Spreadsheet Link™

User's Guide

MATLAB®

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

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

Contents

2 Solving Problems with the Spreadsheet Link Software

3
Error Messages and Troubleshooting

4
Functions — Alphabetical List

1

Getting Started

Spreadsheet Link Product Description

Use MATLAB from Microsoft Excel

Spreadsheet Link connects Excel® spreadsheet software with the MATLAB® workspace, enabling you to access the MATLAB environment from an Excel spreadsheet. With Spreadsheet Link software, you can exchange data between MATLAB and Excel, taking advantage of the familiar Excel interface while accessing the computational speed and visualization capabilities of MATLAB.

Key Features

- Data preprocessing, editing, and viewing in the familiar Excel environment
- Sophisticated analysis of Excel data using MATLAB and application toolboxes
- Delivery of Excel based applications, using MATLAB as a computational and graphics engine and Excel as an interface
- Interactive selection of available functions using the MATLAB Function Wizard
- Visual interface for customization of all Spreadsheet Link preferences

Installation

In this section...
"Product Installation" on page 1-3
"Supported Excel Versions" on page 1-3
"Files and Folders Created by the Installation" on page 1-4
"After You Upgrade the Spreadsheet Link Software" on page 1-4

To use Spreadsheet Link, you must install Microsoft Excel first, and then install Spreadsheet Link. Ensure that you use the correct MATLAB version based on the supported version of Excel. After Spreadsheet Link is installed on your computer, you must set up the Spreadsheet Link add-in in Excel.

Product Installation

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

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

To install the Spreadsheet Link add-in, you need administrator system privileges on the computer. Contact your system administrator to enable these privileges.

Supported Excel Versions

Use the following table to determine the correct MATLAB version to install, based on the version of Excel installed on your computer. Each row in the table matches the MATLAB version with the supported versions of Excel.

MATLAB Version	Excel Versions
R2016b and later	2019, 2016, 2013, 2010, 2007
R2013b	2013, 2010, 2007

MATLAB Version	Excel Versions
R2010b	2010, 2007
R2007a	2007

Files and Folders Created by the Installation

Note The MATLAB root folder, `matlabroot`, is where MATLAB is installed on your system.

The Spreadsheet Link installation program creates a subfolder under `matlabroot` `\toolbox\`. The `exlink` folder contains these files:

- `excllink.xlam`: The Spreadsheet Link add-in for Microsoft Excel
- `ExliSamp.xls`: Spreadsheet Link example files described in this documentation

After You Upgrade the Spreadsheet Link Software

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

1 Install the new version of MATLAB and Spreadsheet Link.
2 Start MATLAB and a Microsoft Excel session.
3 Configure the Spreadsheet Link software. For details, see "Add-In Setup" on page 1-6.
4 If you have existing workbooks with macros that use Spreadsheet Link, update references to Spreadsheet Link in each workbook.

To update the references in an existing workbook in Microsoft Excel:

1 In a Microsoft Excel session, open the Visual Basic® Editor window by clicking **Visual Basic** on the **Developer** tab. (If you do not find the **Developer** tab, see the Excel Help.)
2 In the left pane, select a module for which you want to update a reference.
3 From the main menu, select **Tools > References**.
4 In the References dialog box, select the **SpreadsheetLink2007_2010** check box.

5 Click **OK**.

See Also

More About

- "Add-In Setup" on page 1-6

Add-In Setup

In this section...
"Configure Microsoft Excel" on page 1-6
"Work with Excel Macros" on page 1-10
"Work with Microsoft Visual Basic Editor" on page 1-11

Configure Microsoft Excel

To enable the Spreadsheet Link add-in, start a Microsoft Excel session and follow these steps.

If you use Microsoft Excel 2007:

1

Click , the Microsoft Office button.

2 Click **Excel Options**. The Excel Options dialog box opens.

If you use Microsoft Excel 2010 and later versions:

1 Select **File** from the main menu.

2 Click **Options**. The Excel Options dialog box opens.

The next steps are the same for both versions:

1 Click **Add-Ins**.

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

3 Click **Go**. The Add-Ins dialog box opens.

4 Click **Browse**.

5 Select *matlabroot*\toolbox\exlink\excllink.xlam. (matlabroot returns the full path to the folder where MATLAB is installed.)

6 Click **Open**. In the Add-Ins dialog box, the **Spreadsheet Link for use with MATLAB and Excel** check box is selected.
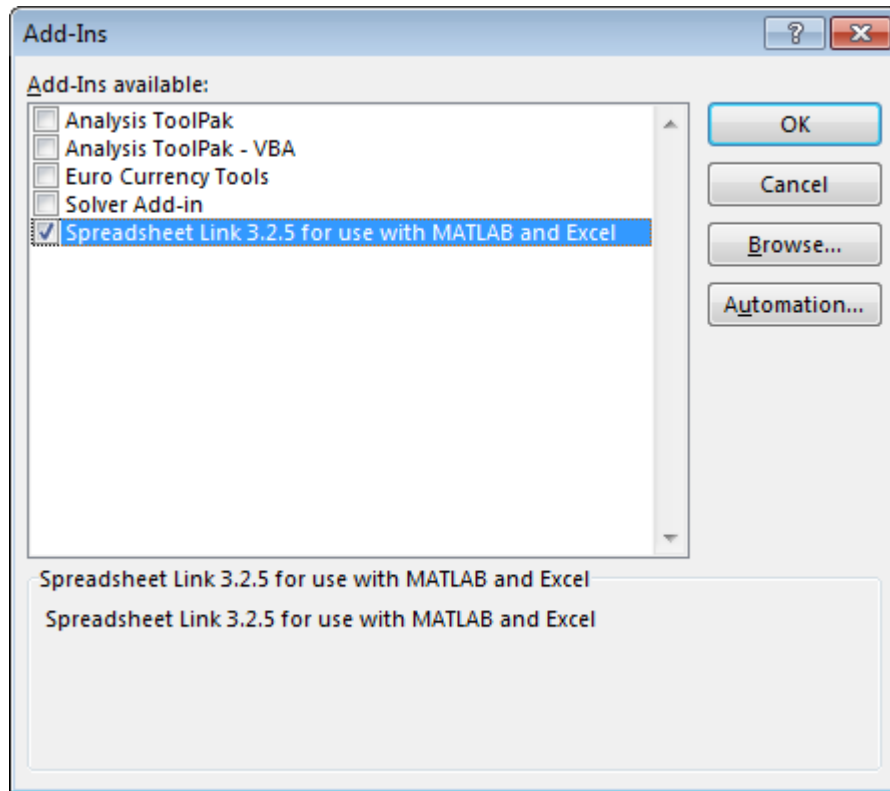
7   Click **OK** to close the Add-Ins dialog box.

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

The Spreadsheet Link add-in loads now and with each subsequent Excel session.

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

The MATLAB group appears on the top right of the **Home** tab in your Excel worksheet.
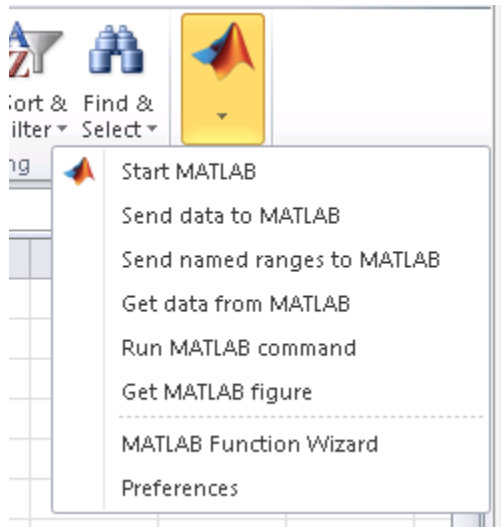
Spreadsheet Link is ready for use.

Right-click a cell to list the MATLAB options.

Calibri ▼ 11 ▼ A˄ A˅ $ ▼ % , ⟱		
B *I* ≡ 🪣 ▼ A ▼ ⬚ ▼ .0 .00 ▼		

✂	Cu**t**	
📋	**C**opy	
📋	**Paste Options:**	
	📋	
	Paste **S**pecial...	
	Insert...	
	Delete...	
	Clear Co**n**tents	
	Filt**e**r	▶
	Sort	▶
📑	Insert Co**mm**ent	
🖼	**F**ormat Cells...	
	Pic**k** From Drop-down List...	
	Define N**a**me...	
🌐	Hyper**l**ink...	
	MATLAB	▶

Send data to MATLAB
Send named ranges to MATLAB
Get data from MATLAB ▶
Run MATLAB command
Get MATLAB figure
Function Wizard
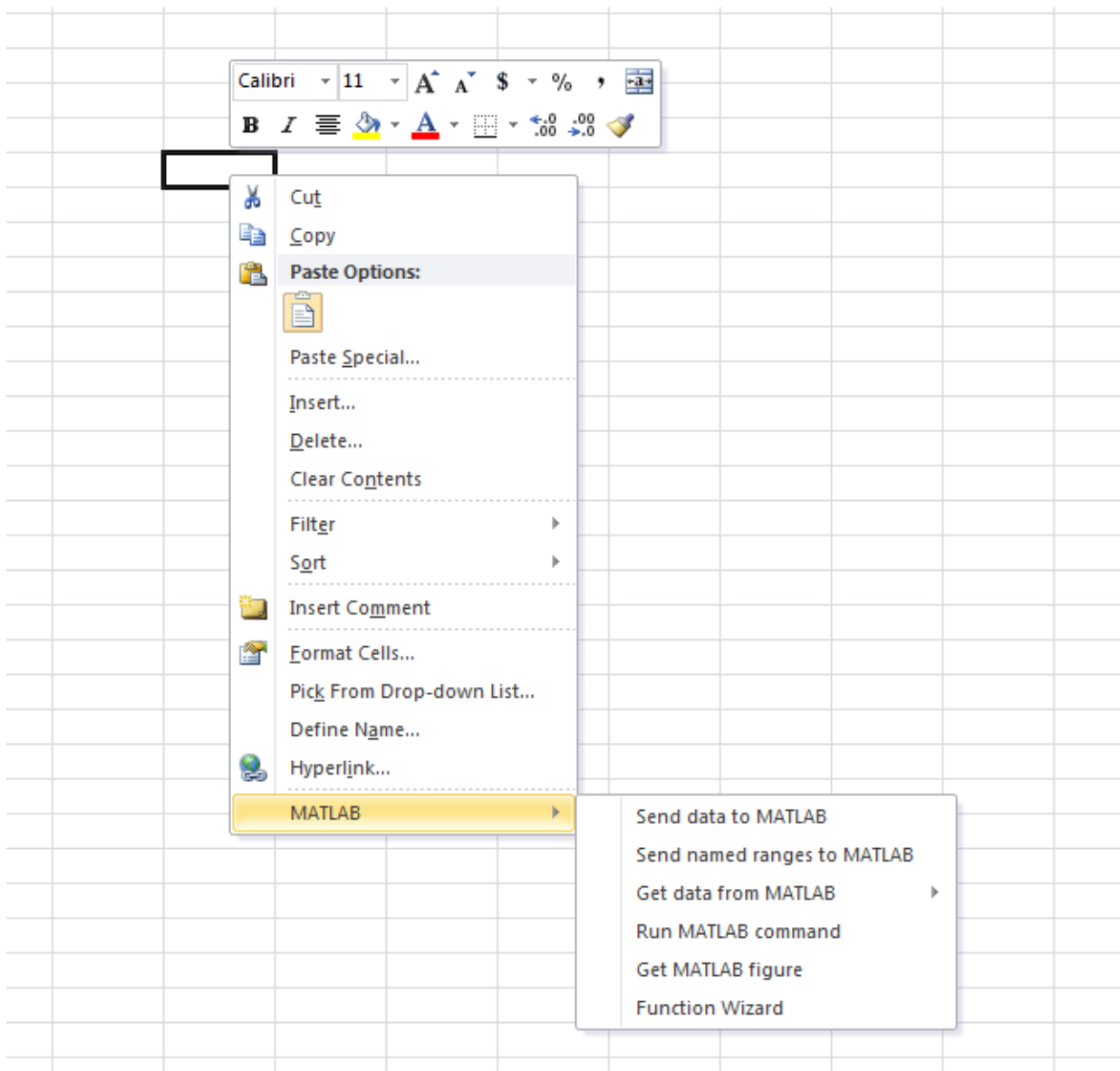
Note If the options are missing from the context menu and if the Trust Center dialog box has the **Require Application Add-ins to be signed by Trusted Publisher** check box selected, then you must click the **Enable Content** button for every session.

To check the settings in the Trust Center dialog box:

1 Click the **Developer** tab.

2 In the **Code** group, click the **Macro Security** button. The Trust Center dialog box opens.

3 Click **Add-ins**.

Work with Excel Macros

To work with Excel macros, follow these steps to enable the **Developer** tab on the Excel ribbon.

If you use Microsoft Excel 2007:

1

Click , the Microsoft Office button.

2 Click **Excel Options**. The Excel Options dialog box opens.

If you use Microsoft Excel 2010 and later versions:

1 Click the **File** menu.

2 Click **Options**. The Excel Options dialog box opens.

The next steps are the same for both versions:

1 Click **Customize Ribbon**.

2 From the **Main Tabs** list, select **Developer** and click **OK**. The Excel ribbon displays the **Developer** tab.

3 In the **Code** section of the **Developer** tab, click **Macros**. The Macro dialog box opens. Use this dialog box to run existing macros, create macros, and edit and delete macros.

Work with Microsoft Visual Basic Editor

To enable Spreadsheet Link as a Reference in the Microsoft Visual Basic Editor:

1   Open a Visual Basic session. Click the **Visual Basic** button on the **Developer** tab, or press **Alt+F11**.

> **Note** For instructions about displaying the **Developer** tab, see Excel Help.

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

3   In the References — VBA Project dialog box, select the **SpreadsheetLink** or **SpreadsheetLink2007_2010** check box.

4   Click **OK**.

See Also
`matlabroot`

More About

- "Installation" on page 1-3

Setting Spreadsheet Link Preferences

In this section...
"Preferences Dialog Box" on page 1-12
"Preferences in Worksheet Cells" on page 1-13

To control how Spreadsheet Link and MATLAB behave when Spreadsheet Link starts MATLAB in Microsoft Excel, you can set preferences with the Preferences dialog box or within individual worksheet cells.

Preferences Dialog Box

1   Click **Preferences** in the MATLAB group. The MATLAB group appears to the top right of the **Home** tab in your Excel worksheet.
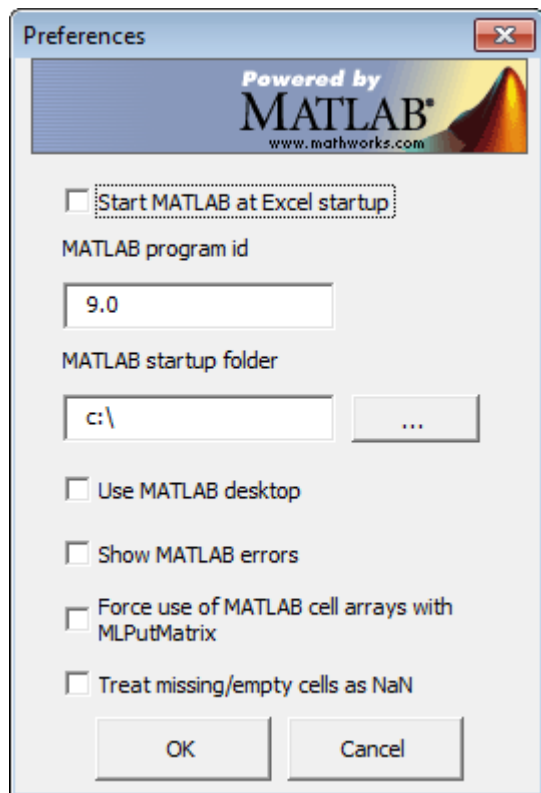
2 Set your preferences by selecting check boxes and filling in the text boxes. For the **MATLAB program id**, enter the MATLAB version as shown in the Windows registry. For the **MATLAB startup folder**, enter the full path of the startup folder.

Preferences in Worksheet Cells

To set a preference in a worksheet cell in Microsoft Excel, enter text that runs the corresponding Spreadsheet Link function in the worksheet cell. For example, to set the MATLAB version in a worksheet cell:

1 Set the MATLAB version to 9.0, which corresponds to MATLAB R2016a, by entering this text.

```
=MLProgramId("9.0")
```

1-13
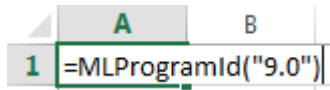
2 To run the function, press enter.

For details, see `MLProgramId`.

See Also

`MLPutMatrix`

More About

- "Worksheet Cell Errors" on page 3-2
- "Startup Errors" on page 3-13

Start and Stop Spreadsheet Link and MATLAB

In this section...
"Start Spreadsheet Link and MATLAB Automatically" on page 1-15
"Start Spreadsheet Link and MATLAB Manually" on page 1-15
"Connect to an Already Running MATLAB Session" on page 1-15
"Specify the MATLAB Startup Folder" on page 1-16
"Stop Spreadsheet Link and MATLAB" on page 1-17

Start Spreadsheet Link and MATLAB Automatically

When installed and configured according to the instructions in "Add-In Setup" on page 1-6, the Spreadsheet Link and MATLAB software automatically start when you start a Microsoft Excel session.

