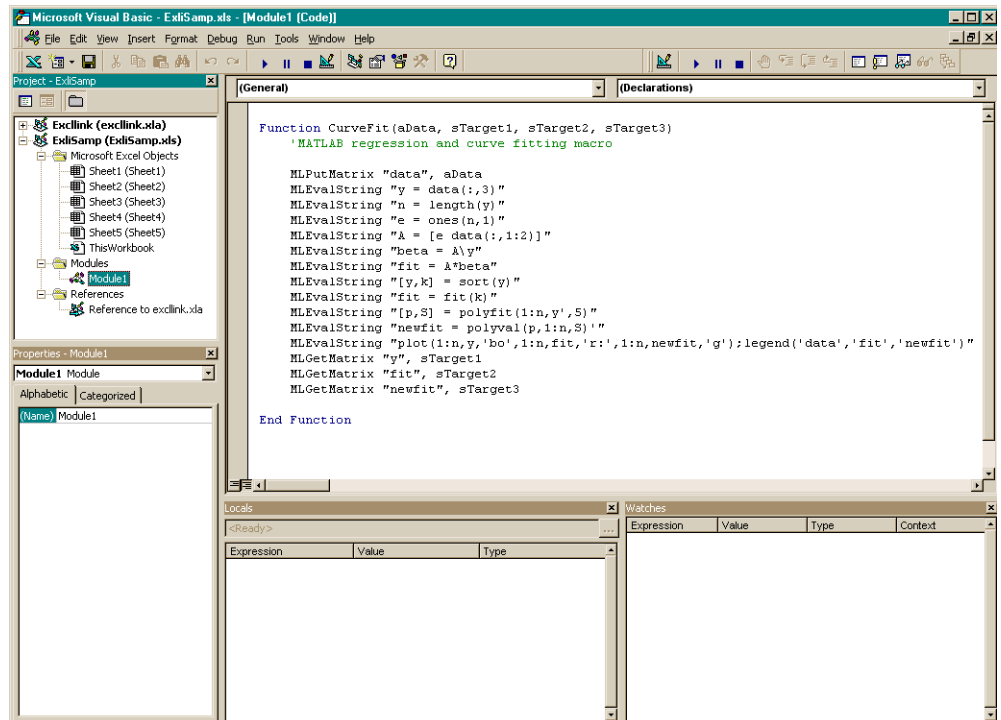
Macro Version

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



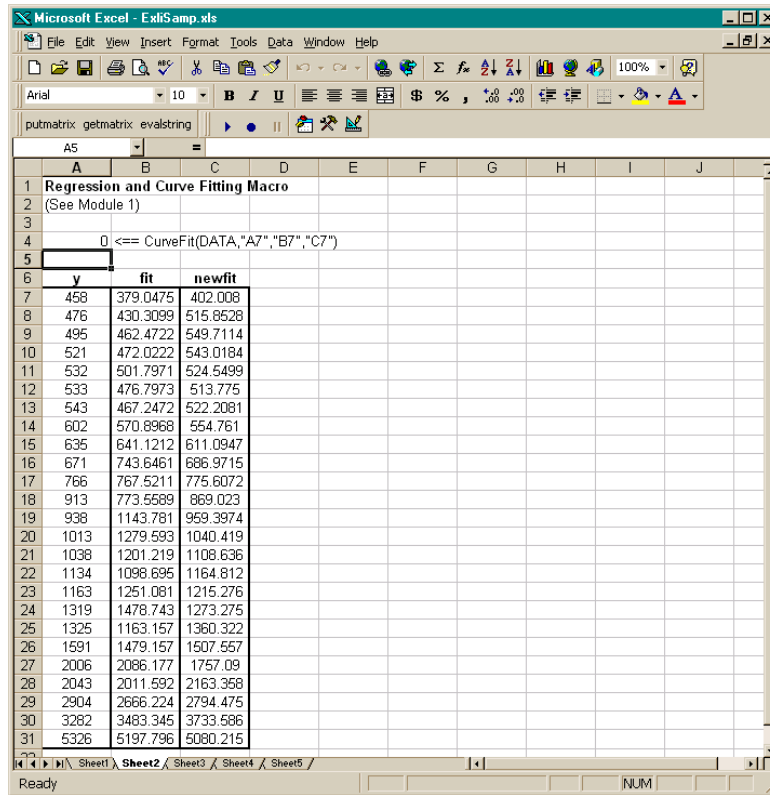
- 2 Make cell A4 the active cell, but do not execute it yet.

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



- 3 While this module is open, click **Tools > References**. In the **References** dialog box, make sure that the **exclink.xls** check box is selected. If not, select the check box and click **OK**.
- 4 In cell A4 of **Sheet2**, press **F2**; then press **Enter** to execute the CurveFit macro. The macro executes the same functions as the worksheet example (in a slightly different order), including plotting the graph. In addition, it uses the MLGetMatrix function in the CurveFit macro to copy the original data y (sorted), the corresponding regressed data fit, and the polynomial data newfit, to the worksheet.

2 Solving Problems with Excel Link



The screenshot shows the Microsoft Excel interface with a macro named "Regression and Curve Fitting Macro" in cell A1. The macro code in cell A2 is: `0 <== CurveFit(DATA,"A7","B7","C7")`. Below the macro, a table with three columns: "y", "fit", and "newfit" is displayed. The table contains 26 rows of data. The status bar at the bottom indicates "Ready" and "NUM".

y	fit	newfit
458	379.0475	402.008
476	430.3099	515.8528
495	462.4722	549.7114
521	472.0222	543.0184
532	501.7971	524.5499
533	476.7973	513.775
543	467.2472	522.2081
602	570.8968	554.761
635	641.1212	611.0947
671	743.6461	686.9715
766	767.5211	775.6072
913	773.5589	869.023
938	1143.781	959.3974
1013	1279.593	1040.419
1038	1201.219	1108.636
1134	1098.695	1164.812
1163	1251.081	1215.276
1319	1478.743	1273.275
1325	1163.157	1360.322
1591	1479.157	1507.557
2006	2086.177	1757.09
2043	2011.592	2163.358
2904	2666.224	2794.475
3282	3483.345	3733.586
5326	5197.796	5060.215

When you finish the example, close the figure window.

Data Interpolation

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 provides a number of interpolation functions that 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 an Excel worksheet to organize and display the original data and the interpolated output data. Excel Link functions copy the data to and from MATLAB, execute the MATLAB interpolation function, and invoke MATLAB graphics to display the interpolated data in a three-dimensional color surface.

