Excel Link

For Use with MATLAB®

- Computation
- Visualization
- Programming

User's Guide

Version 2

How to Contact The MathWorks:

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup

`support@mathworks.com`	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. Mail
3 Apple Hill Drive
Natick, MA 01760-2098

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

Excel Link User's Guide

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 forVersion 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)

Contents

1 Getting Started

2 Using Excel Link

3 Function Reference

A Error Messages and Troubleshooting

B Installed Files

Index

1

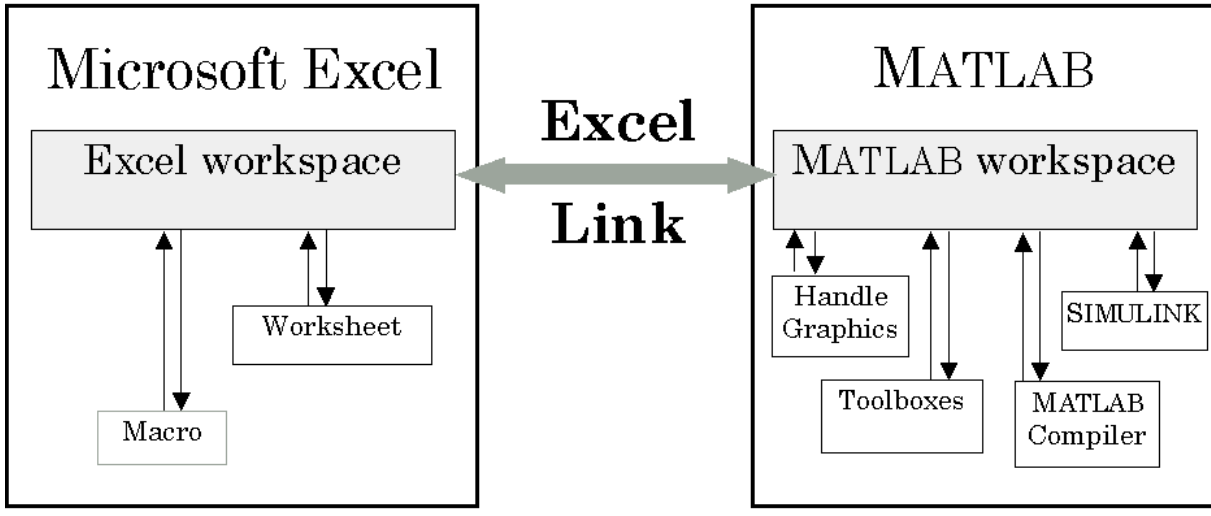
Getting Started

What Is Excel Link?

Excel Link is a software add-in that integrates Microsoft Excel and MATLAB in a Microsoft Windows-based computing environment. By connecting Excel and MATLAB, you can access the numerical, computational, and graphical power of MATLAB from Excel worksheet and macro programming tools. Excel Link lets you exchange and synchronize data between the two environments.

Understanding the Environment

Excel Link communicates between the Excel workspace and the MATLAB workspace. It positions Excel as a front end to MATLAB. You use Excel Link functions from an Excel worksheet or macro, and you never have to leave the Excel environment. With a small number of functions to manage the link and manipulate data, Excel Link is powerful in its simplicity.



Installing and Operating Excel Link

Follow these instructions to install Excel Link and then configure Excel.

System Requirements

Excel Link requires approximately 202 kilobytes of disk space. Operating system requirements are

- Microsoft Windows XP
- Microsoft Windows 2000

Excel Link also requires one of the following versions of Excel:

- Microsoft Excel 98
- Excel 2000
- Excel 2002
- Excel 2003

and MATLAB for Windows version 5.1 or later.

For best results with MATLAB figures and graphics, set the color palette of your display to a value greater than 256 colors. Click **Start**, then **Settings** and **Control Panel**. Open **Display**, and on 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. Click in the box for Excel Link when you select MATLAB components to install.

Configuring Excel to Work with Excel Link

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

1. Start Microsoft Excel.

2. Pull down the **Tools** menu, select **Add-Ins** and click **Browse**.

3 Find and select the Excel Link add-in `excllink.xla` under `<matlab>/toolbox/exlink`. Click **OK**.

Note Throughout this document the notation `<matlab>` represents the MATLAB root directory, the directory where MATLAB is installed on your system.

---

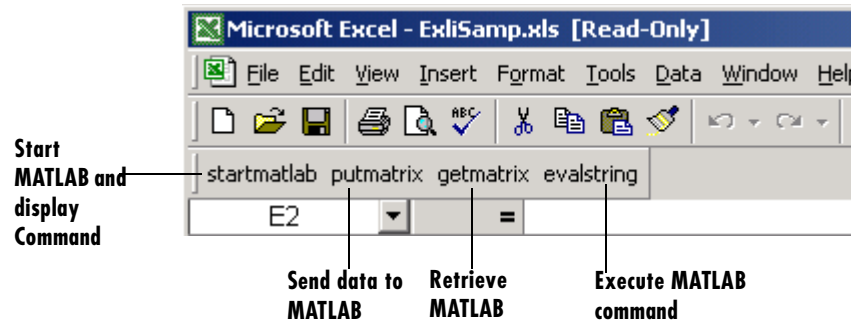
4 Back in the **Add-Ins** window, make sure there is a check in the box for Excel Link for use with MATLAB and click **OK**. The Excel Link add-in loads now and with each subsequent invocation of Excel.

5 Watch for the appearance of the **MATLAB Command Window** button on the Windows taskbar.

Note The MATLAB desktop does not start automatically at this time. If you want to run the desktop, enter the `desktop` command in the Command Window.

---

6 Watch for the appearance of the Excel Link toolbar on your Excel worksheet.



Excel Link is now ready for your use.

Starting Excel Link

Automatic Start

When installed and configured according to the preceding instructions, Excel Link and MATLAB automatically start when you start Excel.

If you do not want Excel Link and MATLAB to start automatically when you start Excel, enter `=MLAutoStart("no")` in a worksheet cell. This function changes the initialization file so that Excel Link and MATLAB no longer start automatically when you start Excel. See `MLAutoStart` in Chapter 3, "Function Reference."

Manual Start

To start Excel Link and MATLAB manually from Excel, pull down the **Tools** menu and select **Macro**. In the **Macro Name/Reference** box enter `matlabinit` and click **Run**. Watch for the **MATLAB Command Window** button to appear on the taskbar. See `matlabinit` in Chapter 3, "Function Reference."

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 Command Window is minimized, and the MATLAB desktop is not running.

To add the `/automation` option to the command line,

1. Right-click on your shortcut to MATLAB.
2. Select **Properties.**
3. Click on the **Shortcut** tab.
4. Add the string `/automation` in the **Target** field. Remember to leave a space between `matlab.exe` and `/automation`.

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()` in an Excel worksheet cell. You can restart Excel Link and MATLAB manually with `MLOpen` or `matlabinit`.

If you stop MATLAB directly in the MATLAB Command Window and leave Excel running, enter `=MLClose()` in an Excel worksheet cell. (`MLClose` tells Excel that MATLAB is no longer running.) You can restart Excel Link and MATLAB manually with `MLOpen` or `matlabinit`.

What the Functions Do

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 functions to manage the link and to manipulate data. You never have to leave the Excel environment. You can invoke functions as worksheet cell formulas or in macros.

See Chapter 3, "Function Reference" for details on each function.

Link Management Functions

Excel Link provides four link management functions to initialize, start, and stop Excel Link and MATLAB.

Function	Purpose
`matlabinit`	Initialize Excel Link and start MATLAB process.
`MLAutoStart`	Automatically start MATLAB process.
`MLClose`	Terminate MATLAB process.
`MLOpen`	Start MATLAB process.