Start Spreadsheet Link and MATLAB Manually

1 Select **Tools > Macro**.

- In Excel 2007, click the **View** or **Developer** tab, and then click the **Macros** button.
- In Excel 2010, click the **View** menu and select **Macros** on the Excel toolstrip, and then click **View Macros**.

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

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

Connect to an Already Running MATLAB Session

By default, Spreadsheet Link starts a new MATLAB session. Alternatively, it can connect to an already running MATLAB session.

Note If several versions of MATLAB are installed on your computer, Spreadsheet Link always uses the last registered version. If you try to connect to an already running MATLAB session that is not the last registered version, Spreadsheet Link starts a new

MATLAB session. Spreadsheet Link does not connect to the existing one. To change the last registered version, see "Startup Errors" on page 3-13.

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

1   In MATLAB, enter the following command:

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

This command converts a running MATLAB session into an Automation server.

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

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

```
matlab -automation
```

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

```
matlab -automation -desktop
```

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

1   Right-click your MATLAB shortcut icon. (You can use the icon on your desktop or in the Windows **Start** menu.)

2   Select **Properties**.

3   Click the **Shortcut** tab.

4   Add `-automation` in the **Target** field. Remember to leave a space between `matlab.exe` and `-automation`.

5   Click **OK**.

For details, see "Manually Create Automation Server" (MATLAB).

Specify the MATLAB Startup Folder

MATLAB starts in the MATLAB root folder and completes the initialization. After starting, MATLAB changes to the Spreadsheet Link MATLAB startup folder. For details about specifying the startup folder, see `MLStartDir`.

Stop Spreadsheet Link and MATLAB

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

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

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

Create Diagonal Matrix Using Microsoft Excel Ribbon

This example shows how to execute Spreadsheet Link functions to export a named range in a worksheet to MATLAB and create a diagonal matrix using the Microsoft Excel ribbon.

The MATLAB group on the Microsoft Excel ribbon contains commands for common Spreadsheet Link functions. For the list of common functions, see "Executing Spreadsheet Link Functions" on page 1-42.

This example assumes that MATLAB is running after Microsoft Excel opens. For starting MATLAB, see "Start and Stop Spreadsheet Link and MATLAB" on page 1-15.

In a worksheet, enter the numbers 1 through 5 into the range of cells from `A1` through `E1`. Define the name `testData` for this range of cells and select it. For instructions, see Excel Help and enter the search term: define and use names in formulas.

The named range `testData` appears in the **Name Box**.
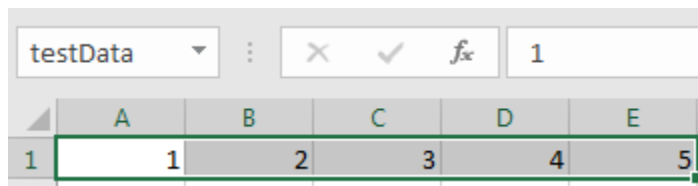
On the **Home** tab of the Microsoft Excel ribbon, click the MATLAB group in the top-right corner. Then, select **Send named ranges to MATLAB**. When you select this command, the software executes `MLPutRanges`.

Microsoft Excel exports the named range `testData` into the MATLAB variable `testData` in the MATLAB workspace.

Select the MATLAB group option **Run MATLAB Command**. When you select this command, Microsoft Excel displays a dialog box. Next, create a diagonal matrix. Use the `diag` function to specify `testData` as the input argument and `d` as the output argument. Enter this MATLAB command in the dialog box and click **OK**.

```
d = diag(testData)
```

The software executes the `MLEvalString` function. The MATLAB variable `d` appears in the MATLAB workspace and contains a diagonal matrix.

Retrieve the diagonal matrix into the worksheet by selecting cell A3. Select the MATLAB group option **Get data from MATLAB**. When you select this command, Microsoft Excel displays a dialog box. Retrieve the diagonal matrix in d by entering d in the dialog box and clicking **OK**. The software executes the MLGetMatrix function.

The diagonal matrix displays in cells A3 through E7.

	A	B	C	D	E
1	1	2	3	4	5
2					
3	1	0	0	0	0
4	0	2	0	0	0
5	0	0	3	0	0
6	0	0	0	4	0
7	0	0	0	0	5

See Also

MLEvalString | MLGetMatrix | MLPutRanges

More About

- "Create Diagonal Matrix Using Microsoft Excel Context Menu" on page 1-21
- "Create Diagonal Matrix Using Worksheet Cells" on page 1-25
- "Create Diagonal Matrix Using VBA Macro" on page 1-28
- "Executing Spreadsheet Link Functions" on page 1-42

Create Diagonal Matrix Using Microsoft Excel Context Menu

This example shows how to execute Spreadsheet Link functions to export a named range in a worksheet to MATLAB and create a diagonal matrix using the Microsoft Excel context menu.

The MATLAB group menu in the Microsoft Excel context menu contains commands for common Spreadsheet Link functions. For the list of common functions, see "Executing Spreadsheet Link Functions" on page 1-42.

This example assumes that MATLAB is running after Microsoft Excel opens. For details, see "Start and Stop Spreadsheet Link and MATLAB" on page 1-15.

In a worksheet, enter the numbers 1 through 5 into the range of cells from `A1` through `E1`. Define the name `testData` for this range of cells and select it. For instructions, see Excel Help and enter the search term: define and use names in formulas.

The named range `testData` appears in the **Name Box**.
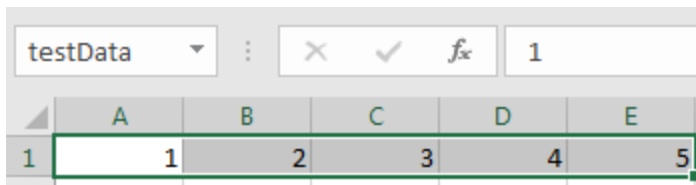
To find the command that exports the named range to MATLAB, right-click another cell outside of the named range in the worksheet. The Microsoft Excel context menu appears. To see the MATLAB group menu, select **MATLAB**.

Select **Send named ranges to MATLAB**. When you select this command, the software executes `MLPutRanges`.

Microsoft Excel exports the named range `testData` into the MATLAB variable `testData` in the MATLAB workspace.

Select the MATLAB group option **Run MATLAB Command**. When you select this command, Microsoft Excel displays a dialog box. Next, create a diagonal matrix. Use the `diag` function to specify `testData` as the input argument and `d` as the output argument. Enter this MATLAB command in the dialog box and click **OK**.

```
d = diag(testData)
```

The software executes the `MLEvalString` function. The MATLAB variable `d` appears in the MATLAB workspace and contains a diagonal matrix.

Retrieve the diagonal matrix into the worksheet. First select cell `A3`, and then select the MATLAB group option **Get data from MATLAB > d**. The software executes the `MLGetMatrix` function.

The diagonal matrix displays in cells `A3` through `E7`.

	A	B	C	D	E
1	1	2	3	4	5
2					
3	1	0	0	0	0
4	0	2	0	0	0
5	0	0	3	0	0
6	0	0	0	4	0
7	0	0	0	0	5

See Also

MLEvalString | MLGetMatrix | MLPutRanges

More About

- "Create Diagonal Matrix Using Microsoft Excel Ribbon" on page 1-18
- "Create Diagonal Matrix Using Worksheet Cells" on page 1-25
- "Create Diagonal Matrix Using VBA Macro" on page 1-28
- "Executing Spreadsheet Link Functions" on page 1-42

Create Diagonal Matrix Using Worksheet Cells

This example shows how to execute Spreadsheet Link functions to export a named range in the worksheet to MATLAB and create a diagonal matrix using Microsoft Excel worksheet cells.

The example assumes that MATLAB is running after Microsoft Excel opens. For details, see "Start and Stop Spreadsheet Link and MATLAB" on page 1-15.

In a worksheet, enter the numbers 1 through 5 into the range of cells from `A1` through `E1`. Define the name `testData` for this range of cells and select it. For instructions, see Excel Help and enter the search term: define and use names in formulas.

The named range `testData` appears in the **Name Box**.
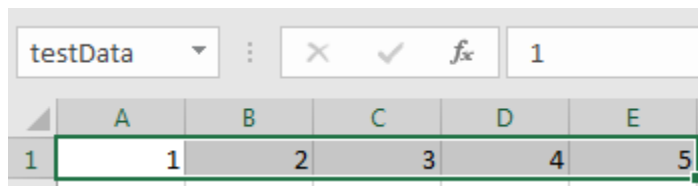
Enter the Spreadsheet Link function `MLPutRanges` directly into the worksheet cell as a worksheet formula. Double-click cell `A3`. Enter this text.

```
=MLPutRanges()
```

Press **Enter**. Microsoft Excel exports the named range `testData` into the MATLAB variable `testData` in the MATLAB workspace. After a Spreadsheet Link function successfully executes as a worksheet formula, the cell contains the value `0`. While the function executes, the cell shows the entered formula.
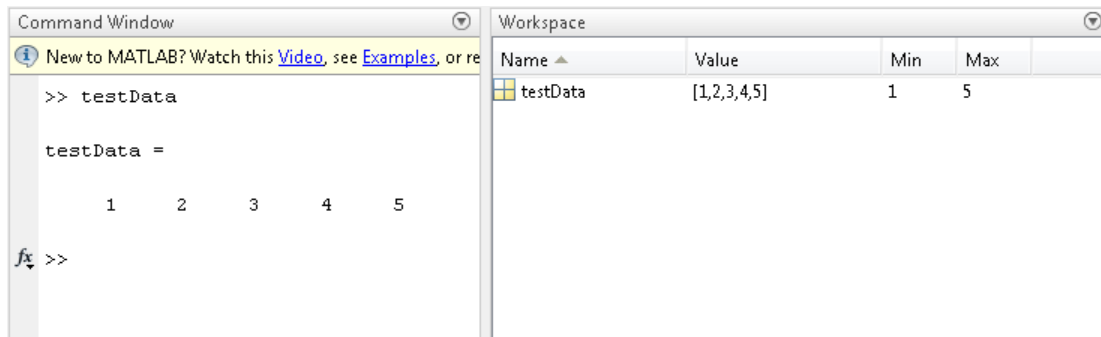
Double-click cell A5. Next, create a diagonal matrix. Use the `diag` function to specify `testData` as the input argument and `d` as the output argument. The Spreadsheet Link function `MLEvalString` executes the MATLAB command. Enter this text.

```
=MLEvalString("d = diag(testData);")
```

Press **Enter**. MATLAB executes the `diag` function. The MATLAB variable `d` appears in the MATLAB workspace and contains the diagonal matrix.

Double-click cell A7. Now retrieve the diagonal matrix into the worksheet using the Spreadsheet Link function `MLGetMatrix`. Enter this text.

```
=MLGetMatrix("d","A9")
```

The diagonal matrix displays in cell A9 through E13.

	A	B	C	D	E
1	1	2	3	4	5
2					
3	0				
4					
5	0				
6					
7	0				
8					
9	1	0	0	0	0
10	0	2	0	0	0
11	0	0	3	0	0
12	0	0	0	4	0
13	0	0	0	0	5

See Also

MLEvalString | MLGetMatrix | MLPutMatrix | MLPutRanges

More About

- "Create Diagonal Matrix Using Microsoft Excel Ribbon" on page 1-18
- "Create Diagonal Matrix Using Microsoft Excel Context Menu" on page 1-21
- "Create Diagonal Matrix Using VBA Macro" on page 1-28
- "Executing Spreadsheet Link Functions" on page 1-42

# Create Diagonal Matrix Using VBA Macro

This example shows how to execute Spreadsheet Link functions to export a named range in the worksheet to MATLAB and create a diagonal matrix using a Microsoft Excel VBA macro.
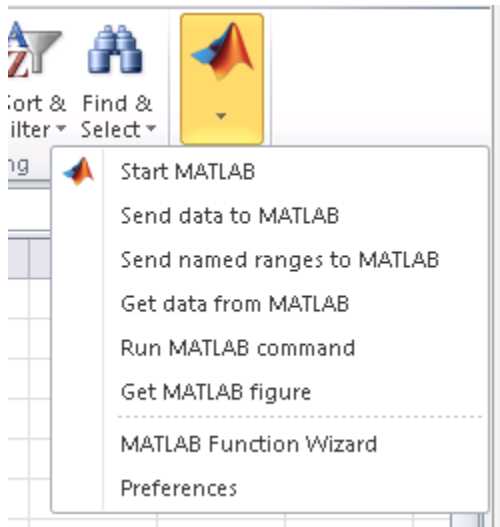
The example assumes that MATLAB is running after Microsoft Excel opens. For details, see "Start and Stop Spreadsheet Link and MATLAB" on page 1-15.

In a worksheet, enter the numbers 1 through 5 into the range of cells from A1 through E1. Define the name testData for this range of cells and select it. For instructions, see Excel Help and enter the search term: define and use names in formulas.

The named range testData appears in the **Name Box**.
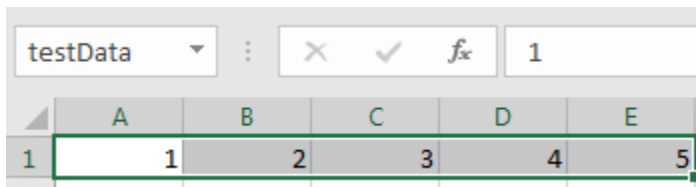
On the **Developer** tab in Microsoft Excel, click **Visual Basic**. The Visual Basic Editor window opens.

Insert a new module and create a diagonal matrix from the data in testData. To insert the module, select **Insert > Module**. In the Code section, enter this VBA code that contains a macro named Diagonal.

```
Sub Diagonal()
    MLPutRanges
    MLEvalString "b = diag(testData);"
    MLGetMatrix "b", "A3"
    MatlabRequest
End Sub
```

The Diagonal macro exports the named range into the MATLAB variable testData using the MLPutRanges function. Then, the macro uses the MLEvalString function to execute MATLAB code. The MATLAB code creates a diagonal matrix from the data in testData using the diag function. The code assigns the diagonal matrix to the MATLAB variable b. Then, the macro uses the MLGetMatrix function to import the diagonal matrix into the worksheet.

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

For details about working with modules, see Excel Help.

Run the macro by clicking **Run Sub/UserForm (F5)**. For details about running macros, see Excel Help.

The diagonal matrix displays in the worksheet cells A3 through E7.

	A	B	C	D	E
1	1	2	3	4	5
2					
3	1	0	0	0	0
4	0	2	0	0	0
5	0	0	3	0	0
6	0	0	0	4	0
7	0	0	0	0	5

See Also
MLEvalString | MLPutRanges | diag

More About
- "Create Diagonal Matrix Using Microsoft Excel Ribbon" on page 1-18
- "Create Diagonal Matrix Using Microsoft Excel Context Menu" on page 1-21
- "Create Diagonal Matrix Using Worksheet Cells" on page 1-25
- "Executing Spreadsheet Link Functions" on page 1-42

Find and Execute MATLAB Function Using MATLAB Function Wizard

This example shows how to find and execute the `triu` function using the MATLAB Function Wizard for Spreadsheet Link. You can use the Function Wizard to find any MATLAB function.

First, open the MATLAB Function Wizard from Microsoft Excel. From the **Home** tab, select the MATLAB group option **MATLAB Function Wizard**. Locate the `triu` MATLAB function, choose the function signature, and then execute it.
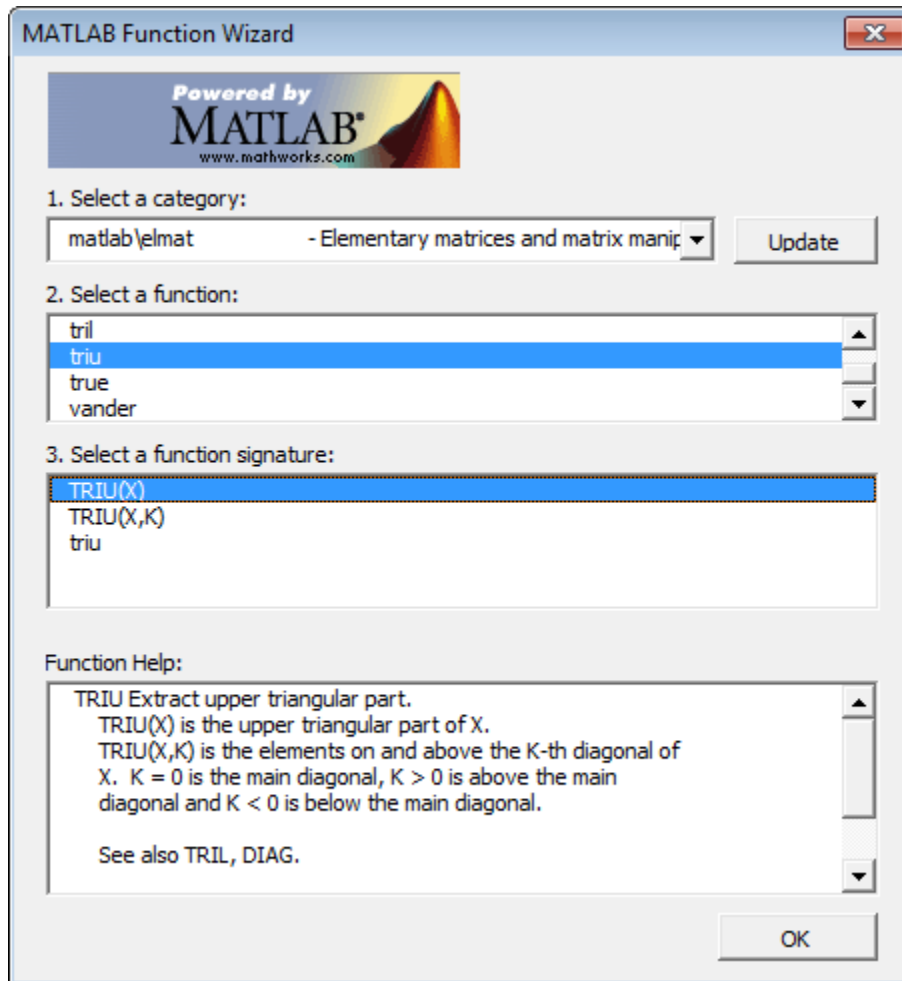
Alternatively, you can execute the function in a Microsoft Excel worksheet cell by using Spreadsheet Link function `matlabfcn` or `matlabsub`.