- 1 Click the **Sheet3** tab on `Ex1iSamp.xls`. The worksheet for this example appears.

2 Solving Problems with Excel Link

The screenshot shows a Microsoft Excel window titled "Microsoft Excel - ExiSamp.xls". The worksheet contains the following data and code:

Original Data			Interpolated Values																	
Time	Temp	Volume	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	
5	0.025	69.00	2504.98																	
6	0.050	69.05	2535.07																	
7	0.075	69.07	2562.91																	
8	0.100	69.09	2575.74																	
9	0.125	69.20	2606.16																	
10	0.150	69.50	2628.58																	
11	0.175	69.85	2681.38																	
12	0.200	69.22	2712.06																	
13	0.225	70.08	2767.52																	
14	0.250	70.33	2805.94																	
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	3099.12																	
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			0.6																	

Excel Link Functions

- 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)
- Transfer interpolation data points to MATLAB.
 - 38 0 <:= MLPutMatrix("Xa", E7:E30)
 - 40 0 <:= MLPutMatrix("Ta", F6:T6)
- Execute MATLAB data interpolation function.
 - 42 0 <:= MLEvalString("D4, T1, V1) = griddata(X,T,V,Xa,Ta,'invdist')")
- Transpose output data matrix and transfer data to Excel.
 - 45 0 <:= MLEvalString("TV = V1;")
 - 47 0 <:= MLGetMatrix("TV", "F7")
- Plot interpolated data and label the figure.
 - 50 0 <:= MLEvalString("surf(X1, T1, V1);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.

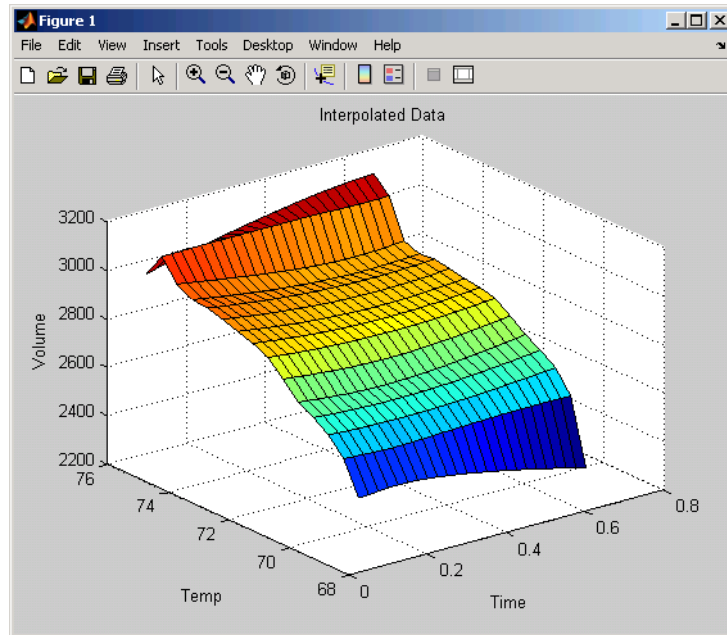
- 2 Make A33 the active cell. Press **F2**; then press **Enter** to execute the Excel Link function that passes the Time, Temp, and Volume labels to MATLAB.
- 3 Make A34 the active cell. Press **F2**; then press **Enter** to execute the Excel Link function that copies the original time data to MATLAB.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

	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
2																
3		Interpolated Values														
4																
5		Temp														
6	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	0.025	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
8	0.05	2507.26	2635.76	2704.79	2746.66	2775.96	2846.35	2907.00	2934.98	2955.07	2976.69	2993.64	2999.35	3034.49	3126.43	3036.68
9	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
10	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
11	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
12	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
13	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
14	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
15	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
16	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
17	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
18	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
19	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
20	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
21	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
22	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
23	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
24	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
25	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
26	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
27	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
28	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
29	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
30	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
34																

- 7 Execute the function in cell A50. MATLAB 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.

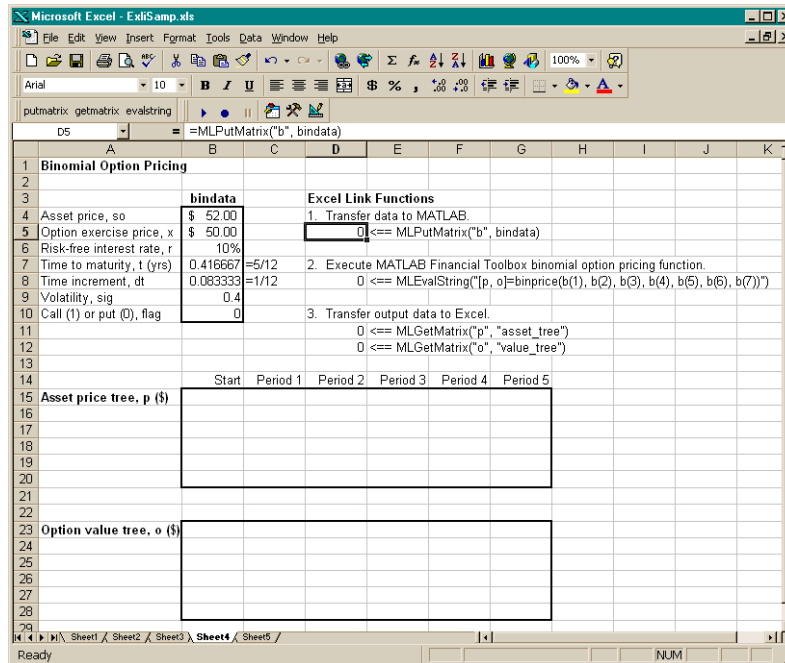
Stock Option Pricing Using the Binomial Model

Financial Toolbox provides several 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, that prices can move to only two values, one up and one down, over any short time period. Plotting the two values, and then the subsequent two values each, and then the subsequent two values each, and so on over time, is known as building a binomial tree.

This example uses the Excel worksheet to organize and display input and output data. Excel Link functions copy data to a MATLAB matrix, calculate the prices, and return data to the worksheet.

Note This example requires Financial Toolbox.

- 1 Click the **Sheet4** tab on `Ex1iSamp.xls` to open the worksheet for this example.



