Excel Link

For Use with MATLAB®

Computation

Visualization

Programming

The MATH WORKS Inc.

User's Guide

Version 1.0.8

How to Contact The MathWorks:

508-647-7000 Phone

508-647-7001 Fax

The MathWorks, Inc.
24 Prime Park Way
Natick, MA 01760-1500 Mail

`http://www.mathworks.com` Web
`ftp.mathworks.com` Anonymous FTP server
`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
`subscribe@mathworks.com` Subscribing user registration
`service@mathworks.com` Order status, license renewals, passcodes
`info@mathworks.com` Sales, pricing, and general information

Excel Link User's Guide

Printing History: May 1996 First Printing for Excel Link 1.0
May 1997 Second Printing for Excel Link 1.0.3
January 1998 Revised (Online only)
January 1999 Revised for Excel Link 1.0.8

Contents

Preface

1 Using Excel Link

2 Solving Sample Problems with Excel Link

3 Reference

A Error Messages and Troubleshooting

B Installed Files

Preface

Setting Your Expectations

In designing Excel Link and this manual, we assume you are a knowledgeable and experienced user of Microsoft Windows, Microsoft Excel, and MATLAB®. If you are unfamiliar with any of this software, please consult the appropriate product documentation before using Excel Link.

After reading this manual, you will understand Excel Link concepts, content, functions, and uses. You will be able to use the functions of choice to develop integrated applications.

Documentation Conventions

We use some or all of these conventions in our manuals.

To Indicate	This Guide Uses	Example
Example code	`Monospace type` (Use **Code** tag.)	To assign the value 5 to `A`, enter `A = 5`
Function names/syntax	`Monospace type` (Use **Code** tag.) For syntax lines, use paragraph tag `Syntax`.	The `cos` function finds the cosine of each array element. Syntax line example is `MLGetVar ML_var_name`
Keys	**Boldface** with an initial capital letter (Use **Menu-Bodytext** tag.)	Press the **Return** key.
Mathematical expressions	Variables in *italics*. Functions, operators, and constants in standard type. (Use **EquationVariables** tag.)	This vector represents the polynomial $p = x^2 + 2x + 3$

MATLAB output	`Monospace type`
	(Use **Code** tag.)
		MATLAB responds with
		`A =`
		`5`

Menu names, menu items, and controls	**Boldface** with an initial capital letter (Use **Menu-Bodytext** tag.)	Choose the **File** menu.
New terms	NCS *italics* (Use **Body text-ital** tag.)	An *array* is an ordered collection of information.

1

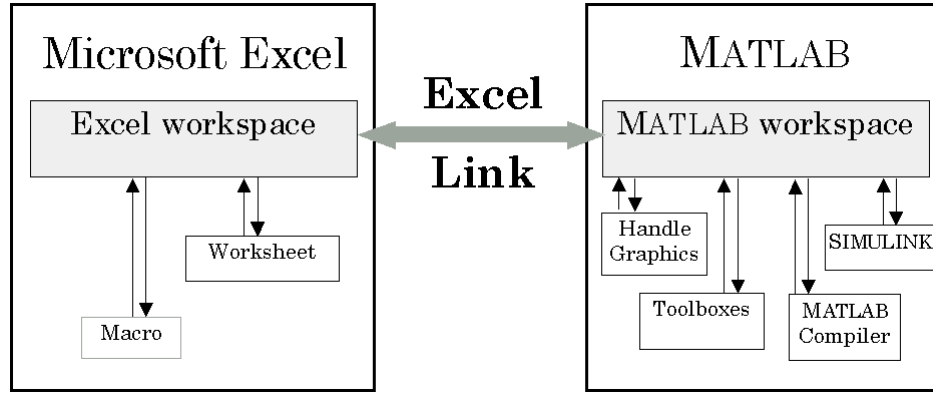
Using Excel Link

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 only 11 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. It requires Microsoft Windows 95 or Microsoft Windows NT 4.0; Microsoft Excel 97; 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 Guide* . 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 `C:\MATLAB\EXLINK` (or your path). Click **OK**.
4. Back in the **Add-Ins** window, make sure there's 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 **MATLAB Command Window** button to appear on the taskbar. Excel Link is now ready to use.

Starting Excel Link Automatically

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 the Reference chapter.

Starting Excel Link Manually

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 the Reference chapter.

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.You cannot connect a new Excel session to an existing MATLAB process unless you have started MATLAB with the `/automation` command line option.

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 the Reference chapter for details on each function.

Link Management Functions

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

`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 seven data management functions to copy data between Excel and MATLAB and to execute MATLAB commands from Excel:

`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 (Visual Basic for Applications) 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 `MLGetMatrix` and `MLGetVar` to copy data from MATLAB to Excel.

Note: `MLGetMatrix` argument syntax differs from the other data management functions. Please see the Reference chapter for details.

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

- 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)`.
- 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.
- 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`.

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 checking **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 re-execute other worksheet functions and generate unpredictable results.
- To recalculate Excel Link functions in a worksheet, re-execute 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, check 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 function is called from a subroutine.