List Folders and Function Categories

All folders or categories in the current search path appear in the **Select a category** field of the MATLAB Function Wizard. Click an entry to select it. Each entry in the list appears as a folder path and a description read from the `Contents.m` file in that folder. If no `Contents.m` file is found, the category list displays contains this message:

finance\finsupport -(No table of contents file)

Refresh and Select Category

Click **Update** to refresh the category list. For details about the search path, see `path`.

The **Select a function** field displays a list of available functions for that category. Click the function name you want to execute. For help with the selected function, view the **Function Help** field.

Select Function Signature and Enter Formula

The **Select a function signature** field displays available signatures for that function. Click a function signature to select it. The Function Arguments dialog box appears.
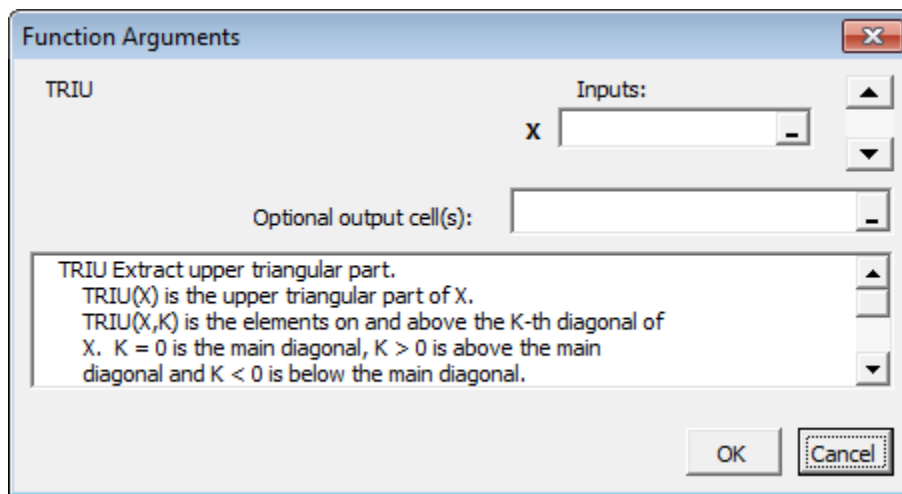
Specify the worksheet cell that contains the input argument **X**. By default, the output of the selected function appears in the current worksheet cell using the Spreadsheet Link function `matlabfcn`.

See Also

`matlabfcn | matlabsub`

More About

- "Find Custom MATLAB Function Using MATLAB Function Wizard" on page 1-34
- "Executing Spreadsheet Link Functions" on page 1-42

Find Custom MATLAB Function Using MATLAB Function Wizard

This example shows how to write a custom function and find it using the MATLAB Function Wizard for Spreadsheet Link. To execute MATLAB functions using the MATLAB Function Wizard, see "Find and Execute MATLAB Function Using MATLAB Function Wizard" on page 1-30.

Create and save a custom function in MATLAB. First, create a help header in the function that contains supported function signatures to use with the MATLAB Function Wizard. Write the function that calculates the Fibonacci numbers, and then save the function in the folder Documents\MATLAB.

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

For writing MATLAB functions, see "Create Functions in Files" (MATLAB).

Add the folder where you saved the function to the MATLAB search path. You can use the pathtool function or select **Set Path** in the MATLAB Toolstrip.

Open the MATLAB Function Wizard in Microsoft Excel using either the Microsoft Excel ribbon or context menu. Select the folder where you saved your function.
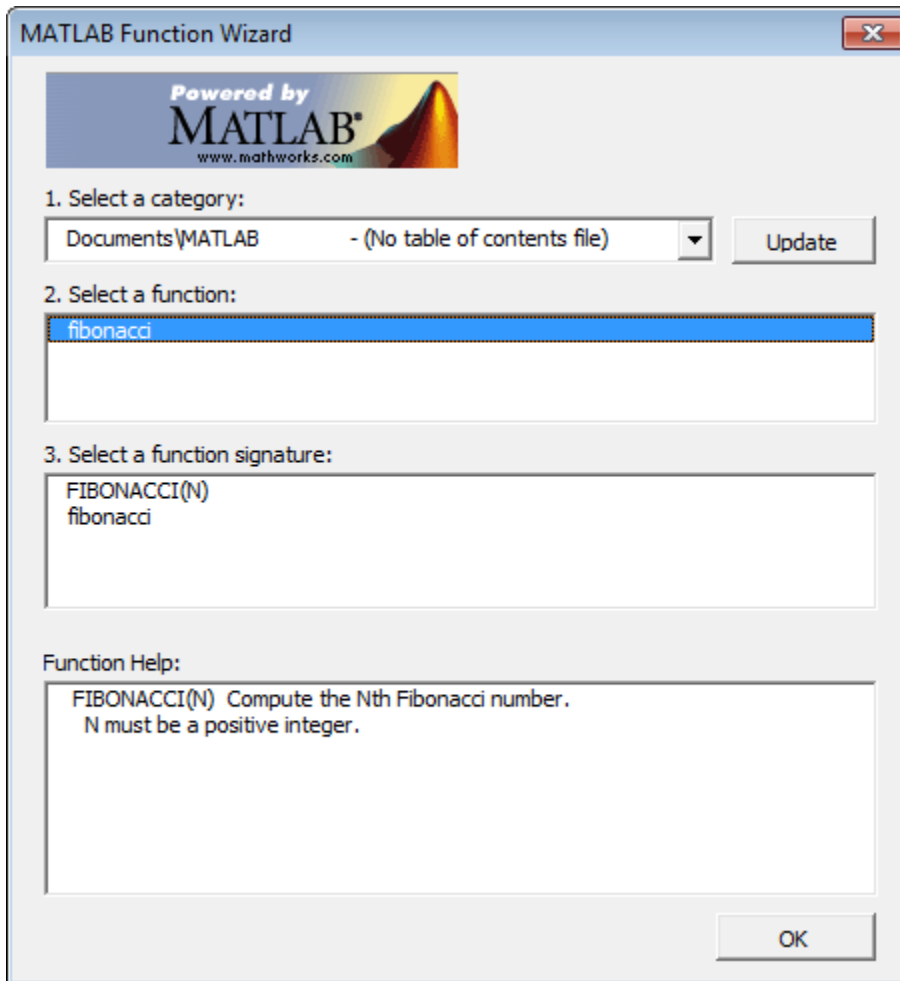
To execute this function, follow the steps in "Find and Execute MATLAB Function Using MATLAB Function Wizard" on page 1-30.

See Also

`matlabfcn` | `matlabsub`

More About

Return Multiple Output Arguments from MATLAB Function

This example shows how to execute a MATLAB function that returns multiple output arguments in Microsoft Excel using a Microsoft Excel VBA macro. The macro writes multiple output arguments from the MATLAB workspace to Microsoft Excel cells.

This example calculates the singular value decomposition of a matrix using `svd`.

In the Microsoft Excel cells from A1 through C3, create a range of data. Enter numbers from 1 through 3 in cells A1 through A3. Enter numbers from 4 through 6 in cells B1 through B3. Enter numbers from 7 through 9 in cells C1 through C3.

Create a Microsoft Excel VBA macro named `applysvd`. For details about creating macros, see Excel Help.

```
Public Sub applysvd()
    MLOpen
    MLPutMatrix "x", Range("A1:C3")
    MLEvalString ("[u,s,v] = svd(x);")
    MLGetMatrix "u", "A5"
    MLGetMatrix "s", "A9"
    MLGetMatrix "v", "A13"
    MatlabRequest
    MLClose
End Sub
```

The macro:

1 Starts MATLAB.

2 Sends the data in the A1 through C3 cell range to the MATLAB workspace and assigns it to the MATLAB variable x.

3 Runs svd with the input argument x and output arguments u, s, and v.

4 Individually retrieves data for one output argument into a specific Microsoft Excel cell while accounting for the size of each output data matrix to avoid overwriting data. For the first output argument, the macro retrieves the data for the output argument u into cell A5.

5 Closes MATLAB.

Run applysvd. MATLAB runs svd and populates the specified cells with data from the three output arguments.

A5						f_x

	A	B	C
1	1	2	3
2	4	5	6
3	7	8	9
4			
5	-0.2148	0.8872	0.4082
6	-0.5206	0.2496	-0.8165
7	-0.8263	-0.3879	0.4082
8			
9	16.8481	0.0000	0.0000
10	0.0000	1.0684	0.0000
11	0.0000	0.0000	0.0000
12			
13	-0.4797	-0.7767	0.4082
14	-0.5724	-0.0757	-0.8165
15	-0.6651	0.6253	0.4082
16			

For details about running macros, see Excel Help.

See Also

MLClose | MLEvalString | MLGetMatrix | MLOpen | MLPutMatrix | svd

More About

- "Create Diagonal Matrix Using VBA Macro" on page 1-28
- "Executing Spreadsheet Link Functions" on page 1-42

Convert Dates Between Microsoft Excel and MATLAB

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

However, MATLAB date numbers represent the number of days that have passed since January 1, 0000, so January 1, 1950 is represented as 712224 in the MATLAB software. Therefore, the difference in dates between the Excel software and the MATLAB software is a constant, 693960 (712224 minus 18264).

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

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

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

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

See Also

Related Examples
- "Create Diagonal Matrix Using Worksheet Cells" on page 1-25
- "Create Diagonal Matrix Using VBA Macro" on page 1-28

Localization Information

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

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

See Also

Related Examples

- "Setting Spreadsheet Link Preferences" on page 1-12

Executing Spreadsheet Link Functions

Spreadsheet Link functions manage the connection and data exchange between Microsoft Excel and MATLAB, without leaving the Microsoft Excel environment.

To execute Spreadsheet Link functions, you must:

- Understand the differences between these functions and Microsoft Excel functions.
- Choose the right function type, execution method, and calculation mode for your situation.
- Decide how to specify functions and arguments.

Spreadsheet Link and Microsoft Excel Function Differences

In Microsoft Excel, entering Spreadsheet Link functions can be similar to Microsoft Excel functions. The differences include:

- Spreadsheet Link functions perform an action, while Microsoft Excel functions return a value.
- Spreadsheet Link function names are case-insensitive. Entering either `MLPutMatrix` or `mlputmatrix` executes the `MLPutMatrix` function.
- MATLAB function names and variable names are case-sensitive. For example, `BONDS`, `Bonds`, and `bonds` are three different MATLAB variables.

Spreadsheet Link Function Types

There are link management and data management functions in Spreadsheet Link.

Link management functions initialize, start, and stop the Spreadsheet Link and MATLAB software. Execute the `matlabinit` function from the Excel **Tools > Macro** menu or in macro subroutines.

Data management functions copy data between Microsoft Excel and the MATLAB workspace. These functions execute MATLAB commands in Microsoft Excel. Except for `MLPutVar` and `MLGetVar`, you can execute any data management function as a worksheet cell formula or in a VBA macro. The `MLPutVar` and `MLGetVar` functions execute only in VBA macros.

Spreadsheet Link Function Execution Method

You can execute Spreadsheet Link functions using these various methods.

Execution Method	Advantages	Limitations
Microsoft Excel ribbon	Quickly access common Spreadsheet Link functionality in the MATLAB group: • `matlabinit` • `MLPutMatrix` • `MLPutRanges` • `MLGetMatrix` • `MLEvalString` • `MLGetFigure` • MATLAB Function Wizard (For details, see "Find and Execute MATLAB Function Using MATLAB Function Wizard" on page 1-30.) • Preferences (For details, see "Setting Spreadsheet Link Preferences" on page 1-12.)	Full Spreadsheet Link functionality is unavailable.

Execution Method	Advantages	Limitations
Microsoft Excel context menu	Quickly access common Spreadsheet Link functionality in a worksheet cell: • `MLPutMatrix` • `MLPutRanges` • `MLEvalString` • `MLGetFigure` • MATLAB Function Wizard (For details, see "Find and Execute MATLAB Function Using MATLAB Function Wizard" on page 1-30.) • `MLGetMatrix`	Full Spreadsheet Link functionality is unavailable.
Microsoft Excel worksheet cell	• Execute any Spreadsheet Link function. • Execute MATLAB functions.	You cannot execute `MLGetVar`, `MLPutVar`, or `matlabinit` within a worksheet cell.
Microsoft Excel VBA macro	• Execute any Spreadsheet Link function. • Execute MATLAB functions. • Execute advanced VBA code.	Requires knowledge of Microsoft Visual Basic.

Execution Method	Advantages	Limitations
MATLAB Function Wizard	• Find MATLAB function by category or folder. • Explore MATLAB function syntaxes. • Execute MATLAB function by choosing a syntax and specifying arguments. • Execute custom MATLAB function.	Execute a MATLAB function using only the Spreadsheet Link functions `matlabfcn` and `matlabsub`.

Specify Spreadsheet Link Function in Microsoft Excel

When you specify a Spreadsheet Link function in a worksheet cell, enter the formula by starting with a + or = sign. Then, enclose function arguments in parentheses. This example formula uses the `MLPutMatrix` function to export data in cell `C10` into matrix A.

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

In VBA macros, leave a space between the function name and the first argument. Do not use parentheses.

```
MLPutMatrix "A",C10
```

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

Set Calculation Mode

Spreadsheet Link functions are most effective in automatic calculation mode. To automate the recalculation of a Spreadsheet Link function, add a cell reference to a cell whose value changes. For example, the `MLPutMatrix` function executes again when the value in cell `C1` changes.

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

To use `MLPutMatrix` in manual calculation mode:

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

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

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

Specify Spreadsheet Link Function Arguments

You can specify arguments in Spreadsheet Link functions using the variable name or by referencing the data location for the argument.

Note: Spreadsheet Link functions expect the default reference style (A1) worksheet cell references. The columns must be designated with letters and the rows with numbers. If your worksheet shows columns designated with numbers instead of letters, then follow this procedure:

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

Variable-Name Arguments

- You can directly or indirectly specify a variable-name argument in most Spreadsheet Link functions.

  - To specify a variable name directly, enclose it in double quotation marks, for example, =MLDeleteMatrix("Bonds").

  - To specify a variable name as an indirect reference, enter it without quotation marks. The function evaluates the contents of the argument to retrieve the variable name. The argument must be a worksheet cell address or range name; for example, =MLDeleteMatrix(C1).

Note Spreadsheet Link functions do not support global variables. When exchanging data between Excel and MATLAB, the software uses the base workspace. Variables in the base workspace exist until you clear them or end your MATLAB session.

Data-Location Arguments

- A data-location argument must be a worksheet cell address or range name.

- Do not enclose a data-location argument in quotation marks (except in `MLGetMatrix`, which has unique argument conventions).

- A data-location argument can include a worksheet number such as `Sheet3!B1:C7` or `Sheet2!OUTPUT`.

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

Specify MATLAB Function in MATLAB Function Wizard

After you find the MATLAB function or custom function in the MATLAB Function Wizard, you can specify the syntax and arguments. Then, Spreadsheet Link specifies this command for evaluation in the MATLAB workspace.

To execute a MATLAB function with multiple outputs, specify where to write the output.

- Specifying a target range of cells using the **Optional output cell(s)** field causes the selected function to appear in the current worksheet cell as an argument of `matlabsub`. The `matlabsub` function includes an argument that indicates where to write the output. For example, the data from A2 is input to the `rand` function and the target cell for output is B2:

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

- Although the Function Wizard lets you specify multiple output cells, it does not return multiple outputs. If you specify a range of output cells, the wizard returns the first output argument starting in the first output cell. For example, if a function returns two elements a and b, and you specify A1:A2 as output cells, the Function Wizard displays a in cell A1. The Function Wizard discards element b. If an output is a matrix, the Function Wizard displays all elements of that matrix starting in the first output cell.