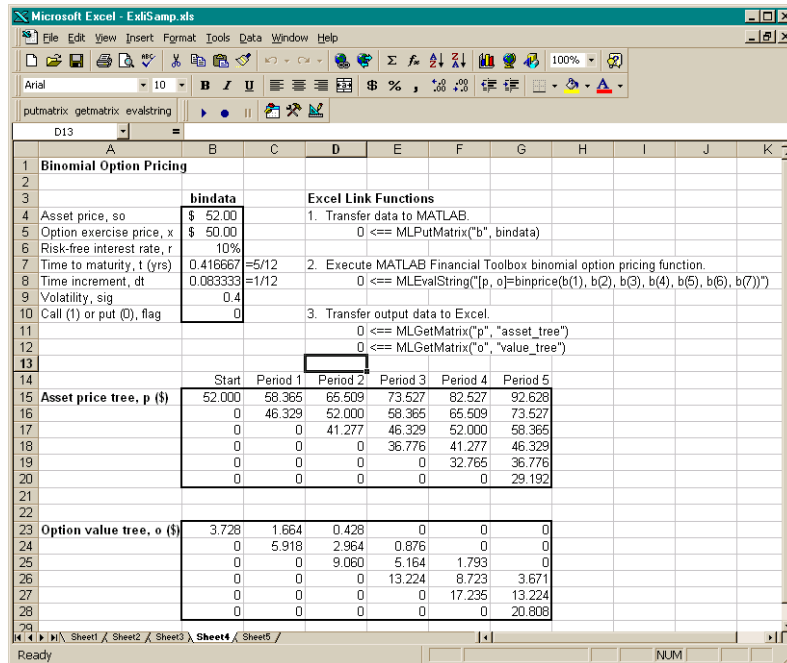
The worksheet contains three named ranges:

- 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 Excel Link function that copies the asset data to MATLAB.

3 Move to D8 and execute the function that computes the binomial prices, then execute the functions in D11 and D12 to copy the price data to Excel.

The worksheet looks like this.



Read the asset price tree this way: 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.

- 4 Try changing the data in B4:B10 and reexecuting the Excel Link functions. Note, however, that if you increase the time to maturity (B7) or change the time increment (B8), you may need to enlarge the output tree areas.
- 5 When you finish the example, close the figure window.

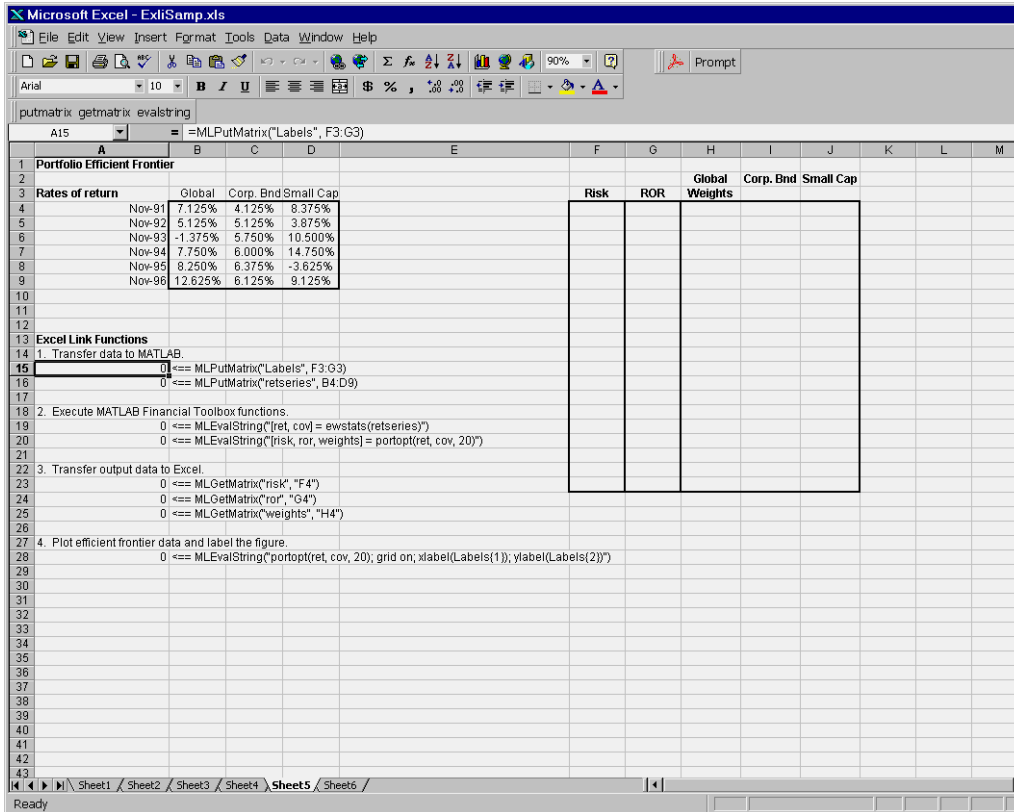
Calculating and Plotting the Efficient Frontier of Financial Portfolios

MATLAB and Financial Toolbox provide functions that 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. Excel and Excel Link 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 Financial Toolbox.

- 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** to execute the Excel Link function that transfers the labels describing the outputs to be computed by MATLAB.
- 3 Make A16 the active cell to copy the portfolio return data to MATLAB.
- 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 Excel Link functions in A23, A24, and A25 to copy the output data to Excel.

The worksheet looks like this.

2 Solving Problems with Excel Link

Microsoft Excel - ExlSamp.xls

File Edit View Insert Format Tools Data Window Help

putmatrix getmatrix evalstring

A25 =MLGetMatrix("weights", "H4")

Portfolio Efficient Frontier				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	Excel Link Functions				2.042%	6.365%	33.3%	38.4%	28.3%
14	1. Transfer data to MATLAB				2.241%	6.445%	37.0%	32.0%	31.1%
15	0 <= MLPutMatrix("Labels", F3:G3)				2.443%	6.525%	40.6%	25.6%	33.8%
16	0 <= MLPutMatrix("retseries", B4:D9)				2.646%	6.605%	44.3%	19.1%	36.6%
17					2.850%	6.685%	48.0%	12.7%	39.3%
18	2. Execute MATLAB Financial Toolbox functions.				3.055%	6.766%	51.6%	6.3%	42.1%
19	0 <= MLEvalString("ret, cov) = ewstats(retseries)")				3.262%	6.846%	55.0%	0.0%	45.0%
20	0 <= MLEvalString("risk, ror, weights) = portopt(ret, cov, 20)")				3.620%	6.926%	41.3%	0.0%	58.7%
21					4.213%	7.006%	27.5%	0.0%	72.5%
22	3. Transfer output data to Excel.				4.955%	7.086%	13.8%	0.0%	86.2%
23	0 <= MLGetMatrix("risk", "F4")				5.791%	7.167%	0.0%	0.0%	100.0%
24	0 <= MLGetMatrix("ror", "G4")								
25	0 <= MLGetMatrix("weights", "H4")								
26									
27	4. Plot efficient frontier data and label the figure.								
28	0 <= MLEvalString("portopt(ret, cov, 20); grid on; xlabel(Labels{1}); ylabel(Labels{2})")								