Data Types

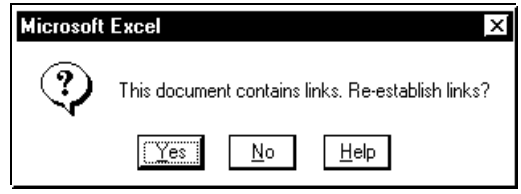
- Excel Link and Excel handle only two-dimensional numeric MATLAB arrays. They do not work with MATLAB multidimensional arrays, cell arrays, character arrays, or structures.
- Worksheet cells or VBA variables that pass to MATLAB must contain only numeric data, not strings.

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 "normal" for Excel. Simply ignore the messages, close any MATLAB figure windows, and re-execute the cell functions one at a time in the correct order by pressing **F2**, 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 `C:\MATLAB\EXLINK` (or your path) 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`.

2

Solving Sample Problems with Excel Link

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 the subdirectory `EXLINK` under your MATLAB directory (for example `C:\MATLAB\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 and 4 use functions in the optional MATLAB Financial Toolbox.

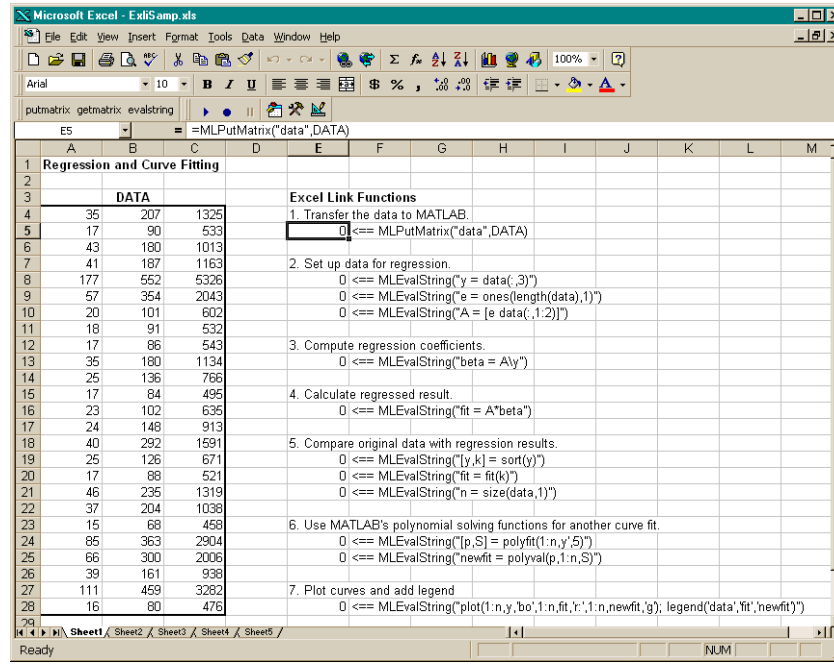
Example 1: Regression and Curve Fitting

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`.

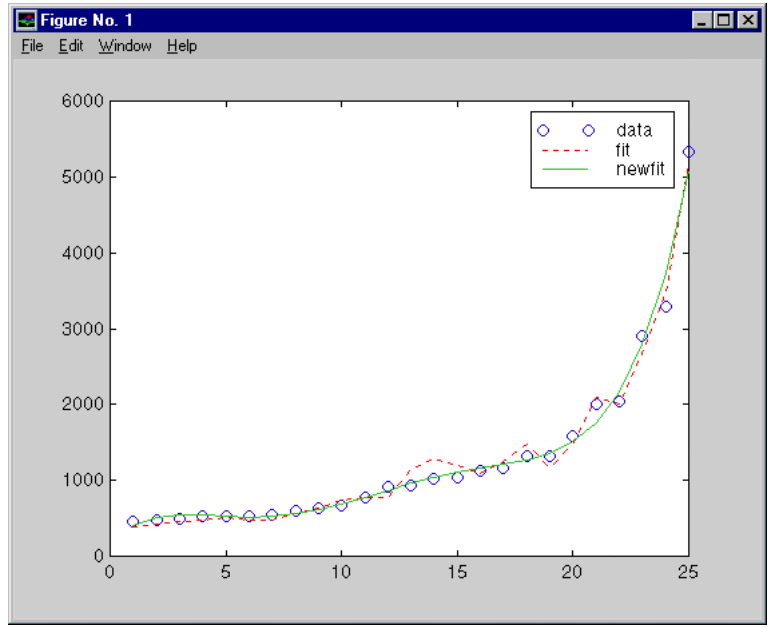
4. Execute the function in cell `E16`. MATLAB matrix-vector multiplication produces the regressed result (`fit`).

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.