For multiple output arguments, see "Return Multiple Output Arguments from MATLAB Function" on page 1-37.

To execute multiple MATLAB functions or use MATLAB objects, write a wrapper function.

- The Function Wizard does not allow simultaneous execution of multiple MATLAB functions. Write a wrapper function instead. For example, to plot historical closing-price data from Bloomberg®, enter this code in MATLAB and save it as a function.

```
function plotbloombergdata(s)
% plotbloombergdata is a wrapper function that connects to
% Bloomberg(R), retrieves historical closing-price data for
% the year 2015, and plots the prices for a given
% Bloomberg(R) security s.
    c = blp;
    f = 'LAST_PRICE';
    fromdate = '01/01/2015';
    todate = '12/31/2015';
    d = history(c,s,f,fromdate,todate);
    plot(d(:,1),d(:,2))
    close(c)
end
```

For details about writing functions, see "Create Functions in Files" (MATLAB).

- Microsoft Excel has no context for MATLAB objects. To work with MATLAB objects, such as connections to service providers, write a wrapper function. The wrapper function executes the functions that create and manipulate these objects.

See Also
MLEvalString | MLGetMatrix | MLPutMatrix | matlabfcn | matlabinit

More About
- "Create Diagonal Matrix Using Microsoft Excel Ribbon" on page 1-18
- "Create Diagonal Matrix Using Microsoft Excel Context Menu" on page 1-21
- "Create Diagonal Matrix Using Worksheet Cells" on page 1-25
- "Create Diagonal Matrix Using VBA Macro" on page 1-28
- "Find and Execute MATLAB Function Using MATLAB Function Wizard" on page 1-30

- "Localization Information" on page 1-41

2

Solving Problems with the Spreadsheet Link Software

Model Data Using Regression and Curve Fitting

This example shows how to execute MATLAB data regression and curve fitting in Microsoft Excel using a worksheet and a VBA macro.

The example organizes and displays the input and output data in a Microsoft Excel worksheet. Spreadsheet Link functions copy the data to the MATLAB workspace and execute MATLAB computational and graphic functions. The VBA macro also returns output data to a worksheet.

Open the ExliSamp.xls file and select the **Sheet1** worksheet. For help finding the ExliSamp.xls file, see "Installation" on page 1-3.

Sheet1 of the spreadsheet contains the named range DATA, which consists of the example data set in worksheet cells A4 through C28.

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

Model Data in Worksheet

To perform regression and curve fitting, execute the specified Spreadsheet Link functions in worksheet cells.

1 Execute the Spreadsheet Link function that copies the sample data set to the MATLAB workspace by double-clicking the cell `E5` and pressing **Enter**. 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 Execute the functions in cells `E8`, `E9`, and `E10`. The Spreadsheet Link functions in these cells regress the third column of data on the other two columns, and create:

- A single vector `y` containing the third-column data
- A three-column matrix `A`, which consists of a column of 1s followed by the rest of the data

3 Execute the function in cell `E13`. This function calculates the regression coefficients by using the MATLAB back slash (`\`) 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:

a Compare the original data with `fit`.

b Sort the data in increasing order and apply the same permutation to `fit`.

c Create a scalar for the number of observations.

6 Execute the functions in cells `E24` and `E25`. Fit a polynomial equation to the data for a fifth-degree polynomial. The MATLAB `polyfit` function automates setting up a system of simultaneous linear equations and solutions for the coefficients. The `polyval` function then evaluates the resulting polynomial at each data point to check the goodness of the fit `newfit`.

7 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).
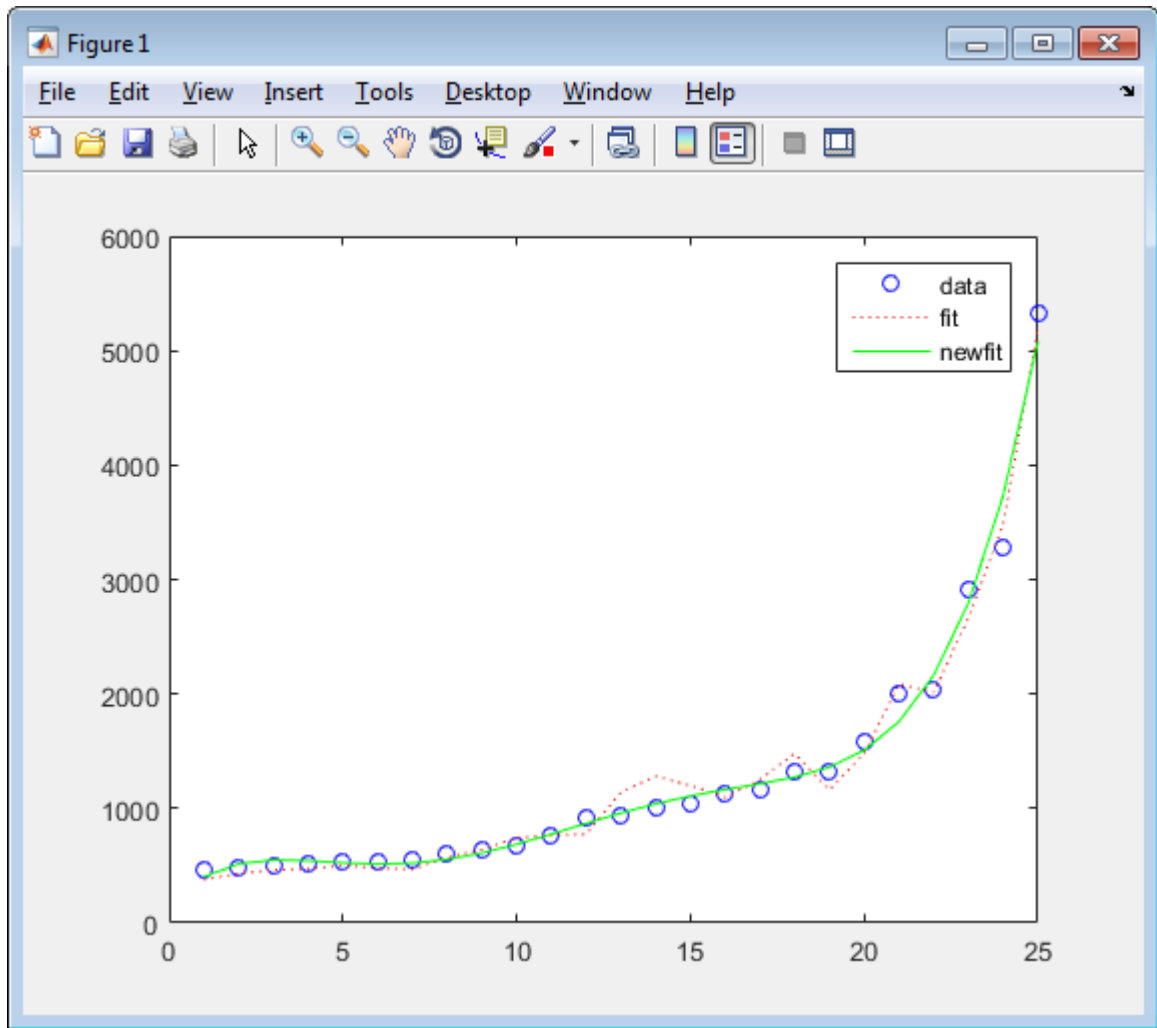
Since the data is closely correlated, but not exactly linearly dependent, the `fit` curve (dashed line) shows a close, but not exact, fit. The fifth-degree polynomial curve `newfit` is a more accurate mathematical model for the data.

Model Data Using VBA Macro

To model the data using a VBA macro, execute the Spreadsheet Link functions in a VBA macro.

1   In the `ExliSamp.xls` file, click the **Sheet2** tab. The worksheet for this example appears.

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

Cell A4 calls the macro CurveFit, which you can examine in the Microsoft Visual Basic environment.
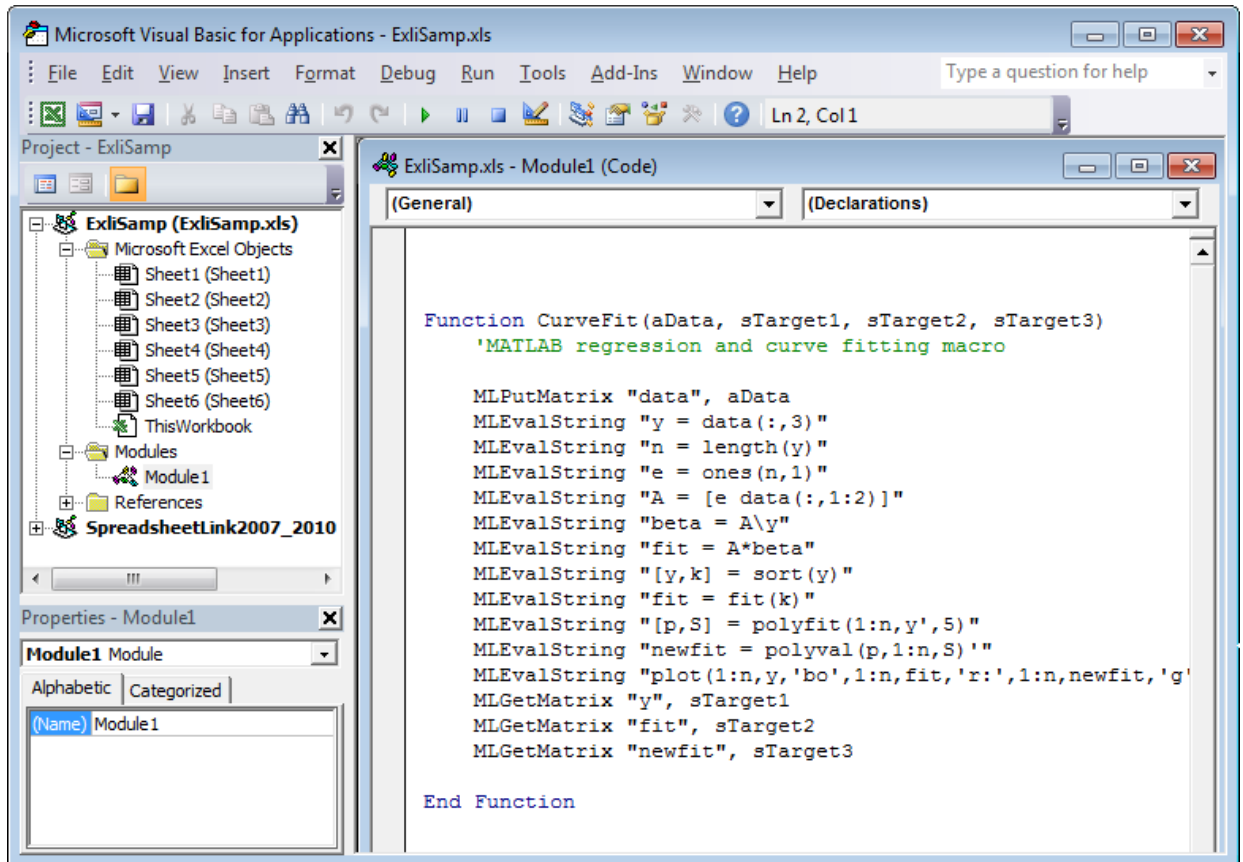
While this module is open, ensure that the Spreadsheet Link add-in is enabled. To enable it, see "Add-In Setup" on page 1-6. After the add-in is enabled, the Project Explorer lists it under the **References** folder.

2 Execute the `CurveFit` macro by double-clicking the cell `A4` and pressing **Enter**. The macro runs the Spreadsheet Link functions. When the macro finishes, the input and output data appears in worksheet cells `A7:C31`.

- Column A contains the original data `y` (sorted).
- Column B contains the corresponding regressed data `fit`.
- Column C contains the polynomial data `newfit`.

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

See Also

MLEvalString | MLGetMatrix | MLPutMatrix | plot | polyfit | polyval

More About

- "Interpolate Thermodynamic Data" on page 2-10
- "Price Stock Options Using Binomial Model" on page 2-15
- "Plot Efficient Frontier of Financial Portfolios" on page 2-19
- "Map Time and Bond Cash Flows" on page 2-23
- "Executing Spreadsheet Link Functions" on page 1-42

Interpolate Thermodynamic Data

This example shows how to interpolate data using Spreadsheet Link to invoke MATLAB functions in Microsoft Excel.

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

To organize and display the original data and the interpolated output data, you can use Microsoft Excel worksheets.

Open the `ExliSamp.xls` file and select the **Sheet3** worksheet. For help finding the `ExliSamp.xls` file, see "Installation" on page 1-3.

This worksheet contains measured thermodynamic data in cell ranges `A5` through `A29`, `B5` through `B29`, and `C5` through `C29`. The time and temperature values for interpolation are in cell ranges `E7` through `E30` and `F6` through `T6`, respectively.

Data Interpolation

Original Data

Time	Temp	Volume
0.025	68.00	2504.08
0.050	68.05	2535.07
0.075	68.07	2562.91
0.100	68.09	2575.74
0.125	68.20	2606.16
0.150	68.50	2628.58
0.175	68.85	2681.38
0.200	69.22	2712.06
0.225	70.08	2767.52
0.250	70.33	2815.54
0.275	70.59	2824.37
0.300	70.85	2873.65
0.325	71.11	2882.20
0.350	71.44	2896.49
0.375	71.82	2902.07
0.400	72.33	2920.04
0.425	72.65	2929.35
0.450	73.46	2934.23
0.475	73.85	2938.55
0.500	74.22	3012.93
0.525	74.37	3099.12
0.550	74.55	3130.01
0.575	74.67	3179.24
0.600	74.72	3180.71
0.625	75.00	3184.15

Interpolated Values

Temp

Time	68.0	68.5	69.0	69.5	70.0	70.5	71.0	71.5	72.0	72.5	73.0	73.5	74.0	74.5	75.0
0.025															
0.05															
0.075															
0.1															
0.125															
0.15															
0.175															
0.2															
0.225															
0.25															
0.275															
0.3															
0.325															
0.35															
0.375															
0.4															
0.425															
0.45															
0.475															
0.5															
0.525															
0.55															
0.575															
0.6															

Spreadsheet Link Functions

1. Transfer original data to MATLAB.
#MATLAE <== MLPutMatrix("Labels", A4:C4)
#MATLAE <== MLPutMatrix("X",A5:A29)
#MATLAE <== MLPutMatrix("T",B5:B29)
#MATLAE <== MLPutMatrix("V",C5:C29)

2. Transfer interpolation data points to MATLAB.
#MATLAE <== MLPutMatrix("Xa",E7:E30)
#MATLAE <== MLPutMatrix("Ta",F6:T6)

3. Execute MATLAB data interpolation function.
#MATLAE <== MLEvalString("[XI, TI, VI] = griddata(X,T,V,Xa,Ta, 'nearest')")

4. Transpose output data matrix and transfer data to Excel.
#MATLAE <== MLEvalString("VI = VI';")
#MATLAE <== MLGetMatrix("VI","sheet3!F7")

5. Plot interpolated data and label the figure.
#MATLAE <== MLEvalString("surf(XI, TI, VI);title('Interpolated Data');xlabel(Labels{1});ylabel(Labels{2});zlabel(Labels{3});grid on")

1 Execute the Spreadsheet Link function that passes the `Time`, `Temp`, and `Volume` labels to the MATLAB workspace by double-clicking the cell `A33` and pressing **Enter**.

2 Copy the original time data to the MATLAB workspace by executing the function in the cell `A34`. To copy the original temperature data, execute the function in the cell `A35`. To copy the original volume data, execute the function in cell `A36`.

3 Copy the interpolation time values to the MATLAB workspace by executing the function in cell `A39`. To copy the interpolation temperature values, execute the function in cell `A40`.

4 Execute the function in cell `A43`. The `griddata` function performs two-dimensional interpolation that generates the interpolated volume data using the inverse distance method.

2-11

5 Transpose the interpolated volume data and copy it to the Excel worksheet by executing the functions in cells A46 and A47. The data fills the cell range F7:T30.