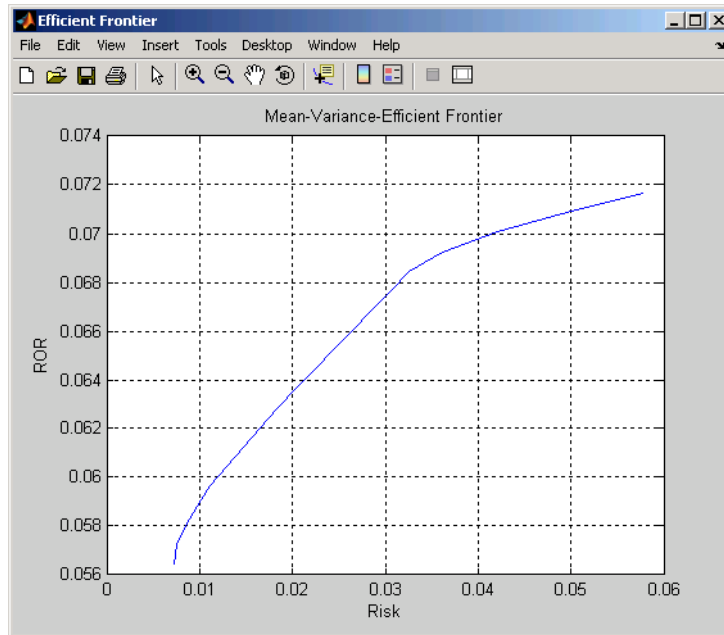
Sheet1 / Sheet2 / Sheet3 / Sheet4 / Sheet5 / Sheet6 /

Ready

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 in MATLAB.



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.

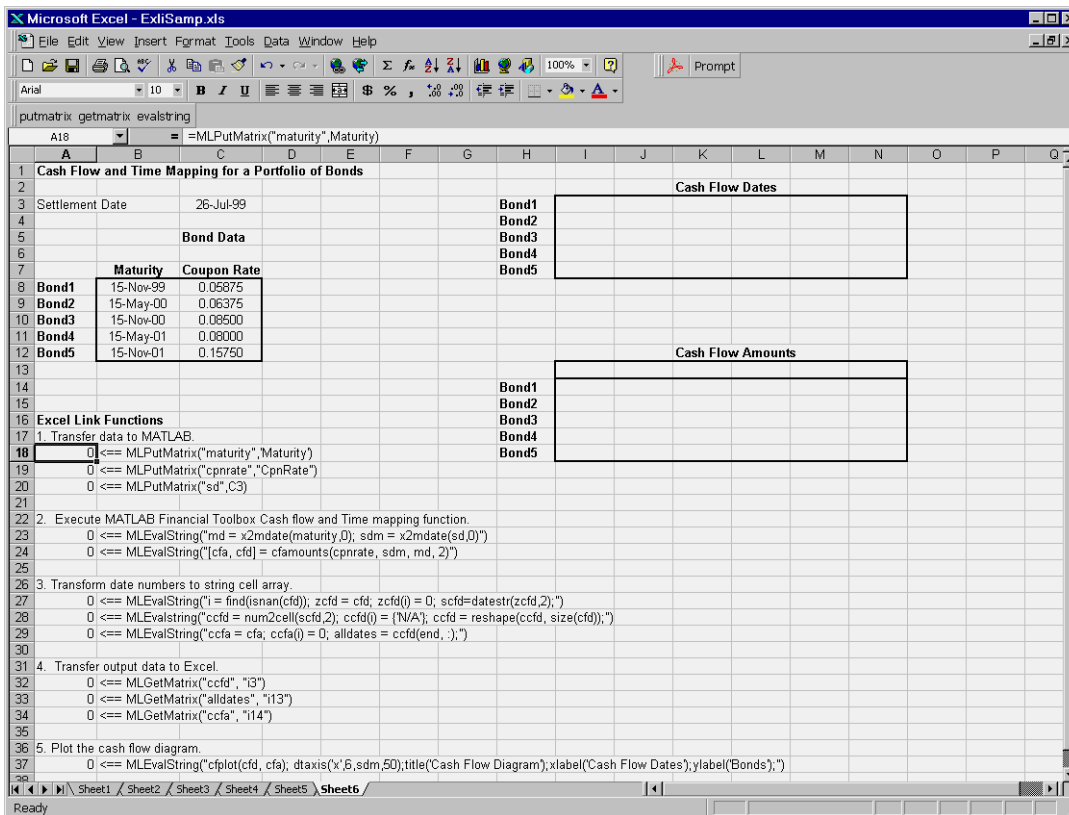
- 7 To try running this example using different data, close the figure window and change the data in cells B4:D9. Then reexecute all the Excel Link functions. The worksheet then shows the new frontier data, and MATLAB displays a new efficient frontier graph.

When you finish this example, close the figure window.

Bond Cash Flow and Time Mapping

This example illustrates the use of Financial Toolbox and Excel Link to compute a set of cash flow amounts and dates given a portfolio of five bonds whose maturity dates and coupon rates are known.

- 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 Excel Link function that transfers the column vector **Maturity** to **MATLAB**.

- 3 Make A19 the active cell to transfer the column vector Coupon Rate to MATLAB.
- 4 Make A20 the active cell to transfer the settlement date to MATLAB.
- 5 Execute the functions in cells A23 and A24 to enable Financial Toolbox 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 Excel.