You can invoke any link management function except `matlabinit` as a worksheet cell formula or in a macro. You invoke `matlabinit` from the Excel **Tools Macro** menu or in a macro subroutine.

Use `MLAutoStart` to toggle automatic startup. If you install and configure Excel Link according to the default instructions, Excel Link and MATLAB automatically start every time you start Excel. If you choose manual startup, use `matlabinit` to initialize Excel Link and start MATLAB.

Use `MLClose` to stop MATLAB without stopping Excel, and use `MLOpen` or `matlabinit` to restart MATLAB in the same Excel session.

Data Management Functions

Excel Link provides nine data management functions to copy data between Excel and MATLAB and to execute MATLAB commands from Excel.

Function	Purpose
`matlabfcn`	Evaluate MATLAB command given Excel data.
`matlabsub`	Evaluate MATLAB command given Excel data and designate output location.
`MLAppendMatrix`	Create or append MATLAB matrix with data from Excel worksheet.
`MLDeleteMatrix`	Delete MATLAB matrix.
`MLEvalString`	Evaluate command in MATLAB.
`MLGetMatrix`	Write contents of MATLAB matrix in Excel worksheet.
`MLGetVar`	Write contents of MATLAB matrix in Excel VBA variable.
`MLPutMatrix`	Create or overwrite MATLAB matrix with data from Excel worksheet.
`MLPutVar`	Create or overwrite MATLAB matrix with data from Excel VBA variable.

You can invoke any data management function except `MLGetVar` and `MLPutVar` as a worksheet cell formula or in a macro. You can invoke `MLGetVar` and `MLPutVar` only in a macro.

Use `MLAppendMatrix`, `MLPutMatrix`, and `MLPutVar` to copy data from Excel to MATLAB.

Use `MLEvalString` to execute MATLAB commands from Excel.

Use `MLDeleteMatrix` to delete a MATLAB variable.

Use `matlabfcn`, `matlabsub`, `MLGetMatrix` and `MLGetVar` to copy data from MATLAB to Excel.

Tips and Reminders

These tips and reminders help you use Excel Link efficiently.

Excel Link functions *perform an action*, while Microsoft Excel functions *return a value*. Keep this distinction in mind as you use Excel Link. Excel operations and function keys may behave differently with Excel Link functions.

Syntax

Function Names

- *Excel Link 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 lower case; for example, `plot(f)`.

Worksheet Formulas

- Begin worksheet formulas with + or =. For example,

  ```
  =mlputmatrix("a", C10)
  ```

- In worksheet formulas, enclose function arguments in parentheses. In macros, leave a space between the function name and the first argument; do not use parentheses.

Variable Names

- 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 quotes; for example, `MLDeleteMatrix("Bonds")`.
  - A variable-name argument without quotes is an *indirect* reference. The function evaluates the contents of the argument to get the variable name. The argument must be a worksheet cell address or range name.
- A data-location argument must be a worksheet cell address or range name. Do not enclose a data-location argument in quotes (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 Excel Link does not accept spaces or square brackets (`[]`) in sheet names or range designations.

Worksheets

- 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 entered formula.
- We suggest selecting **Move Selection after Enter** on the **Excel Tools Options -> Edit** tab. The active cell changes when an operation is complete, providing a useful confirmation for lengthy operations.
- We recommend using Excel Link functions in automatic calculation mode. If you use `MLGetMatrix` in manual calculation mode, enter the 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.
- To recalculate Excel Link functions in a worksheet, reexecute each function by pressing **F2**, then **Enter**.
- Pressing **F9** to recalculate a worksheet affects only Excel functions (which return a value). **F9** does not operate on Excel Link functions, which perform an action.
- To "automate" the recalculation of an Excel Link function, add to it some cell whose value changes. For example:

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

  When the value in cell `C1` changes, Excel re-executes the `MLPutMatrix` function. Be careful, however, not to create endless recalculation loops.
- Excel Link functions expect A1-style worksheet cell references. Select `A1` cell **Reference Style** on the **Excel Tools Options -> General** tab.
- If you use explicit cell addresses in `MLGetMatrix` and later insert or delete rows or columns, or move or copy the function to another cell, edit the

argument to correct the addresses. Excel Link does not automatically adjust cell addresses in `MLGetMatrix`.

- Enter (type) Excel Link functions directly in worksheet cells. Do not use the Excel Function Wizard; it generates unpredictable results.