Interpolated Values

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

6 Execute the function in cell A50. The MATLAB software plots and labels the interpolated data on a three-dimensional color surface, with the color proportional to the interpolated volume data.
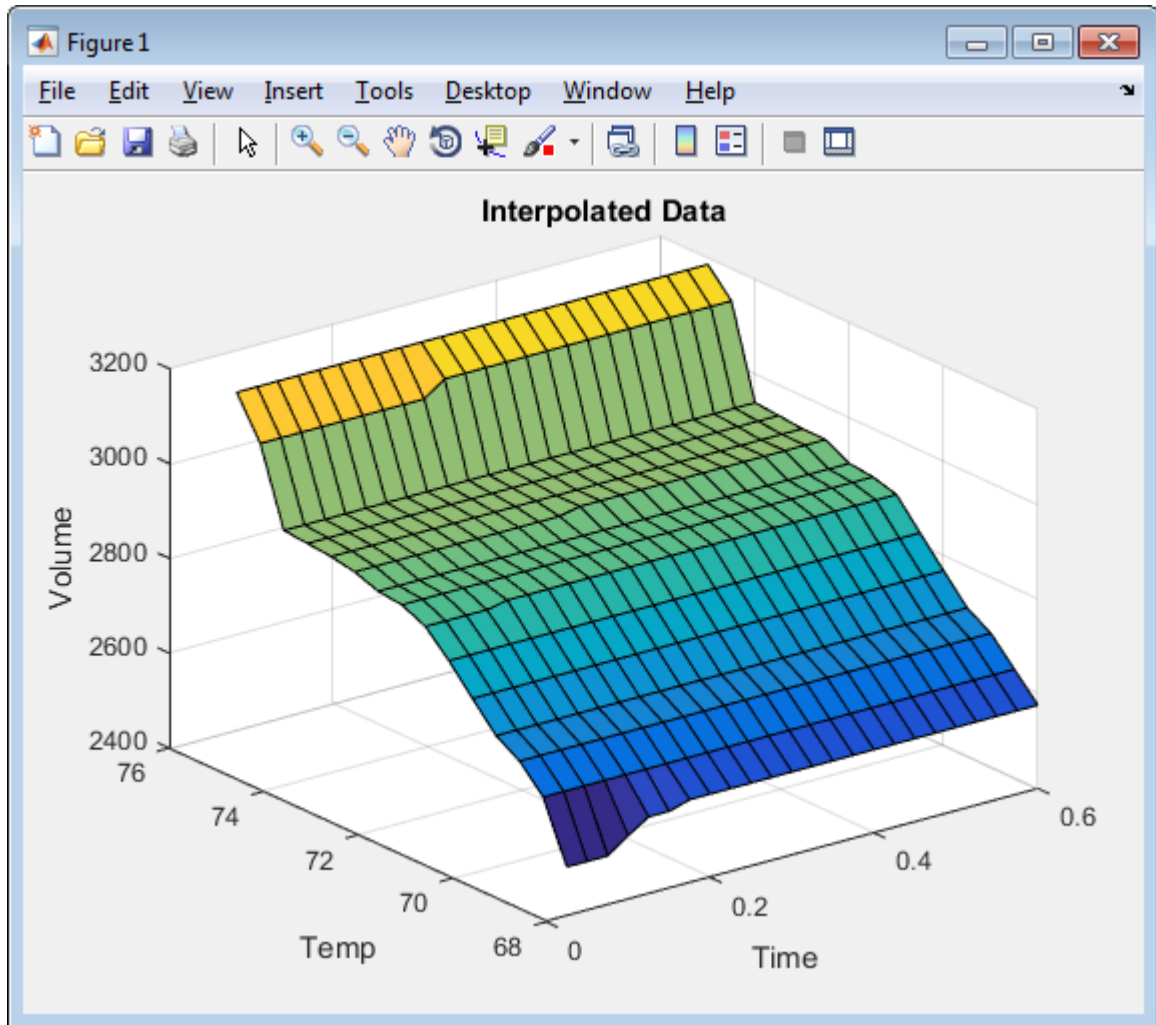
To generate different volume values, close the figure and change the measured thermodynamic data in cells A5:C29. Then, execute all the Spreadsheet Link functions again. The worksheet updates with new volume rates and MATLAB generates a new figure of the interpolated volume data.

See Also

MLEvalString | MLGetMatrix | MLPutMatrix | griddata | surf

More About

- "Model Data Using Regression and Curve Fitting" on page 2-2
- "Price Stock Options Using Binomial Model" on page 2-15
- "Plot Efficient Frontier of Financial Portfolios" on page 2-19
- "Map Time and Bond Cash Flows" on page 2-23
- "Executing Spreadsheet Link Functions" on page 1-42

Price Stock Options Using Binomial Model

This example uses the binomial model to price a stock option. The binomial model assumes that the probability of each possible price over time follows a binomial distribution. Price values can become either one up or one down over any short time period. Plotting these two values over time is known as building a binomial tree. For details about the binomial model, see "Pricing and Analyzing Equity Derivatives" (Financial Toolbox).

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

Open the `ExliSamp.xls` file and select the **Sheet4** worksheet. For help finding the `ExliSamp.xls` file, see "Installation" on page 1-3.

This worksheet contains these named ranges:

- `B4:B10` named `bindata`. Two cells in `bindata` contain formulas:

    - B7 contains =5/12
    - B8 contains =1/12
- `B15` named `asset_tree`.
- `B23` named `value_tree`.

	A	B	C	D	E	F	G	H	I	J	K
1	Binomial Option Pricing										
2											
3		bindata			Spreadsheet Link Functions						
4	Asset price, so	$ 52.00			1. Transfer data to MATLAB.						
5	Option exercise price, x	$ 50.00			#MATLAB <== MLPutMatrix("b", bindata)						
6	Risk-free interest rate, r	10%									
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		#MATLAB <== 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					#MATLAB <== MLGetMatrix("p", "asset_tree")						
12					#MATLAB <== MLGetMatrix("o", "value_tree")						
13											
14			Start	Period 1	Period 2	Period 3	Period 4	Period 5			
15	Asset price tree, p ($)										
16											
17											
18											
19											
20											
21											
22											
23	Option value tree, o ($)										
24											
25											
26											
27											
28											

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

1 Execute the Spreadsheet Link function that copies the asset data to the MATLAB workspace by double-clicking the cell D5 and pressing **Enter**.

2 Execute the function that calculates the binomial prices in cell D8.

3 Copy the price data to the worksheet by executing the functions in cells D11 and D12.

The data in the worksheet updates.

	A	B	C	D	E	F	G	H	I	J	K
1	Binomial Option Pricing										
2											
3		bindata		Spreadsheet 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				

The asset price tree contains these prices:

- Period 1 — The up and down prices
- Period 2 — The up-up, up-down, and down-down prices
- Period 3 — The up-up-up, up-up, down-down, and down-down-down prices
- And so on.

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

You can generate different binomial prices by changing the data in the cell range B4:B10 and executing the Spreadsheet Link functions again. If you increase the time to maturity

in cell B7 or change the time increment in cell B8, enlarge the output tree areas as needed.

See Also

MLEvalString | MLGetMatrix | MLPutMatrix | binprice

More About

- "Model Data Using Regression and Curve Fitting" on page 2-2
- "Interpolate Thermodynamic Data" on page 2-10
- "Plot Efficient Frontier of Financial Portfolios" on page 2-19
- "Map Time and Bond Cash Flows" on page 2-23
- "Pricing and Analyzing Equity Derivatives" (Financial Toolbox)
- "Executing Spreadsheet Link Functions" on page 1-42

Plot Efficient Frontier of Financial Portfolios

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

For details about the efficient frontier of financial portfolios, see "Analyzing Portfolios" (Financial Toolbox). To learn about portfolio optimization theory, see "Portfolio Optimization Theory" (Financial Toolbox).

The example organizes and displays the input and output data in a Microsoft Excel worksheet. Spreadsheet Link functions copy data to a MATLAB matrix, perform calculations using Financial Toolbox functions, and return data to the worksheet.

Open the `ExliSamp.xls` file and select the **Sheet5** worksheet. For help finding the `ExliSamp.xls` file, see "Installation" on page 1-3.

This worksheet contains rates of return for three different portfolios: Global, Corporate Bond, and Small Cap.

	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%						
5	Nov-92	5.125%	5.125%	3.875%						
6	Nov-93	-1.375%	5.750%	10.500%						
7	Nov-94	7.750%	6.000%	14.750%						
8	Nov-95	8.250%	6.375%	-3.625%						
9	Nov-96	12.625%	6.125%	9.125%						
10										
11										
12										
13	Spreadsheet Link Functions									
14	1. Transfer data to MATLAB.									
15	#MATLAB?	<== MLPutMatrix("Labels", F3:G3)								
16	#MATLAB?	<== MLPutMatrix("retseries", B4:D9)								
17										
18	2. Execute MATLAB Financial Toolbox functions.									
19	#MATLAB?	<== MLEvalString("[ret, cov] = ewstats(retseries)")								
20	#MATLAB?	<== MLEvalString("[risk, ror, weights] = portopt(ret, cov, 20)")								
21										
22	3. Transfer output data to Excel.									
23	#MATLAB?	<== MLGetMatrix("risk", "sheet5!F4")								
24	#MATLAB?	<== MLGetMatrix("ror", "sheet5!G4")								
25	#MATLAB?	<== MLGetMatrix("weights", "sheet5!H4")								
26										
27	4. Plot efficient frontier data and label the figure.									
28	#MATLAB?	<== MLEvalString("portopt(ret, cov, 20); grid on; xlabel(Labels{1}); ylabel(Labels{2})")								

> **Note** This example requires Financial Toolbox, Statistics and Machine Learning Toolbox, and Optimization Toolbox.

1 Execute the Spreadsheet Link function that transfers the plot labels for the *x*-axis and *y*-axis to the MATLAB workspace by double-clicking the cell A15 and pressing **Enter**.

2 Copy the portfolio return data to the MATLAB workspace by executing the function in the cell A16.

3 Generate efficient frontier data for 20 points along the frontier by executing the Financial Toolbox functions in A19 and A20.

4 Copy the output data to the Excel worksheet by executing the Spreadsheet Link functions in A23, A24, and A25.

The output data contains the highest rate of return ROR for a given risk. The output data also contains the weighted investment in each portfolio Weights that achieves that rate of return.

	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	**Spreadsheet 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", "sheet5!F4")			5.791%	7.167%	0.0%	0.0%	100.0%
24		0	<== MLGetMatrix("ror", "sheet5!G4")							
25		0	<== MLGetMatrix("weights", "sheet5!H4")							
26										
27	4. Plot efficient frontier data and label the figure.									
28	#MATLAB?		<== MLEvalString("portopt(ret, cov, 20); grid on; xlabel(Labels{1}); ylabel(Labels{2})")							

5 Plot the efficient frontier for the same portfolio data by executing the Financial Toolbox functions in cell A28.
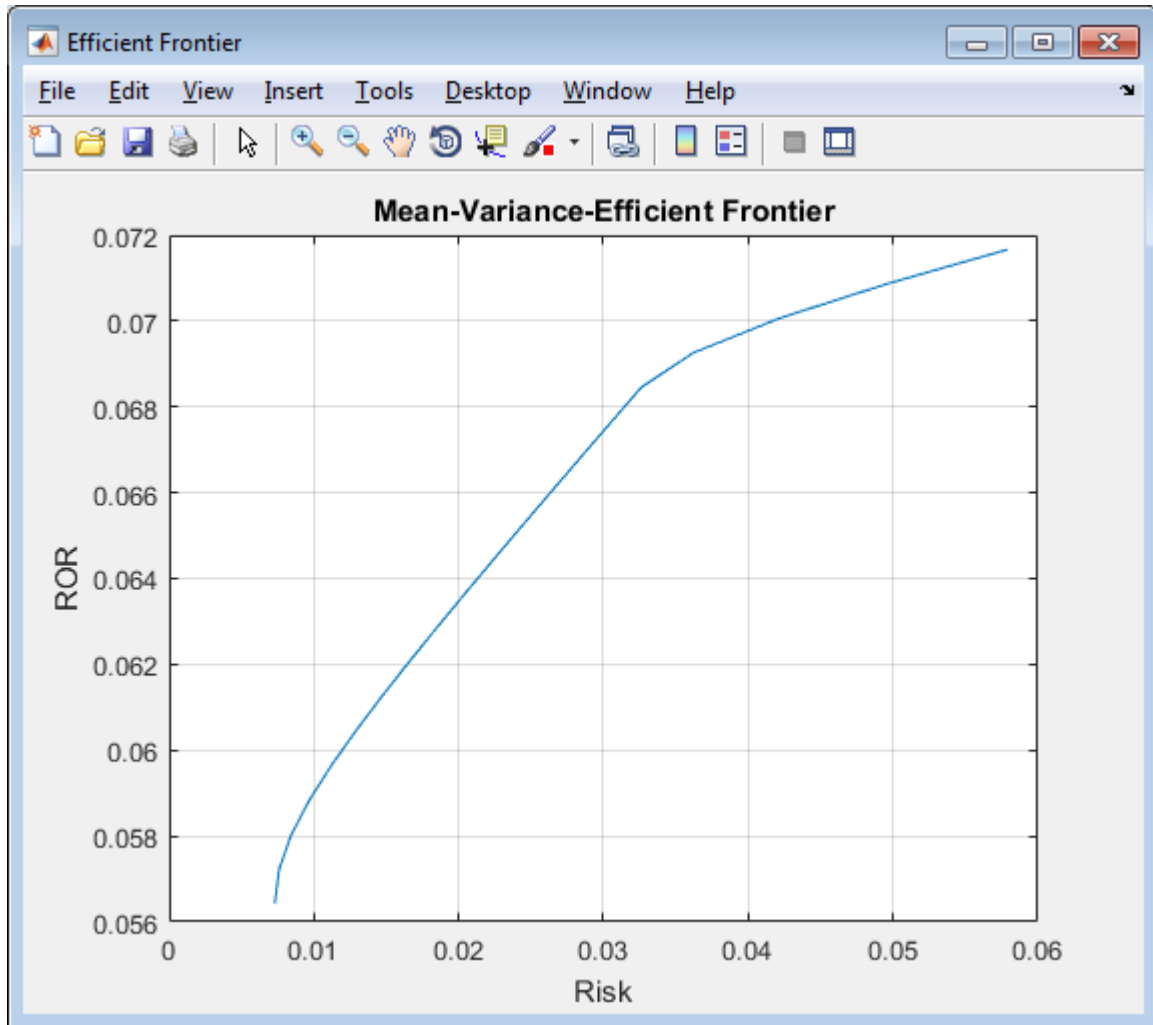
The light blue line shows the efficient frontier. Observe the change in slope above a 6.8% return because the Corporate Bond portfolio no longer contributes to the efficient frontier.

To generate different efficient frontier data, close the figure and change the data in cells B4:D9. Then, execute all the Spreadsheet Link functions again. The worksheet updates with new frontier data and MATLAB generates a new efficient frontier plot.

See Also

MLEvalString | MLGetMatrix | MLPutMatrix | ewstats | portopt

More About

Map Time and Bond Cash Flows

This example shows how to use Financial Toolbox and Spreadsheet Link to calculate a set of cash flow amounts and dates for a portfolio of five bonds.

Open the `ExliSamp.xls` file and select the **Sheet6** worksheet. For help finding the `ExliSamp.xls` file, see "Installation" on page 1-3.

This worksheet contains the maturity dates and coupon rates for five bonds.

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

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

1 Execute the Spreadsheet Link function that transfers the column vector `Maturity` to the MATLAB workspace by double-clicking the cell `A18` and pressing **Enter**.