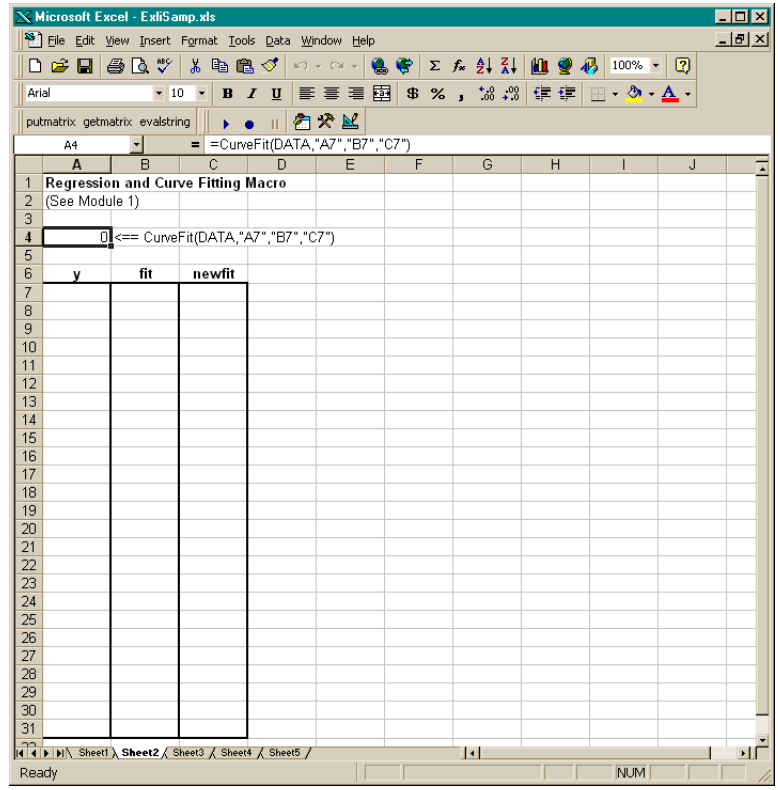


Since the data is closely correlated but not exactly linearly dependent, the `fit` curve (dashed line) shows a close, but not an exact, fit. The fifth-degree polynomial curve, `newfit`, represents a more accurate mathematical model for the data.

When you have finished with this version of the example, close the **Figure** window.