The screenshot shows a Microsoft Excel spreadsheet titled "ExliSamp.xls" with the following content:

Cash Flow and Time Mapping for a Portfolio of Bonds

Settlement Date: 26-Jul-99

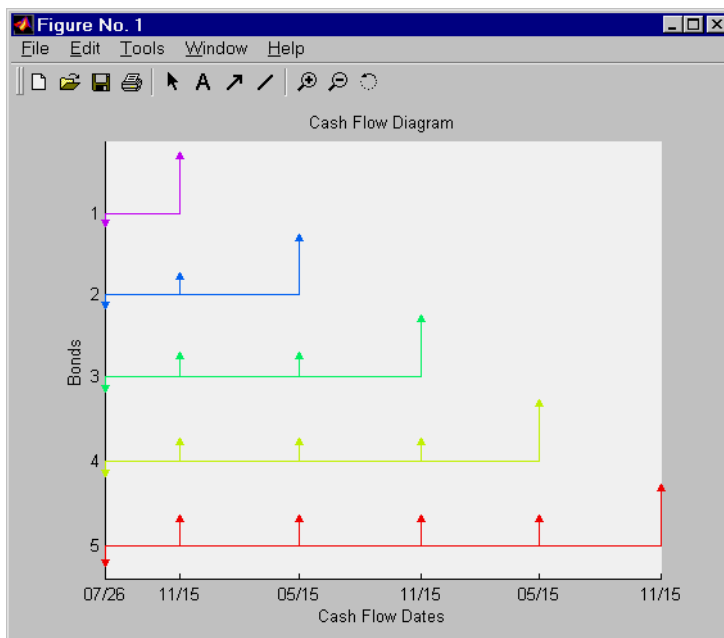
Bond Data		Cash Flow Dates							
	Maturity	Coupon Rate							
Bond1	15-Nov-99	0.05875	Bond1	07/26/99	11/15/99	N/A	N/A	N/A	N/A
Bond2	15-May-00	0.06375	Bond2	07/26/99	11/15/99	05/15/00	N/A	N/A	N/A
Bond3	15-Nov-00	0.08500	Bond3	07/26/99	11/15/99	05/15/00	11/15/00	N/A	N/A
Bond4	15-May-01	0.08000	Bond4	07/26/99	11/15/99	05/15/00	11/15/00	05/15/01	N/A
Bond5	15-Nov-01	0.15750	Bond5	07/26/99	11/15/99	05/15/00	11/15/00	05/15/01	11/15/01

Cash Flow Amounts						
	07/26/99	11/15/99	05/15/00	11/15/00	05/15/01	11/15/01
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

Excel Link Functions

- Transfer data to MATLAB.
 - A18: 0 <== MLPutMatrix("maturity", Maturity)
 - A19: 0 <== MLPutMatrix("cpnrate", CpnRate)
 - A20: 0 <== MLPutMatrix("sd", C3)
- Execute MATLAB Financial Toolbox Cash flow and Time mapping function.
 - A23: 0 <== MLEvalString("md = x2mdate(maturity,0); sdm = x2mdate(sd,0)")
 - A24: 0 <== MLEvalString("cfa, cfd = cfamounts(cpnrate, sdm, md, 2)")
- Transform date numbers to string cell array.
 - A27: 0 <== MLEvalString("i = find(isnan(cfd)); zcfd = cfd; zcfd(i) = 0; scfd = datestr(zcfd,2);")
 - A28: 0 <== MLEvalString("ccfd = num2cell(scfd,2); ccfdf(i) = {N/A}; ccfd = reshape(ccfd, size(cfd));")
 - A29: 0 <== MLEvalString("ccfa = cfa; ccfa(i) = 0; alldates = ccfdf(end, :);")
- Transfer output data to Excel.
 - A32: 0 <== MLGetMatrix("ccfd", "I3")
 - A33: 0 <== MLGetMatrix("alldates", "I13")
 - A34: 0 <== MLGetMatrix("ccfa", "I14")
- Plot the cash flow diagram.
 - A37: 0 <== MLEvalString("cfplot(cfd, cfa); dtaxis('x',6,50); title('Cash Flow Diagram'); xlabel('Cash Flow Dates'); ylabel('Bonds');")

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



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

Functions — By Category

- Link Management Functions (p. 3-2) Working with link management functions
- Data Management Functions (p. 3-3) Working with data management functions

Link Management Functions

<code>matlabinit</code>	Initialize Excel Link and start MATLAB process
<code>MLAutoStart</code>	Automatically start MATLAB process
<code>MLClose</code>	Terminate MATLAB process
<code>MLOpen</code>	Start MATLAB process
<code>MLUseCellArray</code>	Toggle <code>MLPutMatrix</code> to use cell arrays in MATLAB

Data Management Functions

<code>matlabfcn</code>	Evaluate MATLAB command given Excel data
<code>matlabsub</code>	Evaluate MATLAB command given Excel data and designate output location
<code>MLAppendMatrix</code>	Create or append MATLAB matrix with data from Excel worksheet
<code>MLDeleteMatrix</code>	Delete MATLAB matrix
<code>MLEvalString</code>	Evaluate command in MATLAB
<code>MLGetFigure</code>	Import current MATLAB figure into Excel spreadsheet
<code>MLGetMatrix</code>	Write contents of MATLAB matrix in Excel worksheet
<code>MLGetVar</code>	Write contents of MATLAB matrix in Excel VBA variable
<code>MLMissingDataAsNaN</code>	Set empty cells to NaN or zero
<code>MLPutMatrix</code>	Create or overwrite MATLAB matrix with data from Excel worksheet
<code>MLPutVar</code>	Create or overwrite MATLAB matrix with data from Excel VBA variable
<code>MLShowMatlabErrors</code>	Return standard Excel Link errors or full MATLAB errors using <code>MLEvalString</code>
<code>MLStartDir</code>	Specify MATLAB current working directory after startup
<code>MLUseFullDesktop</code>	Specify whether to use full MATLAB desktop or MATLAB Command window

Functions — Alphabetical List

matlabfcn

Purpose Evaluate MATLAB command given Excel data

Syntax

Worksheet: matlabfcn(command, inputs)

command MATLAB command to evaluate. The MATLAB command must be written as "command" (in double quotation marks).

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

sums the data in the spreadsheet cells B1 through B10 returning the output to the active worksheet cell or Excel Visual Basic for Applications (VBA) output variable.

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

plots the data in worksheet cells B1 through B10 using x as the marker type.

See Also `matlabsub`

Purpose Initialize Excel Link and start MATLAB process

Syntax matlabinit

Note To run matlabinit, pull down the Excel **Tools** menu and click **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 Excel Link and starts MATLAB process. If Excel Link has already been initialized and MATLAB is running, subsequent invocations do nothing. Use matlabinit to start Excel Link and MATLAB manually when you have set MIAutoStart to no. If MIAutoStart is set to yes, matlabinit executes automatically.

See Also MIAutoStart, MIOpen

matlabsub

Purpose Evaluate MATLAB command given Excel data and designate output location

Syntax

Worksheet:	matlabsub(command, edat, inputs)
command	MATLAB command to evaluate. The MATLAB command must be written as "command" (in double quotation marks).
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.
inputs	Variable length input argument list passed to MATLAB command. Argument list may contain a range of worksheet cells that contain input data.