2 Transfer the column vector `Coupon Rate` to the MATLAB workspace by executing the function in cell `A19`.

3 Transfer the settlement date to the MATLAB workspace by executing the function in cell `A20`.

4 Calculate cash flow amounts and dates by executing the Financial Toolbox functions in cells `A23` and `A24`.

5 Transform the dates into a cell array of character vectors by executing the functions in cells `A27` through `A29`.

6 Transfer the data to the Excel worksheet by executing the functions in cells `A32` through `A34`.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Cash Flow and Time Mapping for a Portfolio of Bonds													
2											Cash Flow Dates			
3	Settlement Date		26-Jul-99					Bond1	7/26/1999	11/15/1999	N/A	N/A	N/A	N/A
4								Bond2	7/26/1999	11/15/1999	5/15/2000	N/A	N/A	N/A
5			Bond Data					Bond3	7/26/1999	11/15/1999	5/15/2000	11/15/2000	N/A	N/A
6								Bond4	7/26/1999	11/15/1999	5/15/2000	11/15/2000	5/15/2001	N/A
7		Maturity	Coupon Rate					Bond5	7/26/1999	11/15/1999	5/15/2000	11/15/2000	5/15/2001	11/15/2001
8	Bond1	15-Nov-99	0.05875											
9	Bond2	15-May-00	0.06375											
10	Bond3	15-Nov-00	0.08500											
11	Bond4	15-May-01	0.08000											
12	Bond5	15-Nov-01	0.15750								Cash Flow Amounts			
13									7/26/1999	11/15/1999	5/15/2000	11/15/2000	5/15/2001	11/15/2001
14								Bond1	-1.1495	102.9375	0	0	0	0
15								Bond2	-1.2473	3.1875	103.1875	0	0	0
16	Spreadsheet Link Functions							Bond3	-1.6630	4.2500	4.2500	104.2500	0	0
17	1. Transfer data to MATLAB.							Bond4	-1.5652	4.0000	4.0000	4.0000	104.0000	0
18	0 <== MLPutMatrix("maturity",'Maturity')							Bond5	-3.0815	7.8750	7.8750	7.8750	7.8750	107.8750
19	0 <== MLPutMatrix("cpnrate","CpnRate")													
20	0 <== MLPutMatrix("sd",C3)													
21														
22	2. Execute MATLAB Financial Toolbox Cash flow and Time mapping function.													
23	0 <== MLEvalString("md = x2mdate(maturity,0); sdm = x2mdate(sd,0)")													
24	0 <== MLEvalString("[cfa, cfd] = cfamounts(cpnrate, sdm, md, 2)")													
25														
26	3. Transform date numbers to cell array of character vectors.													
27	0 <== MLEvalString("i = find(isnan(cfd)); zcfd = cfd; zcfd(i) = 0; scfd=datestr(zcfd,2);")													
28	0 <== MLEvalstring("ccfd = num2cell(scfd,2); ccfd(i) = {'N/A'}; ccfd = reshape(ccfd, size(cfd));")													
29	0 <== MLEvalString("ccfa = cfa; ccfa(i) = 0; alldates = ccfd(end, :);")													
30														
31	4. Transfer output data to Excel.													
32	0 <== MLGetMatrix("ccfd", "sheet6li3")													
33	0 <== MLGetMatrix("alldates", "sheet6li13")													
34	0 <== MLGetMatrix("ccfa", "sheet6li14")													
35														
36	5. Plot the cash flow diagram.													
37	0 <== MLEvalString("cfplot(cfd, cfa); dtaxis('x',6,sdm,50);title('Cash Flow Diagram');xlabel('Cash Flow Dates');ylabel('Bonds');")													

7 Display a plot of cash flows for each bond by executing the function in cell A37.
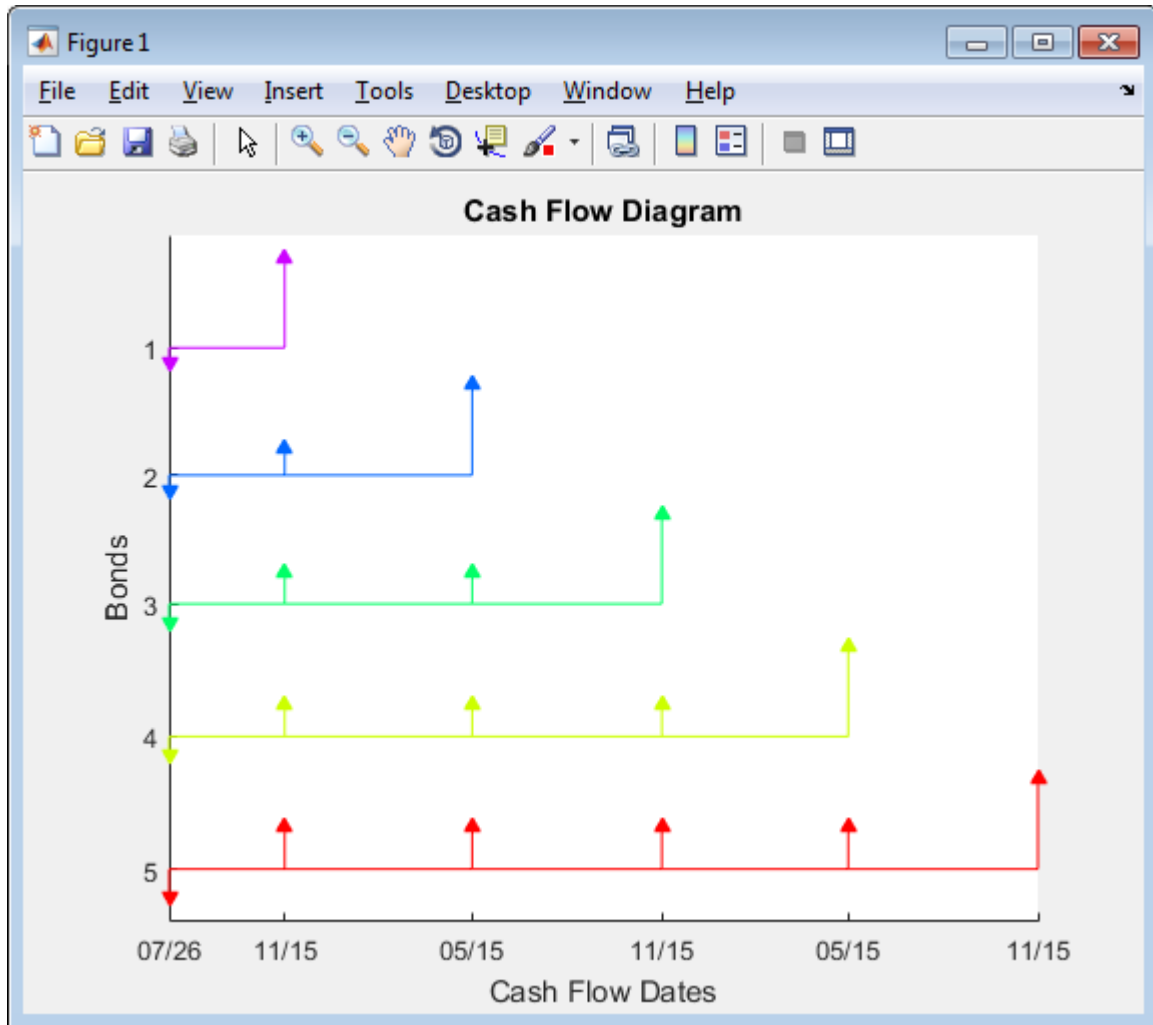
To generate cash flows for a different set of five bonds, close the figure and change the bond data in cells B8:C12. Then, execute all the Spreadsheet Link functions again. The worksheet updates with new cash flow dates and amounts and MATLAB generates a new figure of the cash flows.

See Also

MLEvalString | MLGetMatrix | MLPutMatrix | cfamounts | cfplot | x2mdate

More About

- "Model Data Using Regression and Curve Fitting" on page 2-2
- "Interpolate Thermodynamic Data" on page 2-10
- "Price Stock Options Using Binomial Model" on page 2-15
- "Plot Efficient Frontier of Financial Portfolios" on page 2-19
- "Executing Spreadsheet Link Functions" on page 1-42

3

Error Messages and Troubleshooting

Worksheet Cell Errors

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

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

Worksheet Cell Error Messages

Error Message	Meaning	Solution
`#COLS>#MAXCOLS!`	Your MATLAB variable exceeds the Microsoft Excel limit of `#MAXCOLS!` columns.	This is a limitation in the Excel product. Try the computation with a variable containing fewer columns.
`#COMMAND!`	This error message represents any error that appears at the MATLAB command line. You can show the full error message from the MATLAB command line by using the `MLShowMatlabErrors` function. Or, you can set the corresponding preference in the Preferences dialog box available from the MATLAB group on the Excel Home tab.	Troubleshoot the MATLAB error.
`#DIMENSION!`	You used `MLAppendMatrix` and the dimensions of the appended data do not match the dimensions of the matrix you want to append.	Verify the matrix dimensions and the appended data dimensions, and correct the argument. For more information, see the `MLAppendMatrix` reference page.
`#INVALIDNAME!`	You entered an invalid variable name.	Ensure that you use valid MATLAB variable names. MATLAB variable names must start with a letter, followed by letters, digits, or underscores. For details, see "Variable Names" (MATLAB).
`#INVALIDTYPE!`	You specified an illegal MATLAB data type with `MLGetVar` or `MLGetMatrix`.	Make sure to use the supported MATLAB data types.
`#MATLAB?`	You used a Spreadsheet Link function and no MATLAB software session is running.	Start the Spreadsheet Link and MATLAB software. See "Start and Stop Spreadsheet Link and MATLAB" on page 1-15.

Error Message	Meaning	Solution
#NAME?	The function name is unrecognized. The `excllink.xla` add-in is not loaded, or the function name might be misspelled.	Be sure the `excllink.xla` add-in is loaded. See "Add-In Setup" on page 1-6. 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 might be misspelled. Also, you receive the #NONEXIST! error when you attempt to use `matlabfcn` to obtain an output.	Verify the spelling of the MATLAB matrix. Use the MATLAB `whos` command to display existing matrices. Correct typing errors.
#ROWS>#MAXROWS!	Your MATLAB variable exceeds the Excel limit of #MAXROWS! rows.	This is a limitation in the Excel product. Try the computation with a variable containing fewer rows.
#SYNTAX?	You entered a Spreadsheet Link function with incorrect syntax. For example, you did not specify double quotation marks ("), or you specified single quotation marks (') instead of double quotation marks.	Verify and correct the function syntax.
#VALUE!	An argument is missing from a function, or a function argument is the wrong type.	Supply the correct number of function arguments, of the correct type.
#VALUE!	A macro subroutine uses `MLGetMatrix` followed by `MatlabRequest`, which is correct standard usage. A macro function calls that subroutine, and you execute that function from a worksheet cell. The function works correctly, but this message appears in the cell.	Since the function works correctly, ignore the message. Or, in this special case, remove `MatlabRequest` from the subroutine.

Error Message	Meaning	Solution
#INVALIDRANGE!	The named range is defined incorrectly, or the named range spans multiple worksheets.	Select a range of data on only one worksheet and create an appropriate name for the range of data. For instructions about defining names, see Excel Help.

Note When you open an Excel worksheet that contains Spreadsheet Link functions, the Excel software tries to execute the functions from the bottom up and right to left. Excel might generate cell error messages such as #COMMAND! or #NONEXIST!. This is expected behavior, so ignore the messages and do the following:
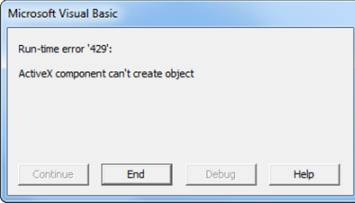
1 Close the MATLAB figure windows.

2 Execute the cell functions again one at a time in the correct order by pressing **F2**, and then **Enter**.

Microsoft Excel Errors

The Excel software can display these error messages.

Error Message	Cause of Error	Solution
`Error in formula`	You entered a formula incorrectly. Common errors include a space between the function name and the left parenthesis; or missing, extra, or mismatched parentheses. **Note** If you use the Spreadsheet Link software with a non-English (`United States`) Windows desktop environment, certain syntactical elements might not work. For details, see "Localization Information" on page 1-41.	Review the entry and correct typing errors.

`Can't find project or library`
 	
or
 	
`Compile error: Sub or Function not defined`
	You executed a macro and the location of `excllink.xla` is incorrect or not specified.	Click **OK**. The References window opens. Remove the check mark from `MISSING: excllink.xla`. Find `excllink.xla` in its correct location, select its check box in the References window, and click **OK**. Or, select **Tools** > **References** to open the References window. Select the box named `SpreadsheetLink2007_2 010`. Click **OK**.

Error Message	Cause of Error	Solution
`Run-time error '1004': Cells method of Application class failed`	You used `MLGetMatrix` and the matrix is larger than the space available in the worksheet. This error destabilizes the Spreadsheet Link software session and changes worksheet calculation mode to manual.	Click **OK**. Reset worksheet calculation mode to `automatic`, and save your worksheet as needed. Restart the Excel, Spreadsheet Link, and MATLAB software sessions.
`MATLAB failed to check out a license of Spreadsheet Link or does not have a valid installation of Spreadsheet Link`	You entered an invalid license pass code or did not install Spreadsheet Link properly.	Ensure that you entered the license pass code properly. Reinstall the Spreadsheet Link add-on. (See "Installation" on page 1-3.) If you followed the installation guidelines, used a proper pass code, and you are still unable to start the Spreadsheet Link software, contact your MathWorks® representative.
`Datasource: Excel; prompt for user name and password`	This message appears when an attempt to connect to the Excel software from the Database Toolbox™ software fails.	Ensure that the Excel worksheet referenced by the data source exists, then retry the connection.
`Could not load some objects because they are not available on this machine`	This message appears when Excel 2013 is not configured properly.	From the Windows **Control Panel**, remove Microsoft Office 2010 in the programs list.

Error Message	Cause of Error	Solution

Microsoft Visual Basic
 	
Run-time error '429':
 	
ActiveX component can't create object
 	
Continue · **End** · Debug · Help
	This error appears when you start the automation server from the Excel interface, and multiple versions of the MATLAB software are installed on your desktop.	To correct this error, ensure that you have administrator privileges on your machine and then perform the following:
		
		1 Shut down all MATLAB and Excel instances.
		
		2 Open a command prompt, and using `cd`, change to the `bin \win64` subfolder of the MATLAB installation folder.
		
		3 Type the command:
		
		`.\matlab /regserver`
		
		4 When the MATLAB session starts, close it. Using `/regserver` fixes the registry entries.
		
		5 Start an Excel session. The Spreadsheet Link add-in now loads properly.
		
		6 Verify that the Spreadsheet Link software is working by entering the following command from the Command Window:
		
		`a = 3.14159`
		
		7 Enter the following formula in cell `A1` of the open Excel worksheet:

Error Message	Cause of Error	Solution
		`=mlgetmatrix("a","a1")`
		8 The value 3.14159 appears in cell A1.

Data Errors

In this section...
"Matrix Data Errors" on page 3-10
"Errors When Opening Saved Worksheets" on page 3-10

Matrix Data Errors

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

Data Errors

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

Errors When Opening Saved Worksheets

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

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

  1 Ignore the messages.

2 Close MATLAB figure windows.

3 Execute the cell functions again one at a time in the correct order by pressing **F2**, and then **Enter**.

- If you save an Excel worksheet containing Spreadsheet Link functions, and then reopen it in an environment where the `excllink.xla` add-in is in a different location, you may see the message: `This document contains links: Re-establish links?`

  To address this issue, do the following:

  1 Click **No**.

  2 Select **Edit > Links**.

  3 In the Links dialog box, click **Change Source**.

  4 In the Change Links dialog box, select *matlabroot*`\toolbox\exlink \excllink.xla`.

  5 Click **OK**.

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

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

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

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

License Errors

If you are running an automation server of MATLAB that does not have a Spreadsheet Link license associated with it, you will receive an license error message. To correct this issue, from the MATLAB installation that includes Spreadsheet Link, run the command:

```
matlab /regserver
```

Startup Errors

In this section...
"MATLAB Automatic Start Error" on page 3-13
"MATLAB Version Errors" on page 3-13

MATLAB Automatic Start Error

If you have enabled `MLAutoStart` and `MLUseFullDesktop`, right-clicking a spreadsheet file in the MATLAB Current Folder browser and choosing **Open Outside MATLAB** causes a MATLAB System Error to appear. To open the file successfully, click **End Now** in the error window.

To avoid this issue, disable `MLUseFullDesktop`.

MATLAB Version Errors

If the MATLAB version is set incorrectly, MATLAB does not start and displays this error: `Unable to start MATLAB. Please register MATLAB Software as a COM Server`. Set the MATLAB version using `MLProgramId`.

Ensure that the correct MATLAB version appears in the Preferences dialog box before starting MATLAB. For details, see "Setting Spreadsheet Link Preferences" on page 1-12.

If MATLAB is installed on your computer and setting the MATLAB version does not work, use the last registered version to start MATLAB. To specify the last registered version of MATLAB:

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

   `.\matlab /regserver`

See Also

`MLAutoStart` | `MLProgramId`

More About

- "Installation" on page 1-3
- "Add-In Setup" on page 1-6
- "Setting Spreadsheet Link Preferences" on page 1-12

Audible Error Signals

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

Functions — Alphabetical List

matlabfcn

Evaluate MATLAB command given Microsoft Excel data

Syntax

```
= matlabfcn(command,inputs)
```

Description

`= matlabfcn(command,inputs)` specifies the MATLAB command for evaluation in the MATLAB workspace, given the input data `inputs`. Depending upon the MATLAB output, `matlabfcn` returns a single value or string into the calling worksheet cell. If the result contains more than one value in MATLAB, only the first value in the array returns in the calling worksheet cell. Use this syntax when working in a worksheet cell.

Examples

Find Maximum Likelihood Estimate

Using `std`, find the maximum likelihood estimate of the standard deviation of data in worksheet cells `A1` through `A10`. Assume that the data comes from a normal population. Specify a weight of one in cell `C1`. Return the result by entering this text into cell `A12`.

```
=matlabfcn("std",A1:A10,C1)
```

Cell A12 displays the maximum likelihood estimate of the standard deviation.

	A	B	C
1	11.3442		1
2	14.5847		
3	4.3529		
4	12.1554		
5	10.7969		
6	6.7308		
7	8.916		
8	10.8566		
9	18.946		
10	16.9236		
11			
12	4.197647		
13			

Input Arguments

command — MATLAB command to evaluate
string

MATLAB command to evaluate, specified as a string. Enclose the string in double quotes. Or, enter the string in a cell without quotes and enter the corresponding cell reference without quotes as the input argument.

Example: `"sum"`

Example: `A1`

inputs — MATLAB command input arguments
Excel cell reference

MATLAB command input arguments, specified as an Excel cell reference or a reference to a range of cells. To specify multiple input arguments for a function, separate the cell references with commas.

Example: `B1:B10`

Example: `A1:C1,A3`

Tip

- If `matlabfcn` fails, a standard Spreadsheet Link error displays by default; for example, `#COMMAND`. To return MATLAB errors, use `MLShowMatlabErrors`.