Macros

- To create macros that use Excel Link functions, you must first configure Excel to reference the functions from the Excel Link add-in. From the Visual Basic environment pull down the **Insert** menu and select **Module**. When the **Module** page opens, pull down the **Tools** menu and select **References**. In the **References** window, select the box for `excllink.xla` and click **OK**. You may have to use **Browse** to find the `excllink.xla` file.
- If you 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
```

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

Data Types

- Excel Link handles only 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.

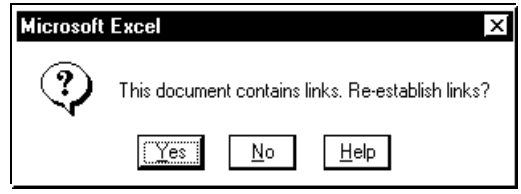
Dates

- Default Excel date numbers start from January 1, 1900, while MATLAB date numbers start from January 1, 0000. Thus May 15, 1996 is 35200 in Excel and 729160 in MATLAB, a difference of 693960. If you use date numbers in MATLAB calculations, apply the 693960 constant: add it to Excel date numbers going into MATLAB, or subtract it from MATLAB date numbers

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

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.



Click **No**. Then pull down the **Edit** menu and select **Links**. In the **Links** window, click **Change Source**. In the **Change Links** window, find and select `excllink.xla` under `<matlab>/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. Back in the **Links** window, click **OK**. The worksheet now correctly connects to the Excel Link add-in.

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

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 Windows documentation to determine which regional setting differences exist among various international versions.

2

Using Excel Link

Example 1: Regression and Curve Fitting (p. 2-3)	Data regression and curve fitting.
Example 2: Interpolating Data (p. 2-9)	Uses an Excel worksheet to organize and display the original data and the interpolated output data.
Example 3: Pricing a Stock Option with the Binomial Model (p. 2-13)	Uses the binomial model to price an option.
Example 4: Calculating and Plotting the Efficient Frontier of Financial Portfolios (p. 2-16)	Analyzes three portfolios, using rates of return for six time periods.
Example 5: Bond Cash Flow and Time Mapping (p. 2-20)	Computes a set of cash flow amounts and dates given a portfolio of five bonds.

This section shows how Microsoft Excel, Excel Link, and MATLAB work together to solve real-world problems.

These examples ship with Excel Link in the file `ExliSamp.xls`, which is installed in `<matlab>/toolbox/exlink/`. Start Excel, Excel Link, and MATLAB. Open and try executing the examples.

Note Examples 1 and 2 use only basic MATLAB functions. Examples 3, 4, and 5 use functions in the optional MATLAB Financial Toolbox. The Financial Toolbox in turn requires the Statistics and Optimization Toolboxes.