Description Passes the specified command to MATLAB 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 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

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

sums the data in worksheet cells B1 through B10, returning the output to cell A1.

See Also

matlabfcn

MLAppendMatrix

Purpose Create or append MATLAB matrix with data from 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, Excel proceeds by performing the necessary type coercion. However, if you call MLAppendMatrix from within a VBA macro, and mdat is not an Excel Range data type, the call fails. Excel generates the error message ByRef Argument Type Mismatch.

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. The function returns an error if the dimensions do not match. 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

B is a 2-by-2 MATLAB matrix.

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

appends the data in cell range A1:A2 to the MATLAB matrix B. B is now a 2-by-3 matrix with the data from A1:A2 in the third column.

		A1
		A2

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.

```
MAppendMatrix(C1, new_data)
```

appends the data in cell range A1:B2 to B. 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 Excel Link and MATLAB every time Excel starts (default).

"no" Cancel automatic startup of Excel Link and MATLAB. If Excel Link and MATLAB are running, it does not stop them.

Description

Sets automatic startup of Excel Link and MATLAB. When Excel Link is installed, the default is yes. A change of state takes effect the next time Excel is started.

Examples

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

Cancels automatic startup of Excel Link and MATLAB. The next time Excel starts, Excel Link and MATLAB will not start.

See Also

matlabinit, MLClose, MLOpen

Purpose Terminate MATLAB process

Syntax

Worksheet:	MLClose()
Macro:	MLClose

Description Terminates the MATLAB process, deletes all variables from the MATLAB workspace, and tells Excel that MATLAB 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

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

deletes matrix A from the MATLAB workspace.

Purpose Evaluate command in MATLAB

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 MATLAB for evaluation. The specified action alters only the MATLAB workspace. Nothing is done in the Excel workspace.

Examples

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

divides the MATLAB variable b by 2 and plots it. Only the MATLAB variable b is modified. To update data in the Excel worksheet, use MLGetMatrix.

See Also

MLGetMatrix

MLGetFigure

Purpose Import current MATLAB figure into 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 Excel.
height	Specify the height in normalized units of the MATLAB figure when imported into Excel.

Description Import the current MATLAB figure into Excel where the left-top 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 Excel Link 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 MLGetFigure(.50,.25)

imports the current MATLAB figure into Excel. The width of the figure is half that of the original MATLAB figure and the height is quarter of the original figure.

See Also MLGetMatrix, MLGetVar

Purpose Write contents of MATLAB matrix in 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 already exists in the specified worksheet cells, it is overwritten. If the dimensions of the MATLAB matrix are larger than those of the specified cells, the data will overflow 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.

If `edat` is an explicit cell address and you later insert or delete rows or columns, or move or copy the function to another cell, edit `edat` to correct the address. Excel Link does not automatically adjust cell addresses in `MLGetMatrix`.

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, 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 Excel Link 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

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

writes the contents of the MATLAB matrix `bonds` starting in cell C10 of Sheet2. If `bonds` is a 4-by-3 matrix, data fills cells C10..E13.

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

accesses the MATLAB matrix named as a string in worksheet cell B12 and writes the contents of the matrix in the worksheet starting at the location named as a string in worksheet cell B13.

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

writes the contents of MATLAB matrix `A` in the worksheet starting at the cell named `RangeA`.

In addition, when using the `MLGetMatrix` function in an Excel macro, you can use a range object returned by the VBA `Cells` function to

specify where to place the data. To do this using the range object's Address property:

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

See Also

MLAppendMatrix, MLPutMatrix

MLGetVar

Purpose Write contents of MATLAB matrix in 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

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

writes the contents of MATLAB matrix J in the VBA variable named DataJ.

See Also `MLPutVar`

Purpose Set empty cells to NaN or zero

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 zero. When Excel Link is installed, the default is "no" which means that empty cells are handled as 0s. If the value of MLUseCellArray is changed to "yes", the change remains in effect the next time Excel is started.

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

Examples

To cancel the use of NaNs for empty cells, enter

```
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 been started, subsequent calls to MLOpen do nothing. Use MLOpen to restart MATLAB after you have stopped it with MLClose in a given Excel session.

Note We recommend using matlabinit rather than MLOpen, since matlabinit starts MATLAB and initializes Excel Link.

Examples

MLOpen()

starts the MATLAB process.

See Also matlabinit, MLClose

Purpose Create or overwrite MATLAB matrix with data from 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 already 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.

To use MLPutMatrix in a subroutine, you must indicate the source of the worksheet data using the 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 use the name instead of actually specifying the range. For example:

MLPutMatrix

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

where temp is a named range in your worksheet.

Examples

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

Creates or overwrites matrix A in the MATLAB workspace with the data in the worksheet range A1:C3.

See Also

MLAppendMatrix, MLGetMatrix

Purpose Create or overwrite MATLAB matrix with data from 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 already 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

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

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

MLPutVar

See Also

[MLGetVar](#)

Purpose	Return standard Excel Link errors or full MATLAB errors using MLEvalString												
Syntax	<table><tr><td>Worksheet:</td><td>MLShowMatlabErrors("yes")</td></tr><tr><td></td><td>MLShowMatlabErrors("no") (Default)</td></tr><tr><td>Macro:</td><td>MLShowMatlabErrors "yes"</td></tr><tr><td></td><td>MLShowMatlabErrors "no" (Default)</td></tr><tr><td>"yes"</td><td>Displays the full MATLAB error string in Excel upon MLEvalString failure.</td></tr><tr><td>"no"</td><td>Displays the standard Excel Link errors in Excel upon MLEvalString failure.</td></tr></table>	Worksheet:	MLShowMatlabErrors("yes")		MLShowMatlabErrors("no") (Default)	Macro:	MLShowMatlabErrors "yes"		MLShowMatlabErrors "no" (Default)	"yes"	Displays the full MATLAB error string in Excel upon MLEvalString failure.	"no"	Displays the standard Excel Link errors in Excel upon MLEvalString failure.
Worksheet:	MLShowMatlabErrors("yes")												
	MLShowMatlabErrors("no") (Default)												
Macro:	MLShowMatlabErrors "yes"												
	MLShowMatlabErrors "no" (Default)												
"yes"	Displays the full MATLAB error string in Excel upon MLEvalString failure.												
"no"	Displays the standard Excel Link errors in Excel upon MLEvalString failure.												
Description	Sets the error display mode of Excel Link when executing MATLAB commands using MLEvalString.												
Examples	<pre>MLShowMatlabErrors("no")</pre> <p>will cause MLEvalString failures to show standard Excel Link errors such as #COMMAND!.</p> <pre>MLShowMatlabErrors("yes")</pre> <p>Will cause MLEvalString failures to show MATLAB error strings, for example:</p> <pre>??? Undefined function or variable 'foo'</pre>												
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 working directory for MATLAB after startup. Note this function does not work like the standard Windows **Start In** setting in that it will not automatically run any startup.m or matlabrc.m found in the directory specified.

Examples To set the MATLAB working directory to d:\work after startup, run:

```
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:

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

See Also MLAutoStart

Purpose Toggle MLPutMatrix to use cell arrays in MATLAB

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 Excel Link is installed, the default is "no". If the value of MLUseCellArray is changed to "yes", the change remains in effect the next time Excel is started.