See Also

`MLShowMatlabErrors` | `matlabsub` | `std`

Topics

"Create Diagonal Matrix Using Worksheet Cells" on page 1-25
"Executing Spreadsheet Link Functions" on page 1-42
"Worksheet Cell Errors" on page 3-2
"Microsoft Excel Errors" on page 3-6

Introduced before R2006a

matlabinit

Initialize Spreadsheet Link and start MATLAB

Syntax

```
matlabinit
```

Description

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

Tips

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

See Also

`MLAutoStart` | `MLOpen`

Topics

"Create Diagonal Matrix Using Microsoft Excel Ribbon" on page 1-18
"Create Diagonal Matrix Using Microsoft Excel Context Menu" on page 1-21
"Create Diagonal Matrix Using Worksheet Cells" on page 1-25
"Create Diagonal Matrix Using VBA Macro" on page 1-28
"Find and Execute MATLAB Function Using MATLAB Function Wizard" on page 1-30
"Executing Spreadsheet Link Functions" on page 1-42

Introduced before R2006a

matlabsub

Evaluate MATLAB command given Microsoft Excel data and designate output location

Syntax

```
= matlabsub(command,result,inputs)
```

Description

`= matlabsub(command,result,inputs)` specifies the MATLAB command for evaluation in the MATLAB workspace, given the input data `inputs`. The function returns the MATLAB output into the worksheet cell specified by `result`. Use this syntax when working in a worksheet cell.
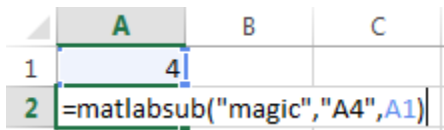
Examples

Return Fourth-Order Magic Square in Worksheet Cell

Enter the number four in cell `A1`.

Enter this text into cell `A2`. Specify the function `magic` as the command. Return the fourth-order magic square into the range of cells starting in cell `A4`. Reference cell `A1` as the input argument for the `magic` function.

```
=matlabsub("magic","A4",A1)
```


The fourth-order magic square displays in the range of cells from `A4` through `D7`.

	A	B	C	D
1	4			
2	0			
3				
4	16	2	3	13
5	5	11	10	8
6	9	7	6	12
7	4	14	15	1

Return Fourth-Order Magic Square in Target Worksheet Cell

Enter the number four in cell A1. Enter a target cell reference to cell A6 in cell A2.

Enter this text into cell A4. Specify the function `magic` as the command. Return the fourth-order magic square into the range of cells starting in cell A6 by entering the cell reference A2. Reference cell A1 as the input argument for the `magic` function.

```
=matlabsub("magic",A2,A1)
```

	A	B	C
1	4		
2	A6		
3			
4	=matlabsub("magic",A2,A1)		

The fourth-order magic square displays in the range of cells from A6 through D9.

	A	B	C	D
1	4			
2	A6			
3				
4	0			
5				
6	16	2	3	13
7	5	11	10	8
8	9	7	6	12
9	4	14	15	1

Input Arguments

command — MATLAB command to evaluate
string

MATLAB command to evaluate, specified as a string. Enclose the string in double quotes. Or, enter the string in a cell without quotes and enter the corresponding cell reference without quotes as the input argument.

Example: "sum"

Example: A1

result — MATLAB result
Excel cell reference

MATLAB result, specified as an Excel cell reference or range name that designates the location where to display the result of the MATLAB command. To return the result in a specific cell, specify the cell reference enclosed in quotes. To denote a worksheet cell address or range name that contains a cell reference to another cell, specify the cell reference without quotes.

Example: "A3"

Example: D6

inputs — MATLAB command input arguments
Excel cell reference

MATLAB command input arguments, specified as an Excel cell reference or a reference to a range of cells. To specify multiple input arguments for a function, separate the cell references with commas.

Example: `B1:B10`

Example: `A1:C1,A3`

Tips

- To return an array of data to the Microsoft Excel Visual Basic for Applications (VBA) workspace, see `MLEvalString` and `MLGetVar`.
- `result` must not include the cell that contains `matlabsub`. Do not overwrite the function itself.
- Ensure that there is enough room in the worksheet for writing matrix contents. If there is insufficient room, the function generates a fatal error.
- If `matlabsub` fails, a standard Spreadsheet Link error displays by default; for example, `#COMMAND`. To return MATLAB errors, use `MLShowMatlabErrors`.

See Also

`MLShowMatlabErrors` | `magic` | `matlabfcn`

Topics

"Create Diagonal Matrix Using Worksheet Cells" on page 1-25
"Executing Spreadsheet Link Functions" on page 1-42
"Worksheet Cell Errors" on page 3-2
"Microsoft Excel Errors" on page 3-6

Introduced before R2006a

MLAppendMatrix

Create or append MATLAB matrix with data from Microsoft Excel worksheet

Syntax

```
= MLAppendMatrix(var_name,mdat)
MLAppendMatrix var_name,mdat
out = MLAppendMatrix(var_name,mdat)
```

Description

`= MLAppendMatrix(var_name,mdat)` appends data in `mdat` to MATLAB matrix `var_name` or creates `var_name` if it does not exist. *Use this syntax when working directly in a worksheet.*

`MLAppendMatrix var_name,mdat` appends data in `mdat` to MATLAB matrix `var_name` or creates `var_name` if it does not exist. *Use this syntax in a VBA macro.*

`out = MLAppendMatrix(var_name,mdat)` lets you catch errors when executing `MLAppendMatrix` in a VBA macro. If `MLAppendMatrix` fails, then `out` is a string containing error code. Otherwise, `out` is `0`.

Input Arguments

var_name

Name of MATLAB matrix to which to append data.

`var_name` in quotes directly specifies the matrix name. `var_name` without quotes specifies a worksheet cell address (or range name) that contains the matrix name. Do not use the MATLAB variable `ans` as `var_name`.

mdat

Location of data to append to `var_name`.

`mdat` must be a worksheet cell address or range name. Do not enclose it in quotes.

`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 character vectors if the data is a string.

Output Arguments

out

`0` if the command succeeded. Otherwise, a string containing error code.

Examples

Append Data from a Worksheet Cell Range to a MATLAB Matrix

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

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

		A1
		A2

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

Append Data from a Named Worksheet Cell Range to a MATLAB Matrix

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

`MLAppendMatrix(C1, new_data)`

A1	B1
A2	B2

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

Tips

- `MLAppendMatrix` checks the dimensions of `var_name` and `mdat` to determine how to append `mdat` to `var_name`. If the dimensions allow appending `mdat` as either new rows or new columns, it appends `mdat` to `var_name` as new rows. If the dimensions do not match, the function returns an error.

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

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

See Also

`MLPutMatrix`

Introduced before R2006a

MLAutoStart

Automatically start MATLAB

Syntax

```
= MLAutoStart(flag)
MLAutoStart flag
out = MLAutoStart(flag)
```

Description

`= MLAutoStart(flag)` sets automatic startup of the Spreadsheet Link and MATLAB software. A change of state takes effect the next time an Excel session starts. *Use this syntax when working directly in a worksheet.*

`MLAutoStart flag` sets automatic startup of the Spreadsheet Link and MATLAB software. A change of state takes effect the next time an Excel session starts. *Use this syntax in a VBA macro.*

`out = MLAutoStart(flag)` lets you catch errors when executing `MLAutoStart` in a VBA macro. If `MLAutoStart` fails, then `out` is a string containing error code. Otherwise, `out` is `0`.

Input Arguments

flag

Either `"yes"` or `"no"`.

Specify `"yes"` to automatically start the Spreadsheet Link and MATLAB software every time a Microsoft Excel session starts. Specify `"no"` to cancel automatic startup of the Spreadsheet Link and MATLAB software.

Default: `"yes"`

Output Arguments

out

`0` if the command succeeded. Otherwise, a string containing error code.

Examples

Cancel Automatic Startup of Spreadsheet Link and MATLAB

Enter this command in a worksheet:

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

Spreadsheet Link and MATLAB do not start on subsequent Excel session invocations.

Tips

- If Spreadsheet Link and MATLAB are running, then `MLAutoStart("no")` does not stop them.

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

Topics
"Start Spreadsheet Link and MATLAB Automatically" on page 1-15

Introduced before R2006a

MLProgramId

Specify MATLAB version

Syntax

```
= MLProgramId(version)
MLProgramId version
```

Description

`= MLProgramId(version)` specifies the MATLAB version to open when Spreadsheet Link starts in Microsoft Excel. Enter this syntax when working in a worksheet cell.

`MLProgramId version` specifies the MATLAB version. Enter this syntax when working in a VBA macro.

Examples

Specify MATLAB Version in Worksheet Cells

To open MATLAB (R2016a) in Microsoft Excel, enter this text in any worksheet cell:

```
=MLProgramId("9.0")
```

The worksheet cell displays `0` when `MLProgramId` runs.

To check that the preference is set, open the Preferences dialog box. The **MATLAB program id** box contains `9.0`. For details about the Preferences dialog box, see "Setting Spreadsheet Link Preferences" on page 1-12.

Start MATLAB.

For troubleshooting, see "Startup Errors" on page 3-13.

Specify MATLAB Version in VBA Macro

To open MATLAB (R2016a) in Microsoft Excel, enter this text at the beginning of the VBA macro:

```
MLProgramId "9.0"
```

Run the macro by clicking **Run Macro (F5)**. For details about running macros, see Excel Help.

To check that the preference is set, open the Preferences dialog box. The **MATLAB program id** box contains `9.0`. For details about the Preferences dialog box, see "Setting Spreadsheet Link Preferences" on page 1-12.

Start MATLAB.

For troubleshooting, see "Startup Errors" on page 3-13.

Input Arguments

version — MATLAB version
string

MATLAB version, specified as a string to indicate which MATLAB version to open when multiple versions are installed on the computer.

The Windows registry defines the MATLAB version number under `...\Mathworks \MATLAB`. For example, MATLAB R2016a corresponds to version 9.0 in the registry.

Example: `"9.0"`

See Also
`MLAutoStart | MLOpen`

Topics
"Setting Spreadsheet Link Preferences" on page 1-12
"Startup Errors" on page 3-13

Introduced in R2016b

MLClose

Stop MATLAB

Syntax

```
= MLClose()
MLClose
out = MLClose()
```

Description

`= MLClose()` ends the MATLAB process, deletes all variables from the MATLAB workspace, and tells the Microsoft Excel software that the MATLAB software is no longer running. *Use this syntax when working directly in a worksheet.*

`MLClose` ends the MATLAB process, deletes all variables from the MATLAB workspace, and tells the Microsoft Excel software that the MATLAB software is no longer running. *Use this syntax in a VBA macro.*

`out = MLClose()` lets you catch errors when executing `MLClose` in a VBA macro. If `MLClose` fails, then `out` is a string containing error code. Otherwise, `out` is `0`.

Output Arguments

out

`0` if the command succeeded. Otherwise, a string containing error code.

Examples

End the MATLAB Session

End the MATLAB session from a worksheet:

```
MLClose()
```

Tips

- If you use `MLClose` when no MATLAB process is running, nothing happens.

See Also

MLAutoStart | MLOpen

Topics

"Stop Spreadsheet Link and MATLAB" on page 1-17

Introduced before R2006a

MLDeleteMatrix

Delete MATLAB matrix

Syntax

```
= MLDeleteMatrix(var_name)
MLDeleteMatrix var_name
out = MLDeleteMatrix(var_name)
```

Description

`= MLDeleteMatrix(var_name)` deletes the named matrix from the MATLAB workspace. *Use this syntax when working directly in a worksheet.*

`MLDeleteMatrix var_name` deletes the named matrix from the MATLAB workspace. *Use this syntax in a VBA macro.*

`out = MLDeleteMatrix(var_name)` lets you catch errors when executing `MLDeleteMatrix` in a VBA macro. If `MLDeleteMatrix` fails, then `out` is a string containing error code. Otherwise, `out` is `0`.

Input Arguments

var_name

Name of MATLAB matrix to delete.

`var_name` in quotes directly specifies the matrix name. `var_name` without quotes specifies a worksheet cell address (or range name) that contains the matrix name.

Output Arguments

out

`0` if the command succeeded. Otherwise, a string containing error code.

Examples

Delete a Matrix from the MATLAB Workspace

Delete matrix A from the MATLAB workspace:

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

See Also
`MLAppendMatrix | MLGetMatrix | MLPutMatrix`

Introduced before R2006a

MLEvalString

Evaluate MATLAB command in MATLAB

Syntax

```
= MLEvalString(command)

MLEvalString command
err = MLEvalString(command)
```

Description

`= MLEvalString(command)` specifies the MATLAB command for evaluation in the MATLAB workspace. Use this syntax when working in a worksheet cell.

`MLEvalString command` works in a VBA macro.

`err = MLEvalString(command)` returns the execution status when executing `MLEvalString` in a VBA macro.

Examples

Create Diagonal Matrix in Worksheet Cells

Enter the variable `a` into cell `A1`. Enter the numbers 1 through 5 into the range of cells from `B1` through `F1`.

Assign the range of cells to variable `a` in MATLAB using `MLPutMatrix`. Enter this text in cell `A3`.

```
=MLPutMatrix(A1,B1:F1)
```

	A	B	C	D	E	F
1	a	1	2	3	4	5
2						
3	=MLPutMatrix(A1,B1:F1)					

Use `diag` to create a matrix `b`, containing a diagonal using the five numbers in variable `a`. Enter this text in cell A5.

```
=MLEvalString("b = diag(a);")
```

	A	B	C	D	E	F
1	a	1	2	3	4	5
2						
3	0					
4						
5	=MLEvalString("b = diag(a)")					

Retrieve matrix `b` from MATLAB into Excel, cell A9. Enter this text in cell A7.

```
=MLGetMatrix("b","A9")
```

The matrix with the diagonal appears in cells A9 through E13.

	A	B	C	D	E	F
1	a	1	2	3	4	5
2						
3	0					
4						
5	0					
6						
7	0					
8						
9	1	0	0	0	0	
10	0	2	0	0	0	
11	0	0	3	0	0	
12	0	0	0	4	0	
13	0	0	0	0	5	

Create Diagonal Matrix in VBA Macro

Enter the variable a into cell A1. Enter the numbers 1 through 5 into the range of cells from B1 through F1.

Click the **Developer** tab in Microsoft Excel, and then click **Visual Basic**. The Visual Basic Editor window opens.

Insert a new module and enter this VBA code into the Code section of the window. This sample code assumes a macro named Diagonal. For details on working with modules, see Excel Help.

```
Sub Diagonal()

    MLPutMatrix "a", Range("B1:F1")
    MLEvalString "b = diag(a);"
    MLGetMatrix "b", "A3"
    MatlabRequest

End Sub
```

Run the macro by clicking **Run Sub/UserForm (F5)**. The diagonal matrix appears in cells A3 through E7. For details on running macros, see Excel Help.

	A	B	C	D	E	F
1	a	1	2	3	4	5
2						
3	1	0	0	0	0	
4	0	2	0	0	0	
5	0	0	3	0	0	
6	0	0	0	4	0	
7	0	0	0	0	5	

Return Error for Invalid Command

Enter the variable a into cell A1. Enter the numbers 1 through 5 into the range of cells from B1 through F1.

Click the **Developer** tab in Microsoft Excel, and then click **Visual Basic**. The Visual Basic Editor window opens.

Insert a new module and enter this invalid VBA code into the Code section of the window. This sample code assumes a macro named Diagonal. For details on working with modules, see Excel Help.

```
Sub Diagonal()

    Dim err As Variant

    MLPutMatrix "a", Range("B1:F1")
    err = MLEvalString("b = diag(2a);") 'Invalid code

    If err <> 0 Then
        MsgBox err
    End If

    MLGetMatrix "b", "A3"
    MatlabRequest

End Sub
```

4-27

Run the macro by clicking **Run Sub/UserForm (F5)**. For details on running macros, see Excel Help.

This Spreadsheet Link error displays: `#COMMAND!`. To display MATLAB errors, see `MLShowMatlabErrors`.

Input Arguments

`command` — MATLAB command to evaluate
string

MATLAB command to evaluate, specified as a string. Enclose the string in double quotes. Or, enter the string in a cell without quotes and enter the corresponding cell reference without quotes as the input argument.

Example: `"sum"`

Example: `A1`

Output Arguments

`err` — Execution status
string | number

Execution status, returned as a string or number. If `MLEvalString` fails, then `err` is a string containing an error code or error message. Otherwise, the command executes successfully and `err` is `0`.

By default when `MLEvalString` fails, `err` contains a standard Spreadsheet Link error, such as `#COMMAND`. To return MATLAB errors, execute `MLShowMatlabErrors`.

Tip

- The specified action alters only the MATLAB workspace and has no effect on the Microsoft Excel workspace.