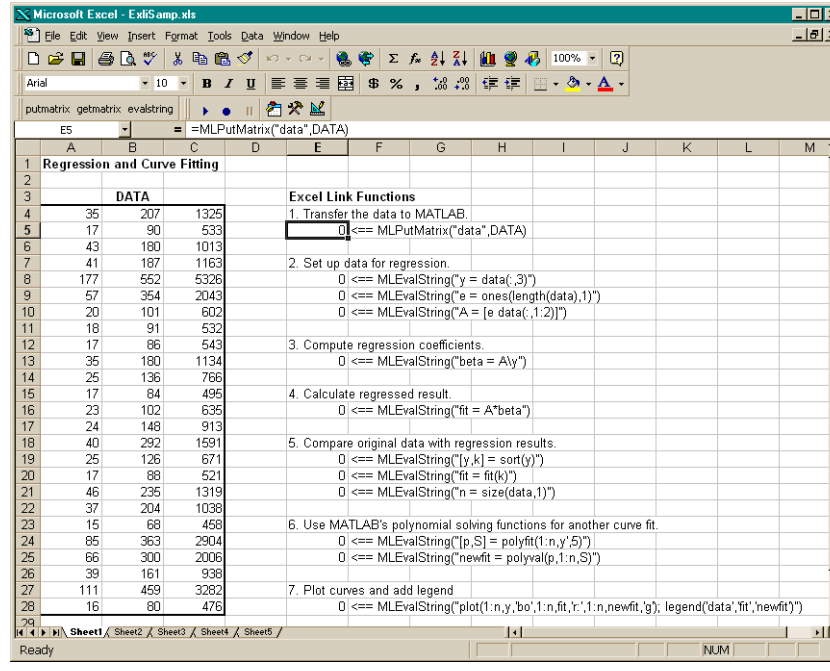
Example 1: Regression and Curve Fitting

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

This example does 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

To try the worksheet-only version of this example, click the `Sheet1` tab on `ExliSamp.xls`.



The worksheet contains one named range: `A4:C28` is named `DATA` and contains the sample data set:

1. Make `E5` the active cell. Press **F2**, then **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.

2. Move to cell `E8` and press **F2**, then **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.

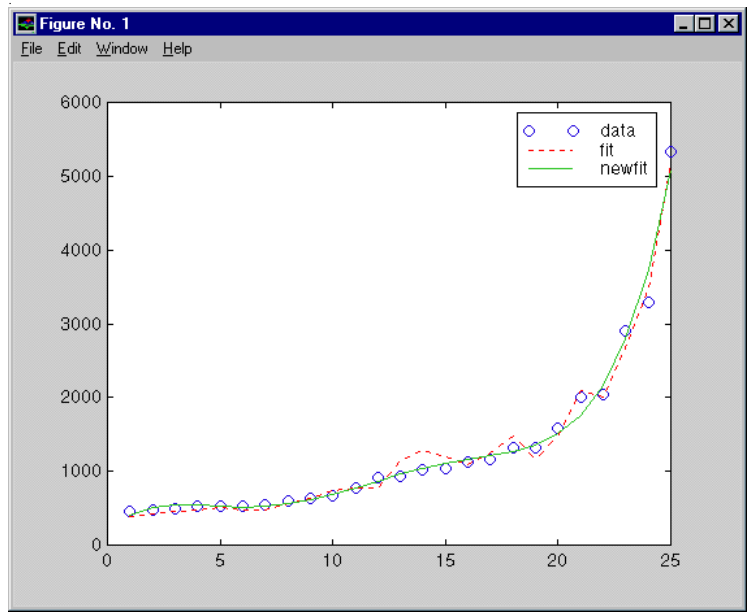
3. Execute the function in cell `E13`. This function computes the regression coefficients by using the MATLAB backslash operation to solve the (overdetermined) system of linear equations, `A*beta = y`.

4. Execute the function in cell `E16`. MATLAB matrix-vector multiplication produces the regressed result (`fit`).

5. 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.

6. 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`).

7. Finally, 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.

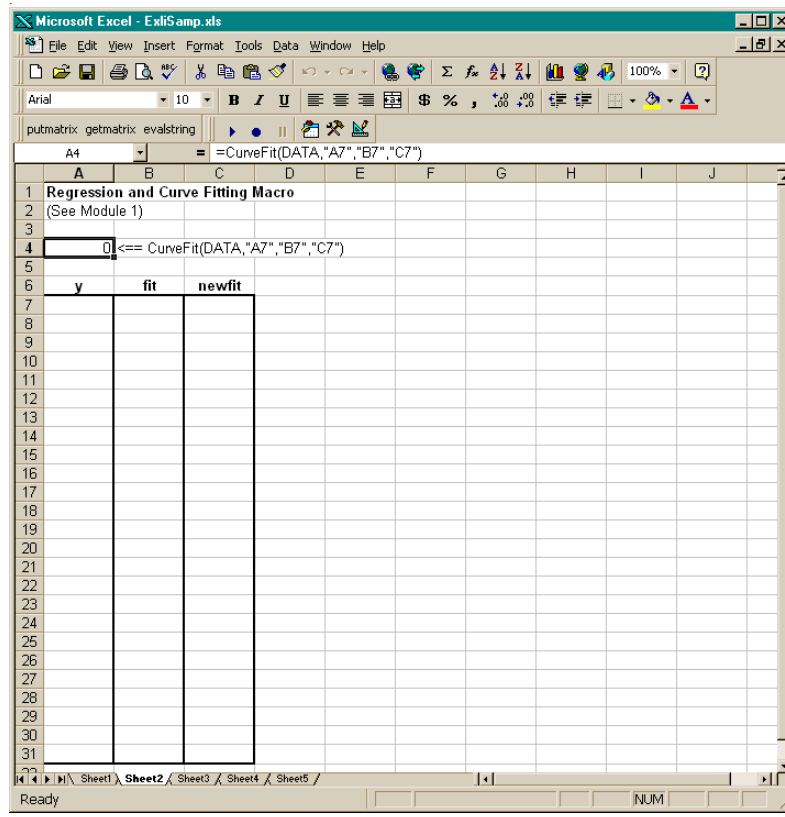


Since the data is closely correlated but not exactly linearly dependent, the `fit` curve (dashed line) shows a close, but not an exact, fit. The fifth-degree polynomial curve, `newfit`, represents a more accurate mathematical model for the data.

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

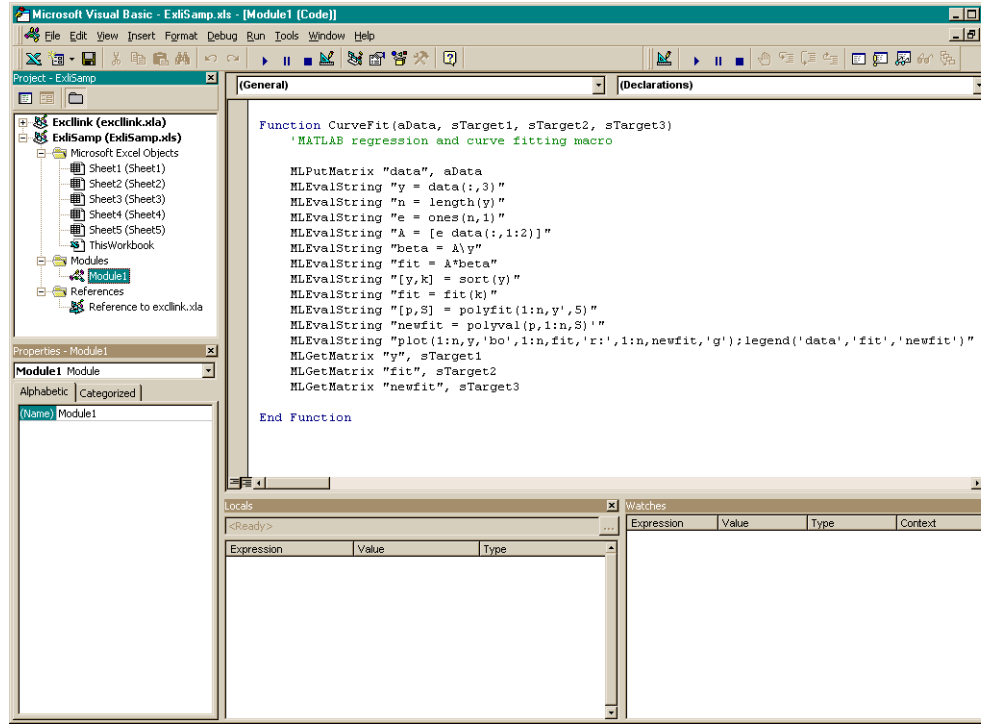
Macro Version

To try the macro-and-worksheet version of this example, click the `Sheet2` tab on `ExliSamp.xls`.



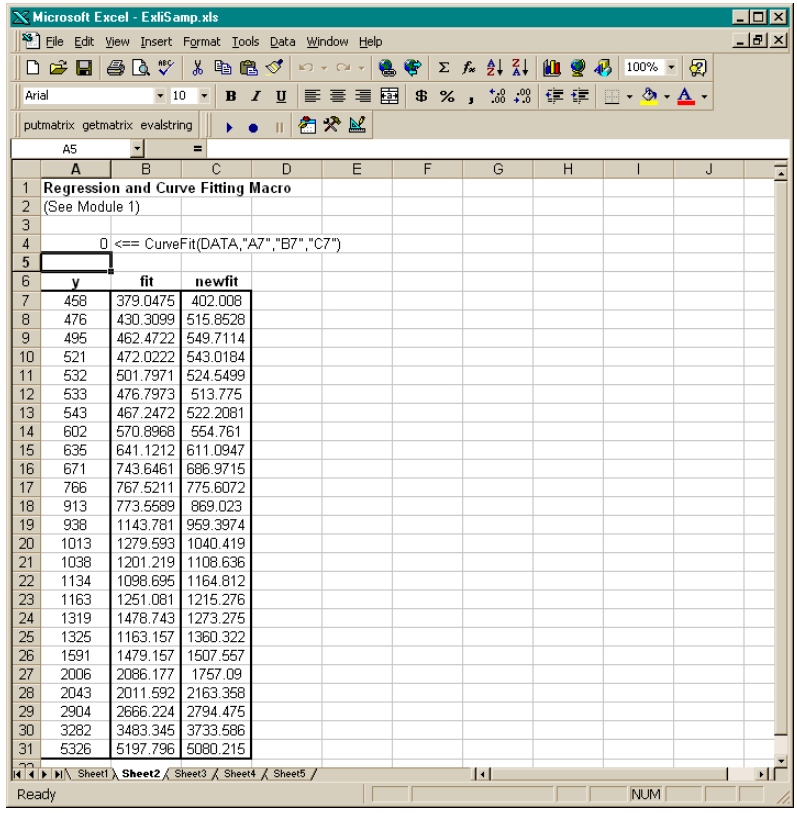
Make cell `A4` the active cell, but do not execute it yet.

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



While this module is open, pull down the **Tools** menu and select **References**. In the **References** window, make sure there is a check in the box for `excllink.xla`. If not, check the box and click **OK**. You may have to use **Browse** to find the `excllink.xla` file.

Back in cell `A4` of `Sheet2`, press **F2**, then **Enter** to execute the `CurveFit` macro. The macro executes the same functions as in Step 1 through Step 7 of the worksheet version (in a slightly different order), including plotting the graph. Plus, it copies the original data `y` (sorted), the corresponding regressed data `fit`, and the polynomial data `newfit`, to the worksheet. (The last three `MLGetMatrix` functions in the `CurveFit` macro copy data to the Excel worksheet.)



When you have finished the example, close the figure window.

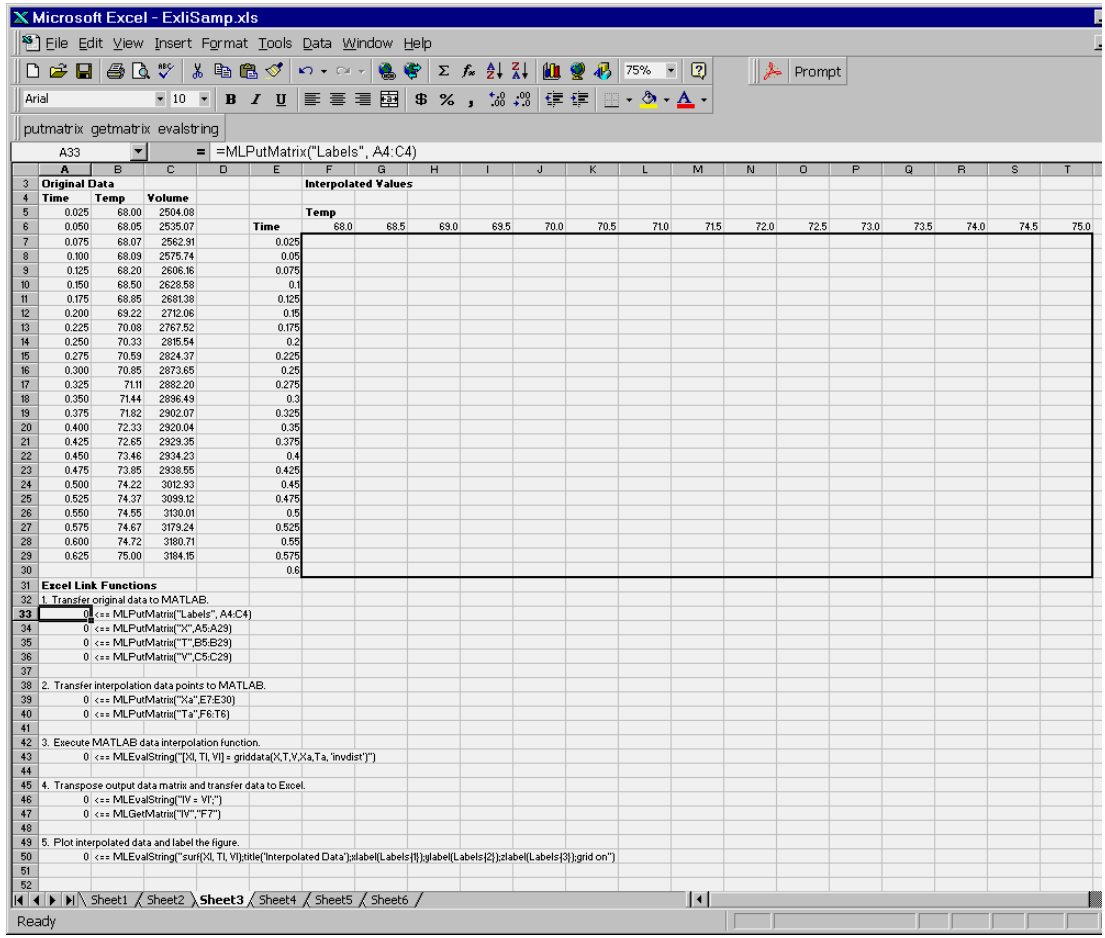
Example 2: Interpolating Data

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 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.