Examples

To cancel automatic use of cell arrays for easy transfer of data, enter

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

See Also

MLPutMatrix

MLUseFullDesktop

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 MATLAB with the full desktop.
"no"	Start MATLAB with the Command window only.

Description Sets MATLAB to start with the full desktop or Command window only. When Excel Link is installed, the default is "yes".

Examples

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

will cause MATLAB to start with the command window only.

See Also `matlabinit`, `MLClose`, `MLOpen`

Error Messages and Troubleshooting

This appendix covers the following topics:

Excel Cell Error Messages (p. A-2)	Error messages displayed in a worksheet cell
Error Messages (p. A-6)	Excel error messages and their meanings
Audible Error Signals (p. A-10)	Audible error signals while passing data to MATLAB
Data Errors (p. A-11)	Undesirable data characteristics

Excel Cell Error Messages

Excel may display one of these error messages in a worksheet cell.

In the following table of Excel cell error messages, the first column contains the message provided by Excel. The error messages all begin with the number sign #. Most end with an exclamation point ! or with a question mark ?.

Excel Cell Error Messages

Excel Cell Error Message	Meaning	Solution
#COLS>#MAXCOLS!	Your MATLAB variable exceeds the Excel limit of #MAXCOLS! columns.	This is a limitation in Excel. Try the computation with a variable containing fewer columns.
#COMMAND!	MATLAB 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. See MLAppendMatrix in Chapter 4, “Functions — Alphabetical List”.
#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 “Data Types” in the MATLAB Programming documentation.
#MATLAB?	You used an Excel Link function and MATLAB is not running.	Start Excel Link and MATLAB. See “Starting and Stopping Excel Link” on page 1-8.

Excel Cell Error Messages (Continued)

Excel Cell Error Message	Meaning	Solution
#NAME?	Excel doesn't recognize the function name. 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 Excel Link" on page 1-5. 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 Excel. Try the computation with a variable containing fewer rows.
#SYNTAX?	You entered an Excel Link function with incorrect syntax; for example, the double quotation marks (") may be missing, or you used single quotation marks (') instead of double quotation marks.	Verify and correct the function syntax. See Chapter 4, "Functions — Alphabetical List" for function syntax.
#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, you may ignore the message. Or, in this special case, remove <code>MatlabRequest</code> from the subroutine.

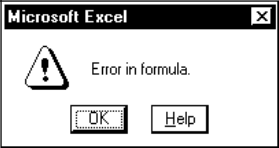

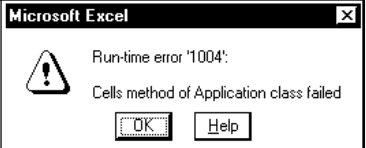
Note When you open an Excel worksheet that contains Excel Link functions, Excel tries to execute the functions from the bottom up and right to left, thus possibly generating cell error messages (#COMMAND!, #NONEXIST!, etc.). Such behavior is usual for Excel. Simply ignore the messages, close any MATLAB figure windows, and reexecute the cell functions one at a time in the correct order by pressing **F2**, then **Enter**.

Error Messages

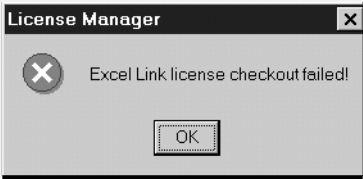
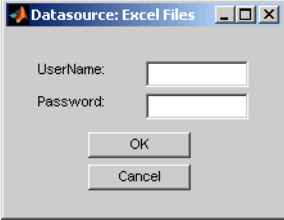
Excel may display one of these error message boxes.

- The first column of this table shows the error messages. The first three are from Excel, and the last one is from the license manager.
- The second column 2 indicates the type of error that caused the message box to appear.
- The third column proposes a solution for the error.

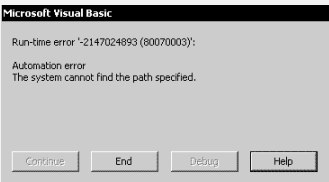
Excel Error Message Boxes

Excel Error Message Box	Meaning	Solution
 <p>The screenshot shows a dialog box titled "Microsoft Excel" with a warning icon and the text "Error in formula." Below the text are "OK" and "Help" buttons.</p>	<p>You entered a formula incorrectly. Common errors include a space between the function name and the left parenthesis; or missing, extra, or mismatched parentheses.</p>	<p>Check entry and correct typing errors.</p>
 <p>The screenshot shows a dialog box titled "Microsoft Excel" with a warning icon and the text "Can't find project or library." Below the text are "OK" and "Help" buttons.</p>	<p>You tried to execute a macro and the location of <code>excllink.xla</code> is incorrect.</p>	<p>Click OK. The References window opens. Remove the check from MISSING: <code>excllink.xla</code>. Find <code>excllink.xla</code> in its correct location, select its check box in the References window, and click OK.</p>
 <p>The screenshot shows a dialog box titled "Microsoft Excel" with a warning icon and the text "Run-time error '1004': Cells method of Application class failed." Below the text are "OK" and "Help" buttons.</p>	<p>You used <code>MLGetMatrix</code> and the matrix is larger than the space available in the worksheet. This error destabilizes Excel Link and changes worksheet calculation mode to manual.</p>	<p>Click OK. Reset worksheet calculation mode to automatic and save your worksheet (if desired). Close Excel and MATLAB. Restart Excel, Excel Link, and MATLAB.</p>

Excel Error Message Boxes (Continued)

Excel Error Message Box	Meaning	Solution
	<p>The license passcode that you entered was invalid.</p>	<p>Check that you entered the license passcode properly. If you used a proper passcode and you are still unable to start Excel Link, contact your MathWorks representative.</p>
	<p>This message appears when an attempt to connect to Excel from Database Toolbox fails.</p>	<p>Make sure that the Excel spreadsheet referenced by the data source exists, then retry the connection.</p>