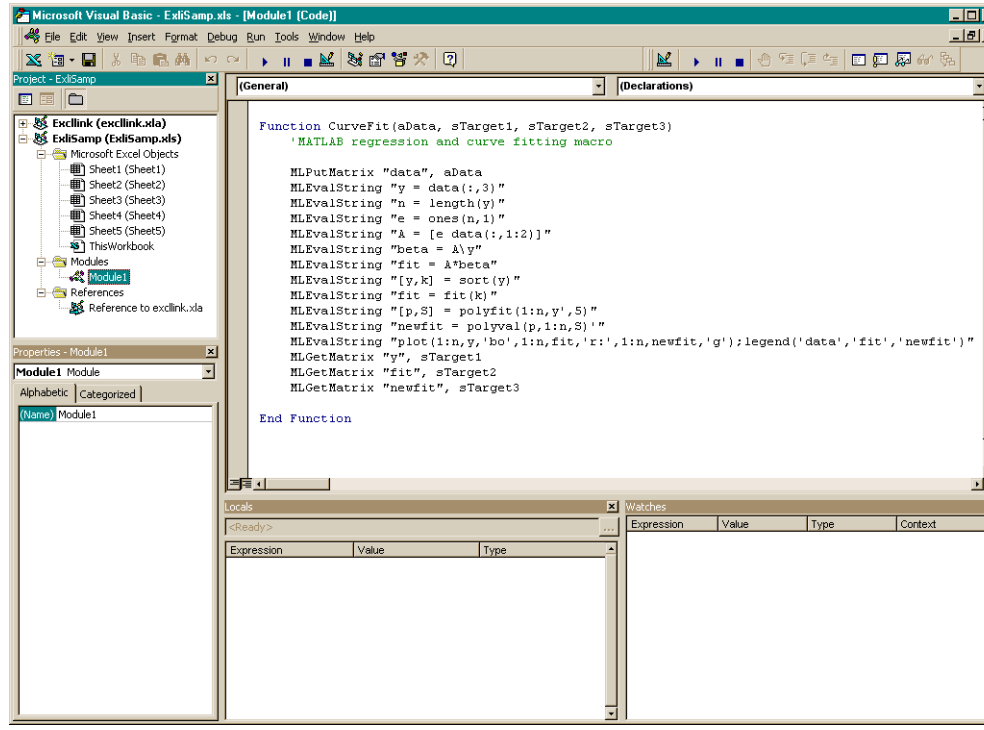
Macro Version

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



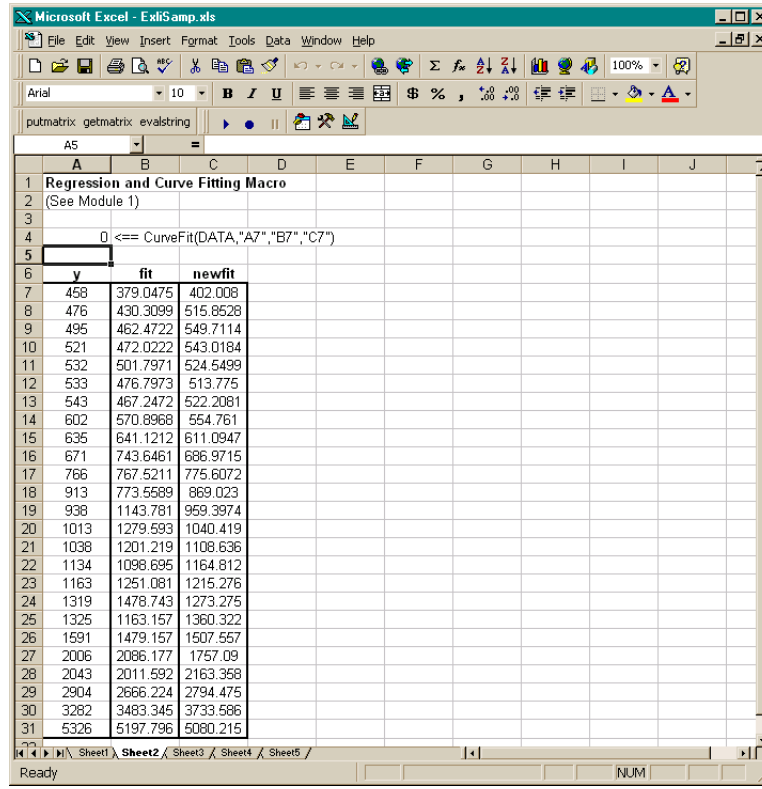
Make cell `A4` the active cell, but do not execute it yet.

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 with the example, close the **Figure** window.

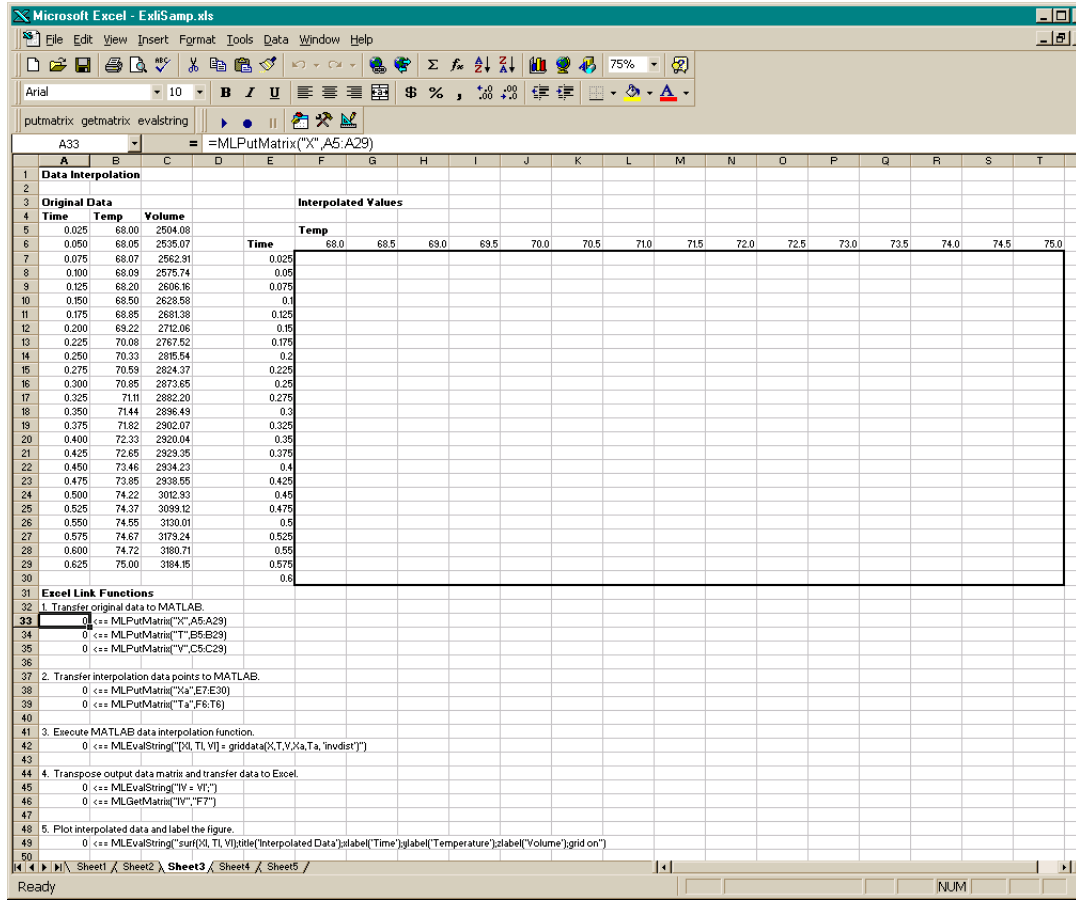
Example 2: Interpolating Data

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's 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 copies the original time data to MATLAB. Move to cell A34 and

execute the function to copy the original temperature data. Execute the function in cell `A35` to copy the original volume data.