To try this example, click the `Sheet3` tab on `ExliSamp.xls`.



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:

1. Make `A33` the active cell. Press **F2**, then **Enter** to execute the Excel Link function that passes the `Time`, `Temp`, and `Volume` labels to MATLAB.

2. Make `A34` the active cell. Press **F2**, then **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.

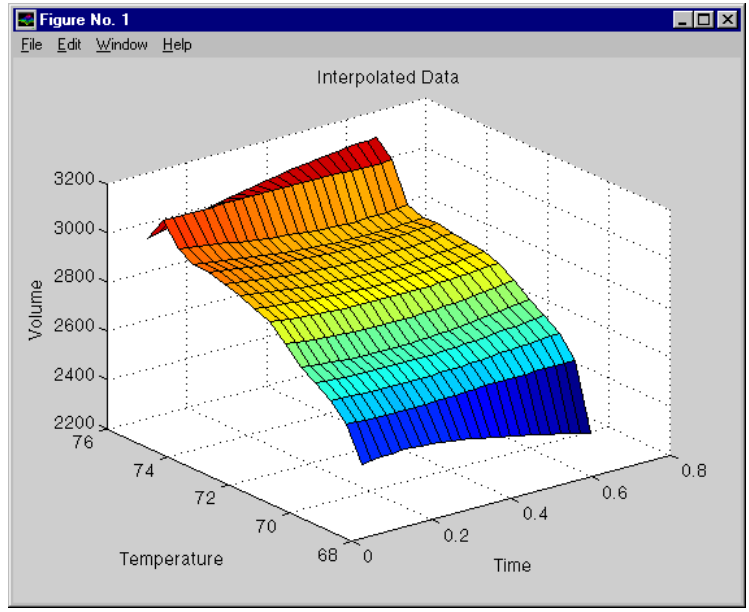
3 Move to cell `A39` and press **F2**, then **Enter** to copy the interpolation time values to MATLAB. Execute the function in cell `A40` to copy the interpolation temperature values.

4 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.

5 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	2779.96	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
31																

6 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 have finished with the example, close the figure window.

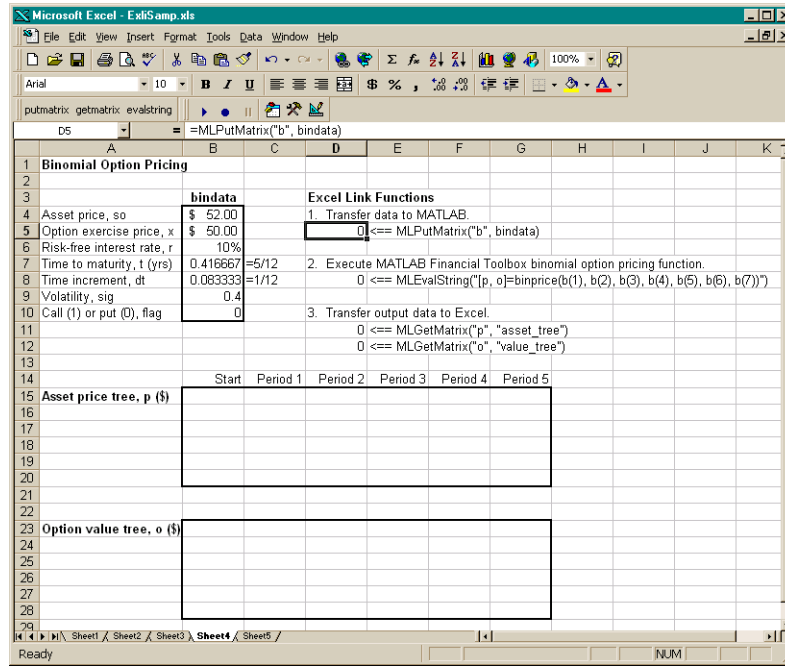
Example 3: Pricing a Stock Option with the Binomial Model

The MATLAB 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 use of the optional MATLAB Financial Toolbox.

Click the `Sheet4` tab on `ExliSamp.xls` to try this example.



The worksheet contains three named ranges:

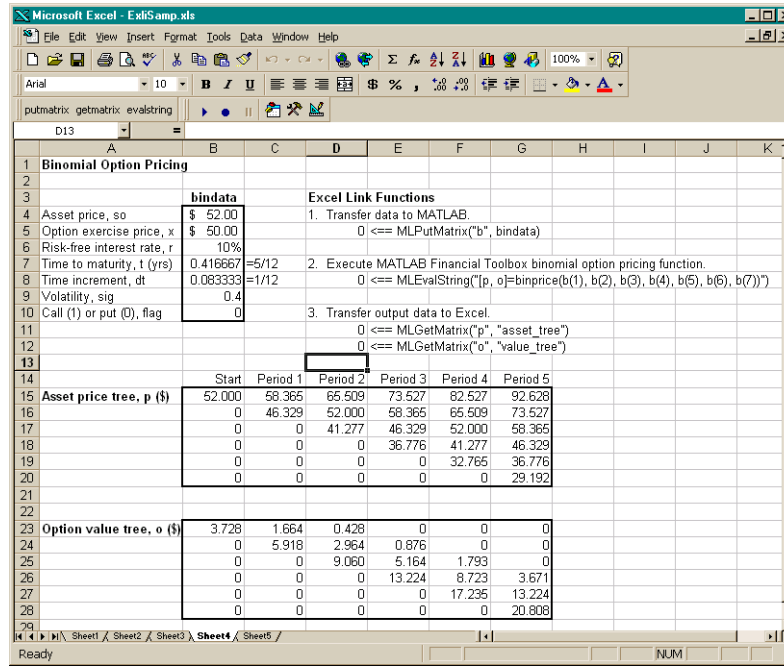
- `B4:B10` named `bindata`
- `B15` named `asset_tree`
- `B23` named `value_tree`

Also, two cells in `bindata` actually contain formulas:

- `B7` contains `=5/12`
- `B8` contains `=1/12`

Make `D5` the active cell. Press **F2**, then **Enter** to execute the Excel Link function that copies the asset data to MATLAB. 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.



	A	B	C	D	E	F	G	H	I	J	K
1	**Binomial Option Pricing**										
2											
3		**bindata**		**Excel Link Functions**							
4	Asset price, so	$ 52.00		1. Transfer data to MATLAB.							
5	Option exercise price, x	$ 50.00		0 <== MLPutMatrix("b", bindata)							
6	Risk-free interest rate, r	10%									
7	Time to maturity, t (yrs)	0.416667	=5/12	2. Execute MATLAB Financial Toolbox binomial option pricing function.							
8	Time increment, dt	0.083333	=1/12	0 <== MLEvalString("[p, o]=binprice(b(1), b(2), b(3), b(4), b(5), b(6), b(7))")							
9	Volatility, sig	0.4									
10	Call (1) or put (0), flag	0		3. Transfer output data to Excel.							
11				0 <== MLGetMatrix("p", "asset_tree")							
12				0 <== MLGetMatrix("o", "value_tree")							
13											
14		Start	Period 1	Period 2	Period 3	Period 4	Period 5				
15	**Asset price tree, p ($)**	52.000	58.365	65.509	73.527	82.527	92.628				
16		0	46.329	52.000	58.365	65.509	73.527				
17		0	0	41.277	46.329	52.000	58.365				
18		0	0	0	36.776	41.277	46.329				
19		0	0	0	0	32.765	36.776				
20		0	0	0	0	0	29.192				
21											
22											
23	**Option value tree, o ($)**	3.728	1.664	0.428	0	0	0				
24		0	5.918	2.964	0.876	0	0				
25		0	0	9.060	5.164	1.793	0				