Excel Error Message Boxes

Excel Error Message Box	Meaning	Solution
	<p>If more than one version of MATLAB is installed on your desktop, when you attempt to start the MATLAB automation server from Microsoft Excel, you receive this error.</p>	<p>To correct this error, perform the following:</p> <ol style="list-style-type: none"> 1 Shut down all instances of MATLAB and Microsoft Excel. 2 Open a Command Prompt window, and using <code>cd</code>, change to the <code>bin\win32</code> subdirectory of your MATLAB 7.0 installation directory. 3 Type the command <pre style="margin-left: 40px;">.\matlab /regserver</pre> 4 When MATLAB starts, close it. Using <code>/regserver</code> fixes the registry entries. 5 Start Microsoft Excel. Excel Link should now load properly. 6 Verify that Excel Link is working by entering the following command from the MATLAB Command Window: <pre style="margin-left: 40px;">a = 3.14159</pre> 7 In the open instance of Microsoft Excel, enter the following formula in cell A1: <pre style="margin-left: 40px;">=mlgetmatrix("a", "a1")</pre> 8 The value 3.14159 appears in cell A1.

Audible Error Signals

Audible error signals while passing data to MATLAB with `MLPutMatrix` or `MLAppendMatrix` usually mean 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.

Data Errors

In this section...

“Matrix Data Errors” on page A-11

“Errors When Opening Saved Worksheets” on page A-11

Matrix Data Errors

Data in the MATLAB or 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.

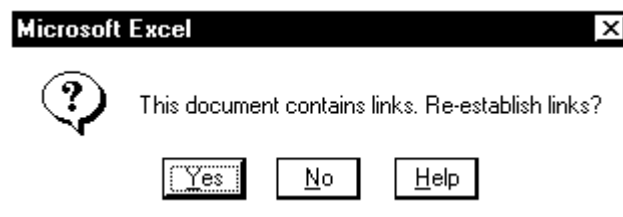
Errors When Opening Saved Worksheets

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

- When you open an Excel worksheet that contains Excel Link functions, Excel tries to execute the functions from the bottom up and right to left, thus possibly generating cell error messages (`#COMMAND!`, `#NONEXIST!`, etc.).

Such behavior is usual for Excel. Simply ignore the messages, close any MATLAB figure windows, and 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 Excel Link functions and later open it under a different computer environment where the `excllink.xla` add-in is in a different location, Excel may display a message box.



To address this issue, do the following:

- a** Click **No**.
- b** Select **Edit > Links**.
- c** In the **Links** dialog box, click **Change Source**.
- d** In the **Change Links** dialog box, find and select `excllink.xla` under `matlabroot/toolbox/exlink` and click **OK**.

Excel 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.

- e** Back in the **Links** dialog box, click **OK**. The worksheet now correctly connects to the Excel Link 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`.

Examples

Use this list to find examples in the documentation.

Macro Examples

“Example: Using MLGetMatrix in a Macro Subroutine” on page 1-18

“Example: Running Excel Link Functions from the Visual Basic Editor”
on page 1-18

Financial Examples

“Data Regression and Curve Fitting” on page 2-3

“Data Interpolation” on page 2-9

“Stock Option Pricing Using the Binomial Model” on page 2-13

“Calculating and Plotting the Efficient Frontier of Financial Portfolios”
on page 2-16

“Bond Cash Flow and Time Mapping” on page 2-20

A

- add-in, Excel Link 1-5 A-3
- audible error signals A-10
- /automation option 1-8

B

- beeps A-10
- binomial tree 2-13

C

- calculation mode A-6
- cash flow example 2-20
- cell error messages A-2
- COLS error A-2
- COMMAND error A-2
- computer memory errors A-10
- curve fitting example 2-3

D

- data errors A-11
- data interpolation example 2-9
- data types 1-2
- data-location argument A-10 to A-11
- date numbers 1-20
- date system 1-20
- dates 1-20
- DIMENSION error A-2
- double quotation marks A-3

E

- efficient frontier example 2-16
- empty matrix A-11
- error messages A-2
- examples
 - cash flow 2-20
 - efficient frontier 2-16
 - interpolating data 2-9

- regression and curve fitting 2-3
- stock option 2-13

- Excel error message boxes A-6

- Excel Link

- configuring 1-5
- installing 1-3
- starting 1-8
- stopping 1-3 1-9
- using 2-1

- Excel Link functions

- about 1-10
- excllink.xla 1-3
- excllink.xla add-in A-6
- exlink.ini file 1-3
- ExliSamp.xls file
 - location 1-3
 - purpose 2-1

F

- file initialization 1-3

- Function Wizard for Excel Link 1-13

- functions

- about 1-10
- arguments
 - working with 1-13

- Excel Link

- types of 1-10

- Excel Link versus Microsoft Excel 1-10

- MATLAB Function Wizard for Excel Link 1-13

- running from within Excel Visual Basic Editor 1-18

I

- initialization file 1-3

- international users

- information for 1-22

- interpolating data 2-9

INVALIDNAME error A-2
INVALIDIDTYPE error A-2

K

Kernel32.dll 1-3

L

license passcode A-7
localization 1-22

M

macros
 creating 1-17
MATLAB error A-2
MATLAB Function Wizard for Excel Link 1-13
matlabfcn 4-2
matlabinit 4-3
matlabsub 4-4
matrix dimensions A-2
MLAppendMatrix 4-6
MLAutoStart 4-8
MLClose 4-9
MLDeleteMatrix 4-10
MLEvalString 4-11
MLFullDesktop 4-26
MLGetFigure 4-12
MLGetMatrix 4-13
MLGetVar 4-16
MLMissingDataAsNaN 4-17
MLOpen 4-18
MLPutMatrix 4-19
MLPutVar 4-21
MLShowMatlabErrors 4-23
MLStartDir 4-24
MLUseCellArray 4-25

N

NAME error A-3
NONEXIST error A-3
nonexistent variable A-11

P

passcode
 license A-7
Preferences
 setting 1-6

R

regression and curve fitting 2-3
requirements 1-3
ROWS error A-3

S

signals error A-10
single quotation marks A-3
spreadsheet formulas 1-11
spreadsheets 1-11
 using 1-11
stock option pricing example 2-13
SYNTAX error A-3
system
 date 1-20
system path
 files on 1-3

T

troubleshooting error messages A-2

V

VALUE error A-3

W

worksheet formulas 1-11
worksheets 1-11
 saved A-11
 using 1-11

Z

zero matrix A-11
zero matrix cells A-11