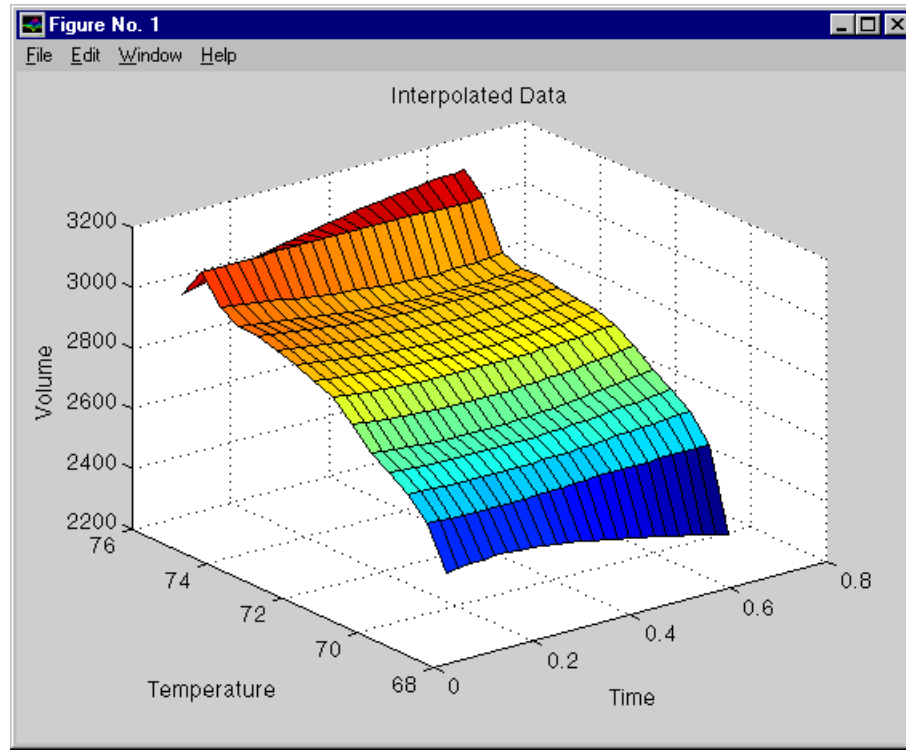
2 2 Move to cell `A38` and press **F2**, then **Enter** to copy the interpolation time values to MATLAB. Execute the function in cell `A39` to copy the interpolation temperature values.

3 Execute the function in cell `A42`. `griddata` is the MATLAB two-dimensional interpolation function that generates the interpolated volume data using the inverse distance method.

4 Execute the functions in cells `A45` and `A46` 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																

5 Execute the function in cell A49. 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.

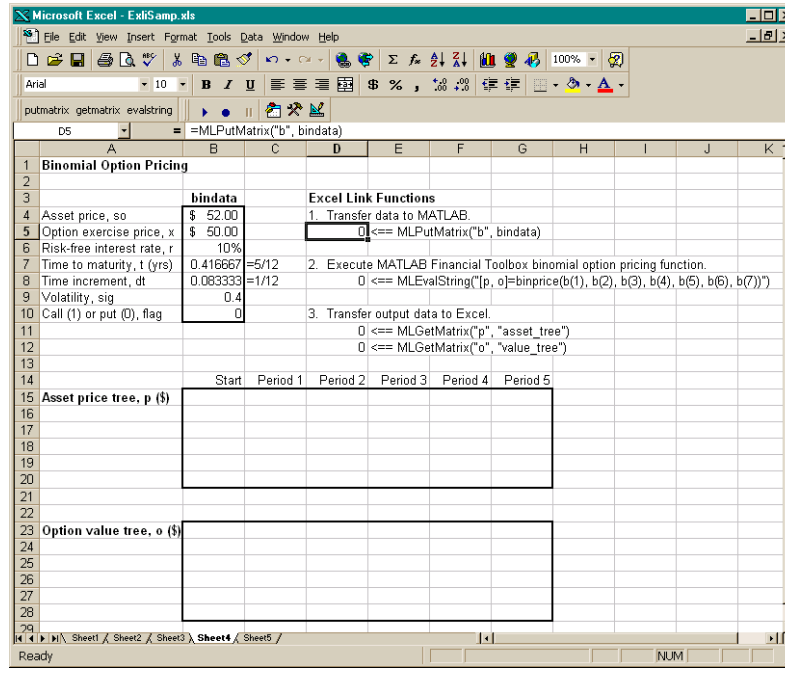
Example 3: Pricing a Stock Option with the Binomial Model

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 the optional MATLAB Financial Toolbox.

Click the `Sheet4` tab on `EXLISAMP.XLS` to try this example.



The worksheet contains three named ranges:

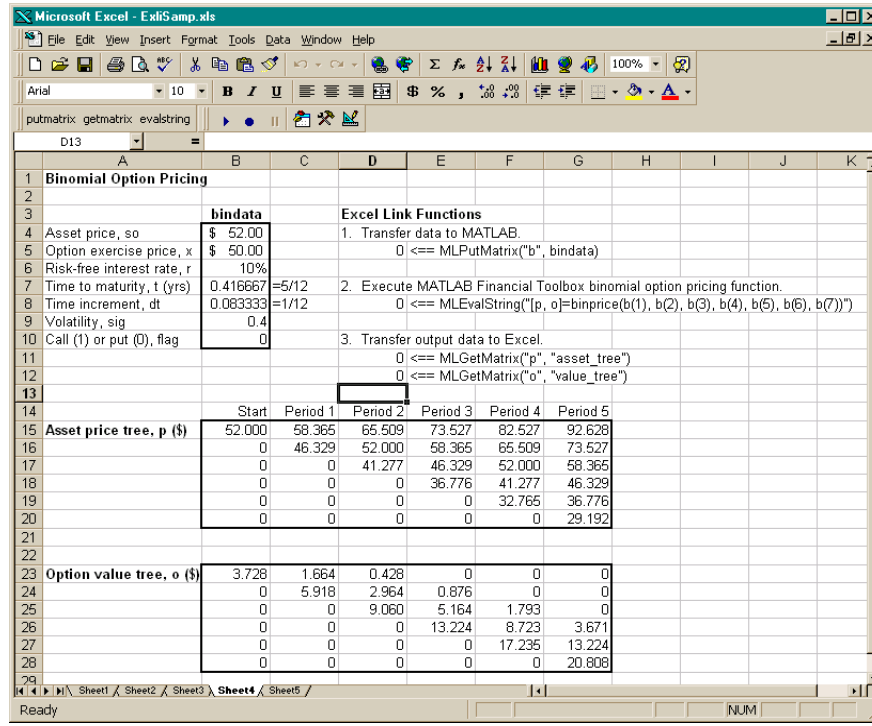
`B4:B10`	named	`bindata`
`B15`	named	`asset_tree`
`B23`	named	`value_tree`

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:



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 re-executing 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.

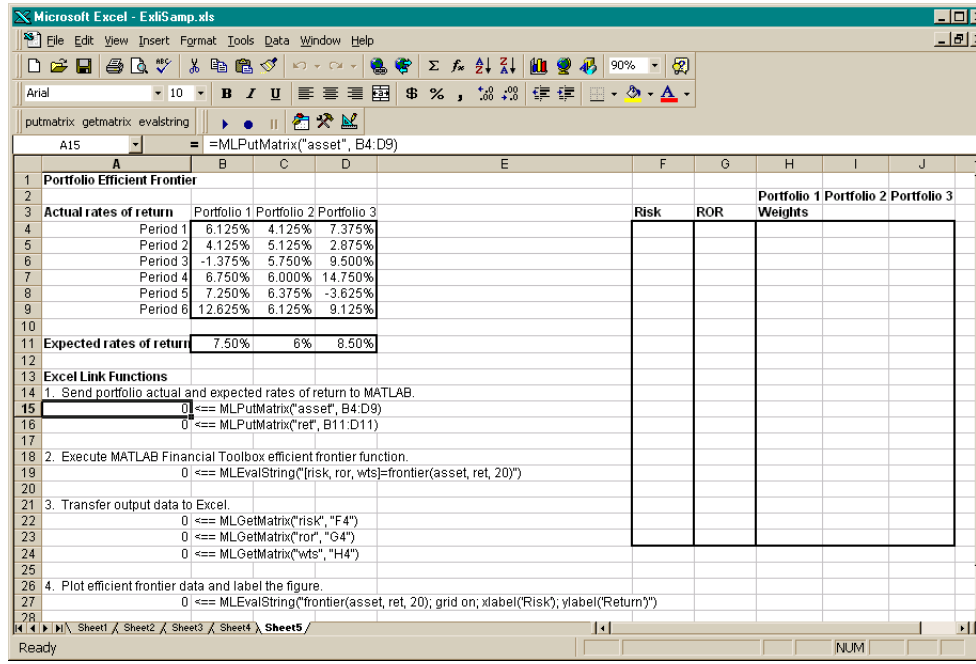
Example 4: Calculating and Plotting the Efficient Frontier of Financial Portfolios

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.

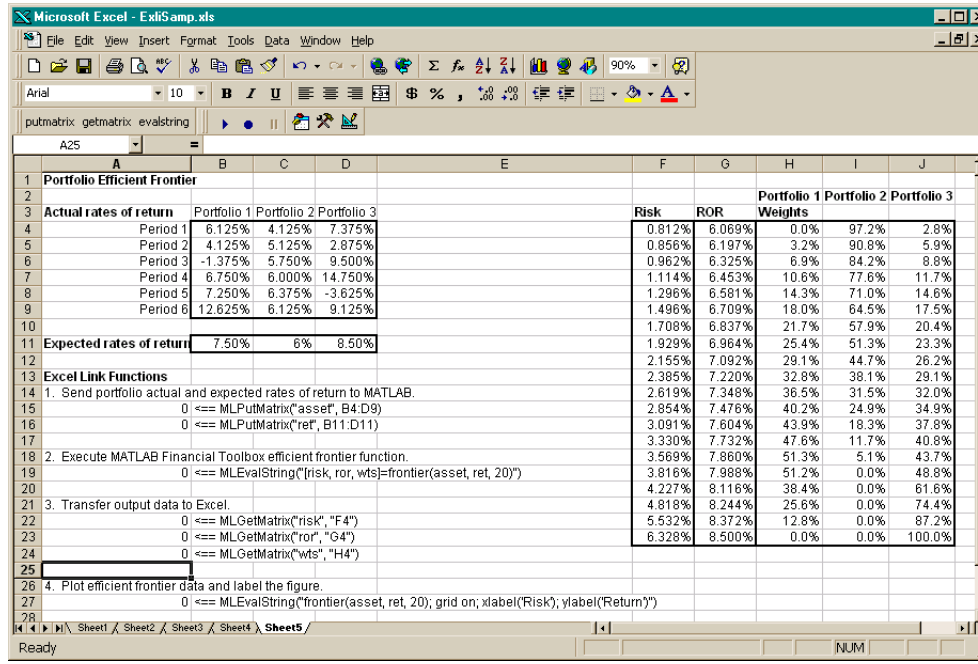
This example analyzes three portfolios, using rates of return for six time periods. In actual practice, these functions can analyze many portfolios over many time periods, limited only by the amount of computer memory available.