26		0	0	0	13.224	8.723	3.671				
27		0	0	0	0	17.235	13.224				
28		0	0	0	0	0	20.808				
29											

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. Because this is a put, the option value is zero for prices significantly above the exercise price. Ignore the zeros that correspond to a zero in the price tree.

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.

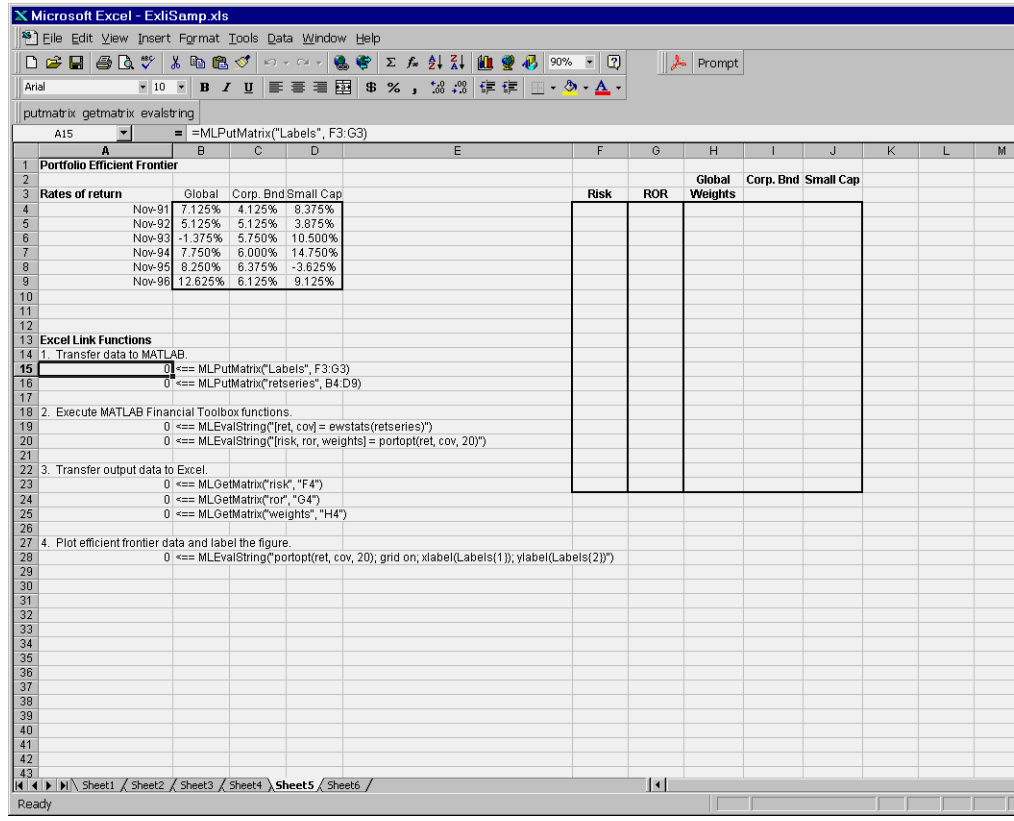
Example 4: Calculating and Plotting the Efficient Frontier of Financial Portfolios

MATLAB and the Financial Toolbox provide functions that compute and graph 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.

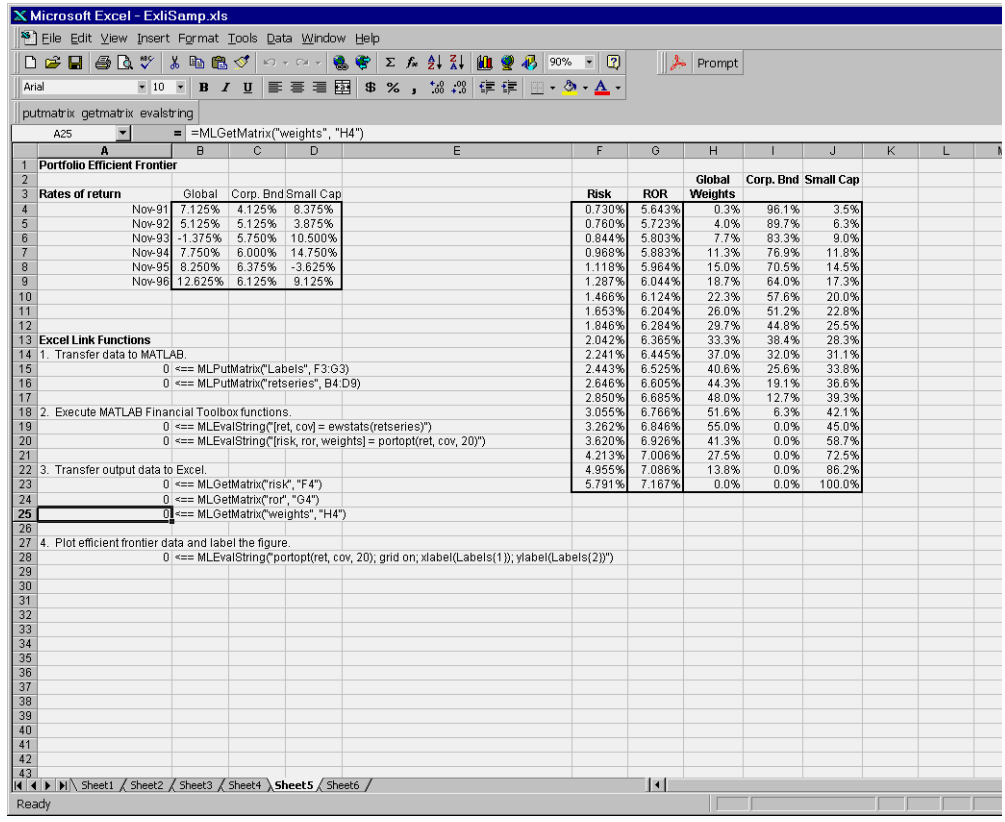
Note This example requires use of the optional MATLAB Financial Toolbox.

Click the `Sheet5` tab on `ExliSamp.xls` to try this example.



Make A15 the active cell. Press **F2**, then **Enter** to execute the Excel Link function that transfers the labels describing the outputs to be computed by MATLAB. Then make A16 the active cell to copy the actual portfolio return data to MATLAB. Execute the functions in A19 and A20 to compute the MATLAB Financial Toolbox efficient frontier function for 20 points along the frontier. Execute the Excel Link functions in A23, A24, and A25 to copy the output data to Excel.

The worksheet looks like this.

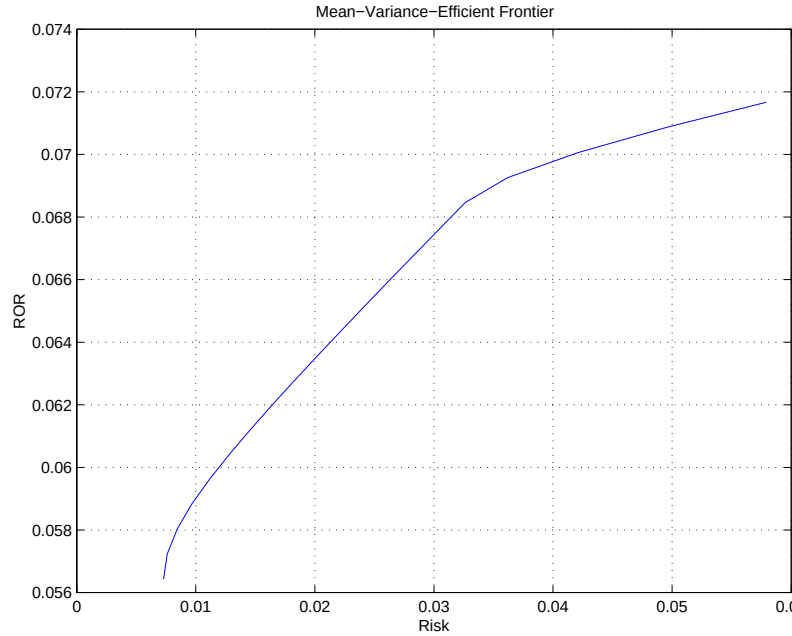


	A	B	C	D	E	F	G	H	I	J
1	**Portfolio Efficient Frontier**									
2								**Global**	**Corp. Bnd**	**Small Cap**
3	**Rates of return**	Global	Corp. Bnd	Small Cap		**Risk**	**ROR**	**Weights**		
4	Nov-91	7.125%	4.125%	8.375%		0.730%	5.643%	0.3%	96.1%	3.5%
5	Nov-92	5.125%	5.125%	3.875%		0.760%	5.723%	4.0%	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))")								

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 **Enter** to execute the Financial Toolbox function that plots the efficient frontier for the same portfolio data.

MATLAB displays a figure.



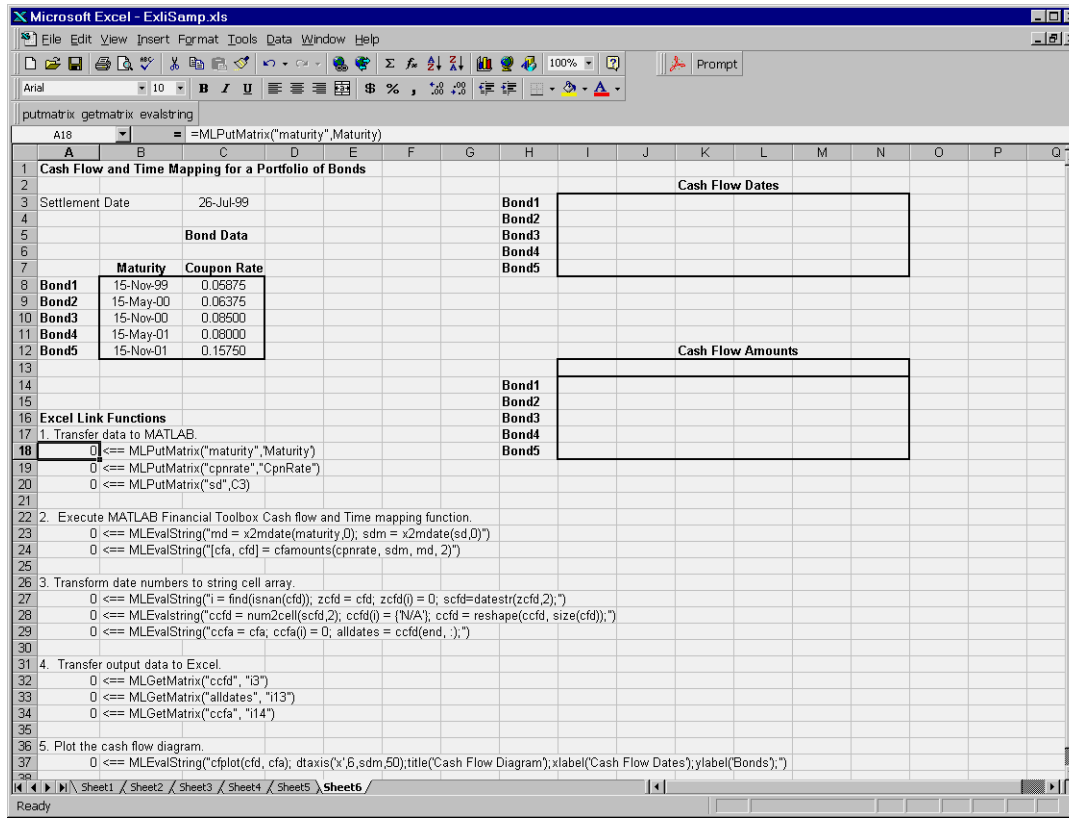
The light blue line shows the efficient frontier. Note the change in slope above a 6.8% return because the Corporate Bond portfolio no longer contributes to the efficient frontier.

To try 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.

Example 5: Bond Cash Flow and Time Mapping

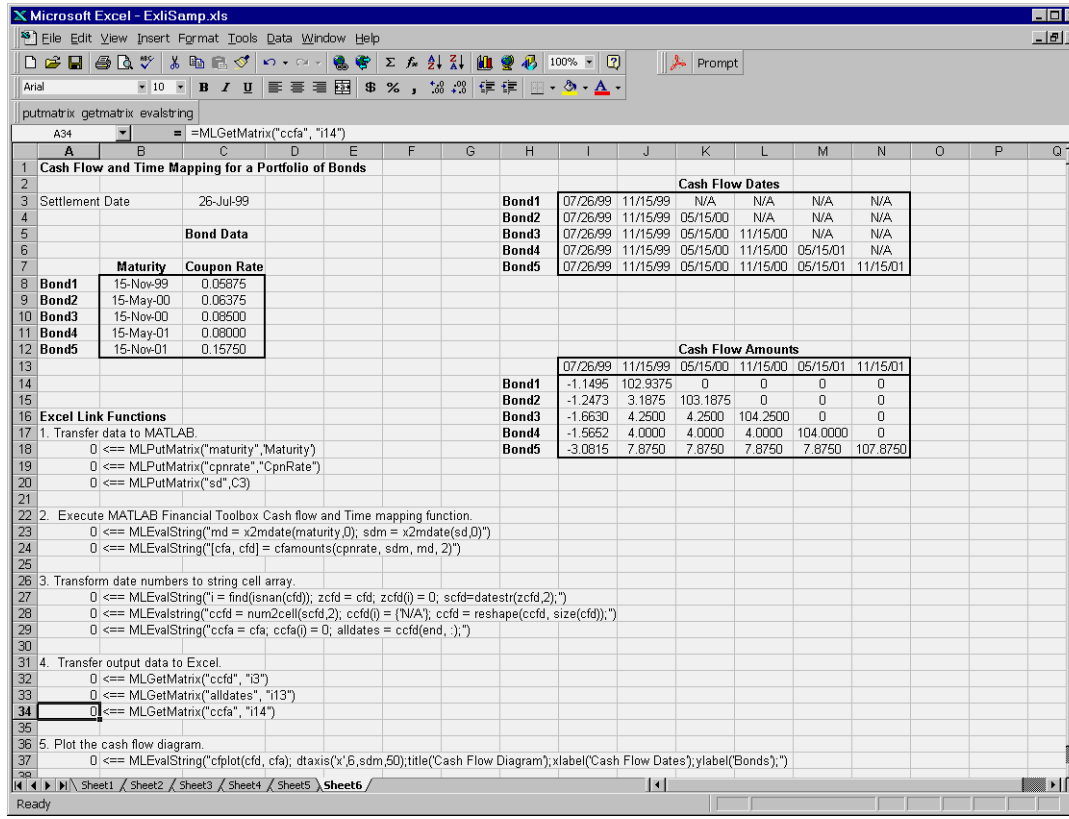
Example 5 illustrates the use of the MATLAB Financial Toolbox and Excel Link to compute a set of cash flow amounts and dates given a portfolio of five bonds whose maturity dates and coupon rates are known.

Click the `Sheet6` tab on `ExliSamp.xls` to try this example.



Make `A18` the active cell. Press **F2**, then **Enter** to execute the Excel Link function that transfers the column vector `Maturity` to MATLAB. Make `A19` the active cell to transfer the column vector `Coupon Rate` to MATLAB. Make `A20` the active cell to transfer the settlement date to MATLAB. Execute the functions in cells `A23` and `A24` to use the Financial Toolbox to compute cash flow

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



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

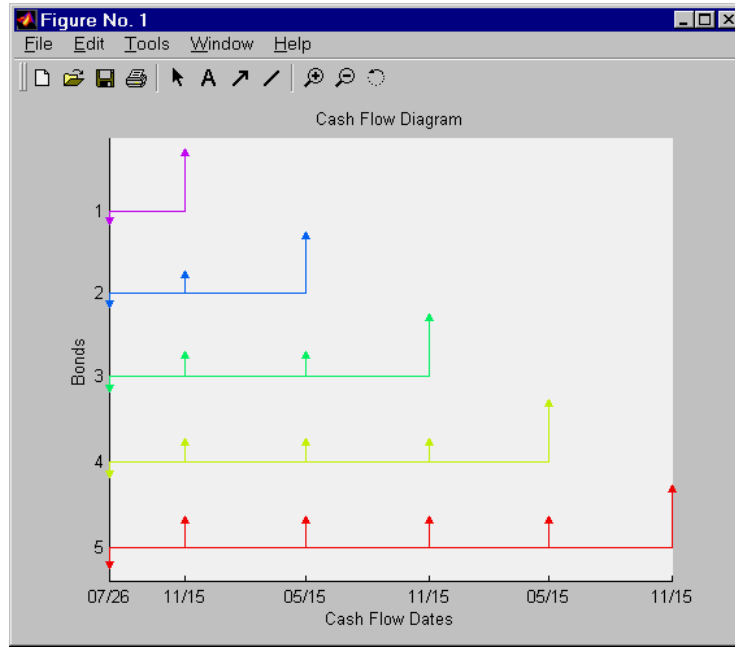


Figure No. 1
File Edit Tools Window Help
Cash Flow Diagram
Bonds
1
2
3
4
5
07/26
11/15
05/15
11/15
05/15
11/15
Cash Flow Dates

3

Function Reference

This chapter provides detailed descriptions of all Excel Link functions. It first groups the functions by task, then alphabetically.

Functions - Categorical List

Link Management Functions

`matlabinit`	Initialize Excel Link and start MATLAB process.
`MLAutoStart`	Automatically start MATLAB process.
`MLClose`	Terminate MATLAB process.
`MLOpen`	Start MATLAB process.

You can invoke any link management function except `matlabinit` as a worksheet cell formula or in a macro. You invoke `matlabinit` from the Excel **Tools Macro** menu or in a macro subroutine.

Data Management Functions

`matlabfcn`	Evaluate MATLAB command given Excel data.
`matlabsub`	Evaluate MATLAB command given Excel data and designate output location.
`MLAppendMatrix`	Create or append MATLAB matrix with data from Excel worksheet.
`MLDeleteMatrix`	Delete MATLAB matrix.
`MLEvalString`	Evaluate command in MATLAB.
`MLGetMatrix`	Write contents of MATLAB matrix in Excel worksheet.
`MLGetVar`	Write contents of MATLAB matrix in Excel VBA variable.
`MLPutMatrix`	Create or overwrite MATLAB matrix with data from Excel worksheet.
`MLPutVar`	Create or overwrite MATLAB matrix with data from Excel VBA variable.

You can invoke any data management function except `MLGetVar` and `MLPutVar` as a worksheet cell formula or in a macro. You can invoke `MLGetVar` and `MLPutVar` only in a macro.

Functions — Alphabetical List

This section contains function reference pages listed alphabetically.

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 quotes) .
`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 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 select **Macro**. In the **Macro Name/Reference** box, enter `matlabinit` and click **Run**. Or, include it 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 `MLAutoStart` to `"no"`. If `MLAutoStart` is set to `"yes"`, `matlabinit` executes automatically.

See Also

`MLAutoStart`, `MLOpen`

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 quotes) .
`edat`	Worksheet location where the function writes the contents of `var_name`. `"edat"` (in quotes) directly specifies the location and it must be a cell address or a range name. `edat` (without quotes) 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 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 quotes) directly specifies the matrix name. `var_name` (without quotes) 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 quotes). 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.