Note: This example requires 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 copies the actual portfolio return data to MATLAB. Move to A16 and execute the function that copies the expected returns to MATLAB. Move to A19 and execute the MATLAB Financial Toolbox efficient frontier function for 20 points along the frontier. Execute the Excel Link functions in A22, A23, and A24 to copy the output data to Excel. The worksheet looks like:



Microsoft Excel - ExliSamp.xls

File Edit View Insert Format Tools Data Window Help

putmatrix getmatrix evalstring

A25 =

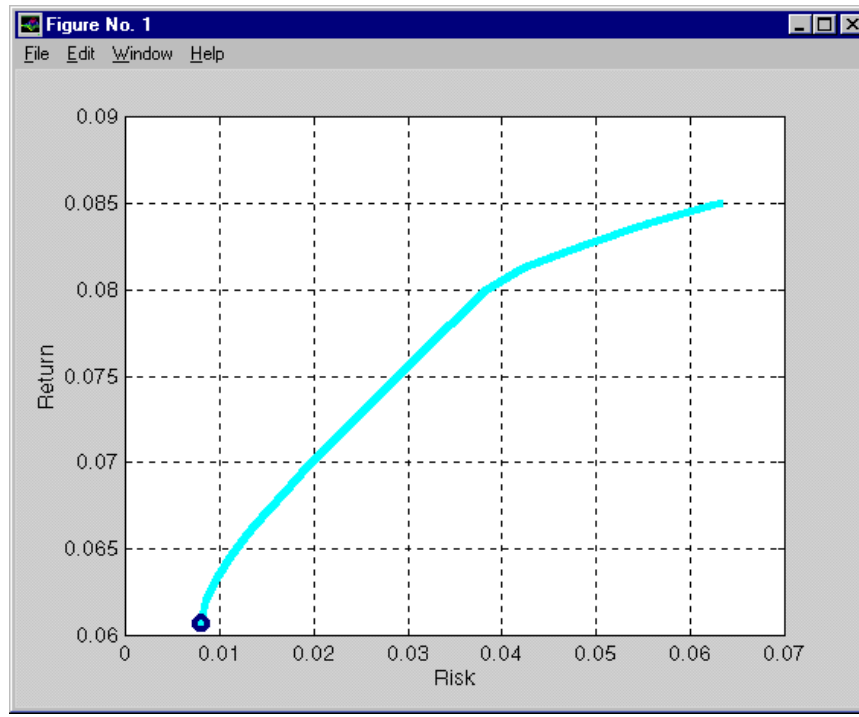
	A	B	C	D	E	F	G	H	I	J
1	**Portfolio Efficient Frontier**									
2								**Portfolio 1**	**Portfolio 2**	**Portfolio 3**
3	**Actual rates of return**	Portfolio 1	Portfolio 2	Portfolio 3		**Risk**	**ROR**	**Weights**		
4	Period 1	6.125%	4.125%	7.375%		0.812%	6.069%	0.0%	97.2%	2.8%
5	Period 2	4.125%	5.125%	2.875%		0.856%	6.197%	3.2%	90.8%	5.9%
6	Period 3	-1.375%	5.750%	9.500%		0.962%	6.325%	6.9%	84.2%	8.8%
7	Period 4	6.750%	6.000%	14.750%		1.114%	6.453%	10.6%	77.6%	11.7%
8	Period 5	7.250%	6.375%	-3.625%		1.296%	6.581%	14.3%	71.0%	14.6%
9	Period 6	12.625%	6.125%	9.125%		1.496%	6.709%	18.0%	64.5%	17.5%
10						1.708%	6.837%	21.7%	57.9%	20.4%
11	**Expected rates of return**	7.50%	6%	8.50%		1.929%	6.964%	25.4%	51.3%	23.3%
12						2.155%	7.092%	29.1%	44.7%	26.2%
13	**Excel Link Functions**					2.385%	7.220%	32.8%	38.1%	29.1%
14	1. Send portfolio actual and expected rates of return to MATLAB.					2.619%	7.348%	36.5%	31.5%	32.0%
15	0 <== MLPutMatrix("asset", B4:D9)					2.854%	7.476%	40.2%	24.9%	34.9%
16	0 <== MLPutMatrix("ret", B11:D11)					3.091%	7.604%	43.9%	18.3%	37.8%
17						3.330%	7.732%	47.6%	11.7%	40.8%
18	2. Execute MATLAB Financial Toolbox efficient frontier function.					3.569%	7.860%	51.3%	5.1%	43.7%
19	0 <== MLEvalString("[risk, ror, wts]=frontier(asset, ret, 20)")					3.816%	7.988%	51.2%	0.0%	48.8%
20						4.227%	8.116%	38.4%	0.0%	61.6%
21	3. Transfer output data to Excel.					4.818%	8.244%	25.6%	0.0%	74.4%
22	0 <== MLGetMatrix("risk", "F4")					5.532%	8.372%	12.8%	0.0%	87.2%
23	0 <== MLGetMatrix("ror", "G4")					6.328%	8.500%	0.0%	0.0%	100.0%
24	0 <== MLGetMatrix("wts", "H4")									
25										
26	4. Plot efficient frontier data and label the figure.									
27	0 <== MLEvalString("frontier(asset, ret, 20); grid on; xlabel('Risk'); ylabel('Return')")									
28										