```
MLAppendMatrix("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`.

```
MLAppendMatrix(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.

Example

```
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 quotes) directly specifies the matrix name. `var_name` (without quotes) 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 quotes) directly specifies the command. `command` (without quotes) 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 the command string to MATLAB for evaluation. The specified action alters only the MATLAB workspace. Nothing is done in the Excel workspace.

Example

```
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`

MLGetMatrix

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 quotes) directly specifies the matrix name. `var_name` (without quotes) 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 quotes) directly specifies the location and it must be a cell address or a range name. `edat` (without quotes) 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 re-execute 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`.

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 quotes) directly specifies the matrix name. `ML_var_name` (without quotes) 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 quotes.

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`. Use `MLGetVar` only in a macro subroutine, not in a macro function or in a subroutine called by a function.

Example

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

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

See Also

`MLPutVar`

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.

Example

```
MLOpen()
```

starts the MATLAB process.

See Also

`matlabinit`, `MLClose`

MLPutMatrix

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 quotes) directly specifies the matrix name. `var_name` (without quotes) 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 quotes). 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 `NaN`s 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:

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

where `temp` is a named range in your worksheet.

Example

```
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`

MLPutVar

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 quotes) directly specifies the matrix name. `ML_var_name` (without quotes) 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 quotes.

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 `NaN`s within the MATLAB cell array.

Example

```
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`.

See Also

`MLGetVar`

A

Error Messages and Troubleshooting

Excel Cell Error Messages

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

Table A-1: Excel Cell Error Messages

Excel Cell Error Message	Meaning	Solution
`#COLS>256`	Your MATLAB variable exceeds the Excel limit of 256 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 3, "Function Reference."
`#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`.	See "Data Types" on page 1-11 for a list of supported MATLAB data types.

Table A-1: Excel Cell Error Messages (Continued)

Excel Cell Error Message	Meaning	Solution
`#MATLAB?`	You used an Excel Link function and MATLAB is not running.	Start Excel Link and MATLAB. See “Starting Excel Link” on page 1-5.
`#NAME?`	Excel doesn't recognize the function name. 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 Excel to Work with Excel Link” on page 1-3. 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>65536`	Your MATLAB variable exceeds the Excel limit of 65536 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 quotes (") may be missing, or you used single quotes (') instead of double quotes.	Verify and correct the function syntax. See Chapter 3, “Function Reference” for function syntax.

Table A-1: Excel Cell Error Messages (Continued)

Excel 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, you may ignore the message. Or, in this special case, remove `MatlabRequest` 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**.

Excel Error Message Boxes

Excel may display one of these error message boxes.

Table A-2: Excel Error Message Boxes

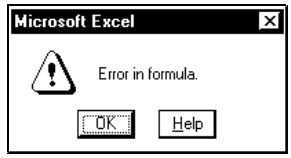
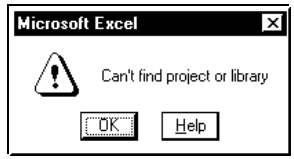
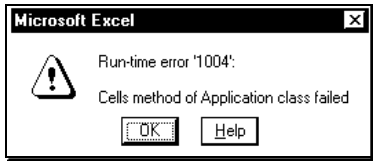
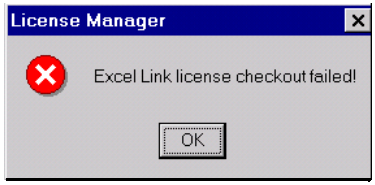
Excel Error Message Box	Meaning	Solution
	You entered a formula incorrectly. Common errors include a space between the function name and the left parenthesis; or missing, extra, or mismatched parentheses.	Check entry and correct typing errors.
	You tried to execute a macro and the location of `excllink.xla` is incorrect.	Click **OK**. The **References** window opens. Remove the check from `MISSING: excllink.xla`. Find `excllink.xla` in its correct location, check its box in the **References** window, and click **OK**.

Table A-2: Excel Error Message Boxes (Continued)

Excel Error Message Box	Meaning	Solution
	You used `MLGetMatrix` and the matrix is larger than the space available in the worksheet. This error destabilizes Excel Link and changes worksheet calculation mode to manual.	Click **OK**. Reset worksheet calculation mode to automatic and save your worksheet (if desired). Close Excel and MATLAB. Restart Excel, Excel Link, and MATLAB.
	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 Excel Link, contact your MathWorks representative.

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

Data in the MATLAB or Excel workspaces may exhibit these undesired characteristics.

Table A-3: 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 quotes around the data-location argument in `MLPutMatrix` or `MLAppendMatrix`.	Correct the syntax to remove quotes.
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.

B

Installed Files

Files and Directories

The Excel Link installation program creates the subdirectory `exlink` under `<matlab>/toolbox/`. This directory contains the files

- `excllink.xla`: Excel Link add-in
- `ExliSamp.xls`: Excel Link samples described in this manual

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

For all operating systems, the `C:\MATLAB\bin` directory should be on your system path. On Windows 2000 add the `C:\Winnt\system` and `C:\Winnt\system32` directories to your path.

Excel Link uses `Kernel32.dll`, which should already be in the appropriate Windows system directory (for example, `C:\Winnt\system32`).

Index