Sheet1 Sheet2 Sheet3 Sheet4 **Sheet5**

Ready NUM

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 A27 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 dark blue circle shows the optimum portfolio allocation: the highest return with the least risk. The light blue line shows the efficient frontier. Note the change in slope above an 8% return as Portfolio 2 no longer contributes to the efficient frontier.

To try different data, close the **Figure** window and change the data in cells B4:D9 or B11:D11. Then re-execute all the Excel Link functions. The worksheet then shows the new frontier data, and MATLAB displays a new efficient frontier graph.

3

Reference

This chapter provides detailed descriptions of all Excel Link functions in alphabetical order. It first groups the functions by task.

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	
`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.

matlabinit

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`

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.

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 only numeric data. `mdat` cells that are empty become MATLAB matrix elements containing zero.

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`

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`

MLClose

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`

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.

MLEvalString

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`

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.

`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`

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.

`VBA_var_name` — Name of Excel 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 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`

MLOpen

Purpose

Start MATLAB process.

Syntax

Worksheet: `MLOpen(matlabpath)`

Macro: `MLOpen matlabpath`

`matlabpath` Path to MATLAB executable matlab.exe. `"matlabpath"` (in quotes) directly specifies the path. `matlabpath` (without quotes) is an indirect reference: the function evaluates the contents of `matlabpath` to get the path, and `matlabpath` must be a worksheet cell address or range name, or a macro variable name.

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("c:\matlab\bin")
```

starts the MATLAB process using matlab.exe found in the directory `c:\matlab\bin`.

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 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`. `mdat` must contain only numeric data. `mdat` cells that are empty become MATLAB matrix elements containing zero.

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 Excel 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`. `VBA_var_name` must contain only numeric data. Use `MLPutVar` only in a macro subroutine, not in a macro function or in a subroutine called by a function.

Example

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

creates or overwrites MATLAB matrix `K` with the data in the 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:

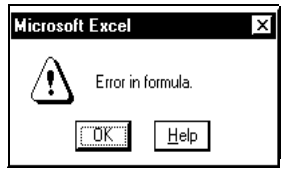
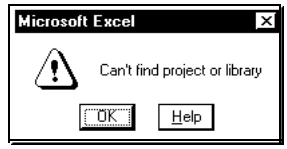
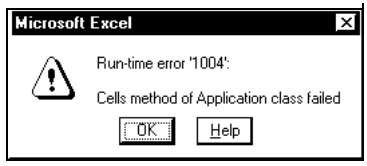
Excel Cell Error Message	Meaning	Solution
`#COMMAND!`	MATLAB does not recognize the command in an `MLEvalString` function. The command may be misspelled.	Check the spelling of the MATLAB command. Correct typing errors.
`#DIMENSION!`	You used `MLAppendMatrix` and the dimensions of the appended data do not match the dimensions of the matrix you want to append.	Check the matrix dimensions and the appended data dimensions, and correct the argument. See `MLAppendMatrix` in the Reference chapter.
`#MATLAB?`	You used an Excel Link function and MATLAB is not running.	Start Excel Link and MATLAB. See “Starting Excel Link Manually.”
`#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.” 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.	Check the spelling of the MATLAB matrix. Use the MATLAB `whos` command to display existing matrices. Correct typing errors.

#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.	Check and correct the function syntax. See the Reference chapter 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.
	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 "normal" for Excel. Simply ignore the messages, close any MATLAB figure windows, and re-execute 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:

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**.
	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.

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:

Data Error	Cause	Solution
MATLAB matrix cells contain zeros (0).	Corresponding Excel worksheet cells are empty.	Excel worksheet cells must contain only numeric 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

Installed Files

The Excel Link installation program creates the subdirectory `EXLINK` under the `MATLAB` directory (for example `C:\MATLAB\EXLINK`) containing the files:

`EXCLLINK.XLA`	Excel Link add-in
`EXLISAMP.XLS`	Excel Link samples described in this manual

It also creates an Excel Link initialization file, `EXLINK.INI`, in the appropriate Windows directory (for example `C:\WINNT`).

The `C:\MATLAB\bin` directory should be on your system path. On Windows 95 also add `C:\WIN95\SYSTEM` to your `PATH`. On Windows NT 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

N

R

S

T

V

W

Z