See Also

`MLGetMatrix` | `MLPutMatrix` | `MLShowMatlabErrors` | `diag`

Topics

"Create Diagonal Matrix Using Worksheet Cells" on page 1-25
"Create Diagonal Matrix Using VBA Macro" on page 1-28
"Model Data Using Regression and Curve Fitting" on page 2-2
"Interpolate Thermodynamic Data" on page 2-10
"Price Stock Options Using Binomial Model" on page 2-15
"Executing Spreadsheet Link Functions" on page 1-42
"Worksheet Cell Errors" on page 3-2
"Microsoft Excel Errors" on page 3-6

Introduced before R2006a

MLGetFigure

Import current MATLAB figure into Microsoft Excel worksheet

Syntax

```
= MLGetFigure(width,height)
MLGetFigure width, height
out = MLGetFigure(width,height)
```

Description

`= MLGetFigure(width,height)` import the current MATLAB figure into an Excel worksheet, where the top-left corner of the figure is the current worksheet cell. *Use this syntax when working directly in a worksheet.*

`MLGetFigure width, height` import the current MATLAB figure into an Excel worksheet, where the top-left corner of the figure is the current worksheet cell. *Use this syntax in a VBA macro.*

`out = MLGetFigure(width,height)` lets you catch errors when executing `MLGetFigure` in a VBA macro. If `MLGetFigure` fails, then `out` is a string containing error code. Otherwise, `out` is 0.

Input Arguments

width

Width (in normalized units) of the MATLAB figure when imported into an Excel worksheet.

height

Height (in normalized units) of the MATLAB figure when imported into an Excel worksheet.

Output Arguments

out

`0` if the command succeeded. Otherwise, a string containing error code.

Examples

Import a MATLAB Figure into an Excel Worksheet

Import the current MATLAB figure into an Excel worksheet. Specify the width and the height of the figure to be half those of the original figure:

```
MLGetFigure(.5,.5)
```

Note that if you use Microsoft Excel 2007 or 2010, the width and the height of the imported figure will be a quarter of those of the original figure.

Tips

- If you use Microsoft Excel 2007 or 2010, `MLGetFigure` scales the imported figure by the product of `width` and `height` along both dimensions.

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

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

See Also

MLGetMatrix | MLGetVar

Introduced in R2006b

MLGetMatrix

Write contents of MATLAB matrix to Microsoft Excel worksheet

Syntax

```
= MLGetMatrix(var_name,edat)
MLGetMatrix var_name, edat
out = MLGetMatrix(var_name,edat)
```

Description

`= MLGetMatrix(var_name,edat)` writes the contents of MATLAB matrix `var_name` in the Excel worksheet, beginning in the upper-left cell specified by `edat`. *Use this syntax when working directly in a worksheet.*

`MLGetMatrix var_name, edat` writes the contents of MATLAB matrix `var_name` in the Excel worksheet, beginning in the upper-left cell specified by `edat`. *Use this syntax in a VBA macro.*

`out = MLGetMatrix(var_name,edat)` lets you catch errors when executing `MLGetMatrix` in a VBA macro. If `MLGetMatrix` fails, then `out` is a string containing error code. Otherwise, `out` is `0`.

Input Arguments

var_name

Name of MATLAB matrix to access.

`var_name` in quotes directly specifies the matrix name. `var_name` without quotes specifies a worksheet cell address (or range name) that contains the matrix name. Do not use the MATLAB variable `ans` as `var_name`.

4-33

edat

Worksheet location where the function writes the contents of var_name.

edat in quotes directly specifies the location. edat without quotes specifies a worksheet cell address (or range name) that contains a reference to the location. In both cases, edat must be a cell address or a range name.

Output Arguments

out

0 if the command succeeded. Otherwise, a string containing error code.

Examples

Specify the Matrix Name and Location Directly

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

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

Specify the Matrix Name and Location Indirectly

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

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

Use MLGetMatrix in a VBA Macro

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

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

Use the Address Property of the Range Object to Specify Location

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

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

Catch Errors in a VBA Macro

Use this function to get the variable A from MATLAB and to test if the command succeeded:

```
Sub myfun()
    Dim out As Variant

    out = MLGetMatrix("A", "A1")

    If out <> 0 Then
        MsgBox out
    End If
    MatlabRequest
End Sub
```

If `MLGetMatrix` fails, `myfun` displays a message box with the error code.

Tips

- If data exists in the specified worksheet cells, it is overwritten.
- If the dimensions of the MATLAB matrix are larger than that of the specified cells, the data overflows into additional rows and columns.

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

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

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

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

- If you use `MLGetMatrix` in a macro subroutine, enter `MatlabRequest` on the line after the `MLGetMatrix`. `MatlabRequest` initializes internal Spreadsheet Link 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.

See Also

`MLAppendMatrix` | `MLPutMatrix` | `MLPutRanges`

Topics

"Create Diagonal Matrix Using Microsoft Excel Ribbon" on page 1-18
"Create Diagonal Matrix Using Microsoft Excel Context Menu" on page 1-21
"Create Diagonal Matrix Using Worksheet Cells" on page 1-25
"Create Diagonal Matrix Using VBA Macro" on page 1-28
"Find and Execute MATLAB Function Using MATLAB Function Wizard" on page 1-30
"Executing Spreadsheet Link Functions" on page 1-42

Introduced before R2006a

MLGetVar

Write contents of MATLAB matrix in Microsoft Excel VBA variable

Syntax

```
MLGetVar ML_var_name, VBA_var_name
```

Description

`MLGetVar ML_var_name, VBA_var_name` 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`.

Input Arguments

`ML_var_name`

Name of MATLAB matrix to access.

`ML_var_name` in quotes directly specifies the matrix name. `ML_var_name` without quotes specifies a VBA variable that contains the matrix name as a string. Do not use the MATLAB variable `ans` as `ML_var_name`. If defined, `ML_var_name` must be of type `VARIANT`. Any other type will give a `"TYPE MISMATCH"` error.

`VBA_var_name`

Name of VBA variable where the function writes the contents of `ML_var_name`.

Use `VBA_var_name` without quotes.

Examples

Write the Contents of a MATLAB Matrix into a VBA Variable

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

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

See Also
MLPutVar

Topics
"Create Diagonal Matrix Using VBA Macro" on page 1-28
"Executing Spreadsheet Link Functions" on page 1-42

Introduced before R2006a

MLMissingDataAsNaN

Set empty cells to NaN or 0

Syntax

```
= MLMissingDataAsNaN(flag)
MLMissingDataAsNaN flag
out = MLMissingDataAsNaN(flag)
```

Description

= MLMissingDataAsNaN(flag) sets empty cells to NaN or 0. When the Spreadsheet Link software is installed, the default is "no", so empty cells are handled as 0s. If you change the value of MLMissingDataAsNaN to "yes", the change remains in effect the next time a Microsoft Excel session starts. *Use this syntax when working directly in a worksheet.*

MLMissingDataAsNaN flag sets empty cells to NaN or 0. *Use this syntax in a VBA macro.*

out = MLMissingDataAsNaN(flag) lets you catch errors when executing MLMissingDataAsNaN in a VBA macro. If MLMissingDataAsNaN fails, then out is a string containing error code. Otherwise, out is 0.

Input Arguments

flag

Either "yes" or "no".

Specify "yes" to set empty cells to use NaNs. Specify "no" to set empty cells to use 0s.

Default: "no"

Output Arguments

out

`0` if the command succeeded. Otherwise, a string containing error code.

Examples

Set Empty Cells to Use 0s

Cancel the use of the value `NaN` for empty cells:

`=MLMissingDataAsNaN("no")`

Tips

- A string in an Excel range always forces cell array output and empty cells as `NaN`s.

See Also
`MLPutMatrix`

Introduced in R2007a

MLOpen

Start MATLAB

Syntax

```
= MLOpen()
MLOpen
out = MLOpen()
```

Description

`= MLOpen()` starts MATLAB process. Use `MLOpen` to restart the MATLAB session after you have stopped it with `MLClose` in a given Microsoft Excel session. *Use this syntax when working directly in a worksheet.*

`MLOpen` starts MATLAB process. Use `MLOpen` to restart the MATLAB session after you have stopped it with `MLClose` in a given Microsoft Excel session. *Use this syntax in a VBA macro.*

`out = MLOpen()` lets you catch errors when executing `MLOpen` in a VBA macro. If `MLOpen` fails, then `out` is a string containing error code. Otherwise, `out` is `0`.

Output Arguments

out

`0` if the command succeeded. Otherwise, a string containing error code.

Examples

Start a MATLAB Session

Start a MATLAB session from a worksheet:

```
MLOpen()
```

Tips

- If a MATLAB process has already started, subsequent calls to `MLOpen` do nothing.
- To start a MATLAB session and initialize the Spreadsheet Link software, use `matlabinit` rather than `MLOpen`.

See Also

`MLClose` | `matlabinit`

Introduced before R2006a

MLPutMatrix

Create or overwrite MATLAB matrix with data from Microsoft Excel worksheet

Syntax

```
= MLPutMatrix(var_name, mdat)
MLPutMatrix var_name, mdat
out = MLPutMatrix(var_name,mdat)
```

Description

`= MLPutMatrix(var_name, mdat)` creates or overwrites matrix `var_name` in MATLAB workspace with specified data in `mdat`. Creates `var_name` if it does not exist. *Use this syntax when working directly in a worksheet.*

`MLPutMatrix var_name, mdat` creates or overwrites matrix `var_name` in MATLAB workspace with specified data in `mdat`. *Use this syntax in a VBA macro.*

`out = MLPutMatrix(var_name,mdat)` lets you catch errors when executing `MLPutMatrix` in a VBA macro. If `MLPutMatrix` fails, then `out` is a string containing error code. Otherwise, `out` is `0`.

Input Arguments

var_name

Name of MATLAB matrix to create or overwrite.

`var_name` in quotes directly specifies the matrix name. `var_name` without quotes specifies a worksheet cell address (or range name) that contains the matrix name.

mdat

Location of data to copy into `var_name`.

`mdat` must be a worksheet cell address or range name. Do not enclose it in quotes.

Output Arguments

out

`0` if the command succeeded. Otherwise, a string containing error code.

Examples

Create or Overwrite a Matrix in the MATLAB Workspace

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

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

Import Data from a Microsoft Excel Worksheet to the MATLAB Workspace Using the putmatrix Toolbar Button

1   In the Excel worksheet, select the columns and/or rows you want to export to the MATLAB workspace.
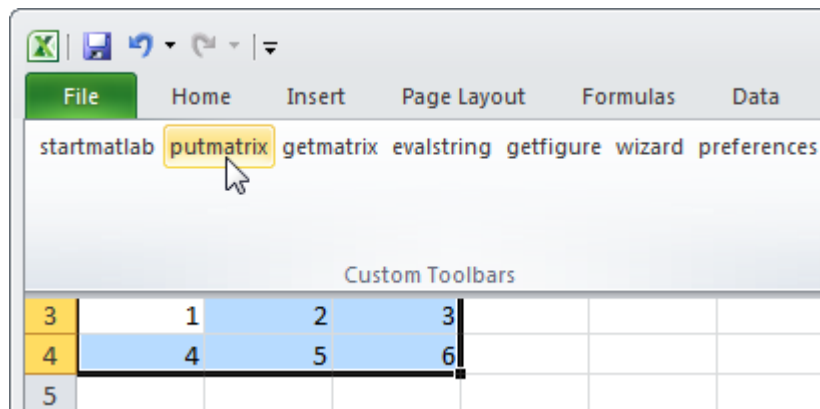
2   Click the **putmatrix** button on the Spreadsheet Link toolbar. A window appears that prompts you to specify the name of the MATLAB variable in which you want to store your data.
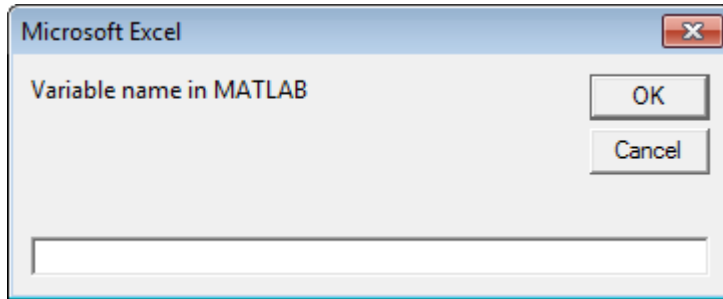
3   Enter `newmatrix` for the MATLAB variable name.

4   Click **OK**.

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

```
newmatrix
newmatrix =

     1     2     3
     4     5     6
```

Tips

- If `var_name` exists, this function replaces the contents with `mdat`.

- Empty numeric data cells within the range of `mdat` become numeric zeros within the MATLAB matrix identified by `var_name`.

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

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

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

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

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

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

See Also

`MLAppendMatrix | MLGetMatrix | MLPutRanges`

Topics

"Create Diagonal Matrix Using Microsoft Excel Ribbon" on page 1-18
"Create Diagonal Matrix Using Microsoft Excel Context Menu" on page 1-21
"Create Diagonal Matrix Using Worksheet Cells" on page 1-25
"Create Diagonal Matrix Using VBA Macro" on page 1-28
"Find and Execute MATLAB Function Using MATLAB Function Wizard" on page 1-30
"Executing Spreadsheet Link Functions" on page 1-42

Introduced before R2006a

MLPutRanges

Send data in Microsoft Excel named ranges to MATLAB

Syntax

```
= MLPutRanges()

MLPutRanges
out = MLPutRanges()
```

Description

`= MLPutRanges()` writes the named cell ranges in a Microsoft Excel worksheet into MATLAB variables. The variables are named with the same name specified by the cell range name in Microsoft Excel. To use this syntax, right-click in any Microsoft Excel cell, enter this syntax, and press **Enter**.

`MLPutRanges` writes the named cell ranges in a Microsoft Excel worksheet into MATLAB variables. The variables are named with the same name specified by the cell range name in Microsoft Excel. Use this syntax when working directly in a Microsoft Visual Basic macro.

`out = MLPutRanges()` returns the named cell ranges in a Microsoft Excel worksheet into MATLAB variables. The variables are named with the same name specified by the cell range name in Microsoft Excel. In this case, `out` specifies whether the `MLPutRanges` function executed successfully. Use this syntax when working directly in a Microsoft Visual Basic macro.

Examples

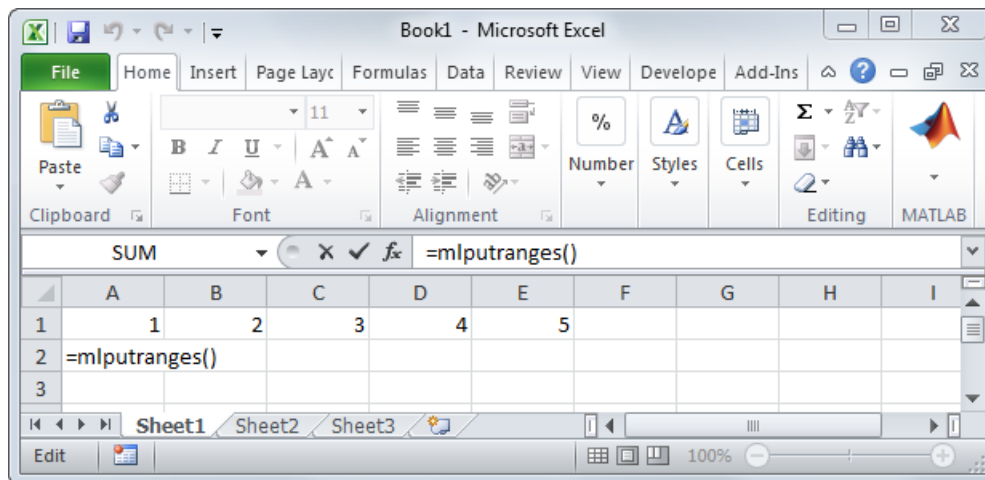**Export Microsoft Excel Named Range to MATLAB in Microsoft Excel Cell**

Define a name for a range of cells from cell A1 through cell E1. For instructions about defining names, see Excel Help and enter the search term: define and use names in formulas.

The name of the range of cells `testData` appears in the **Name Box**.
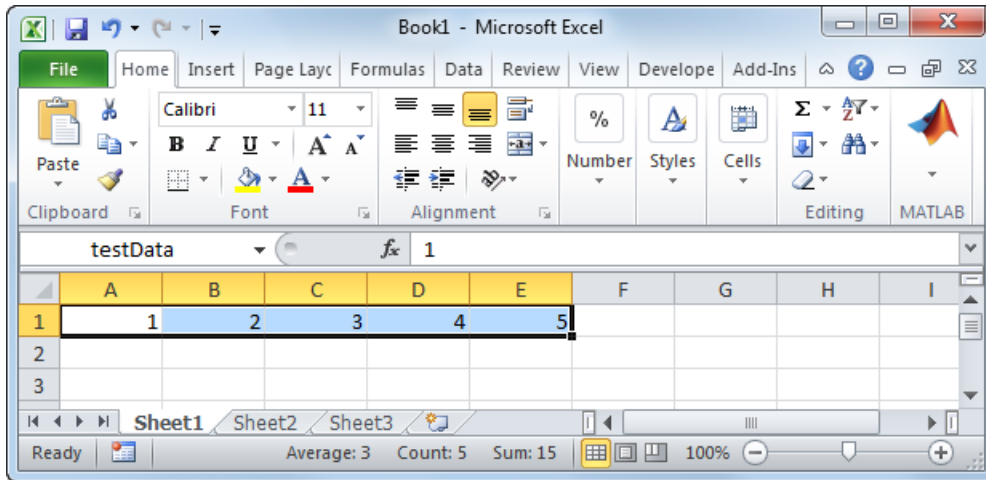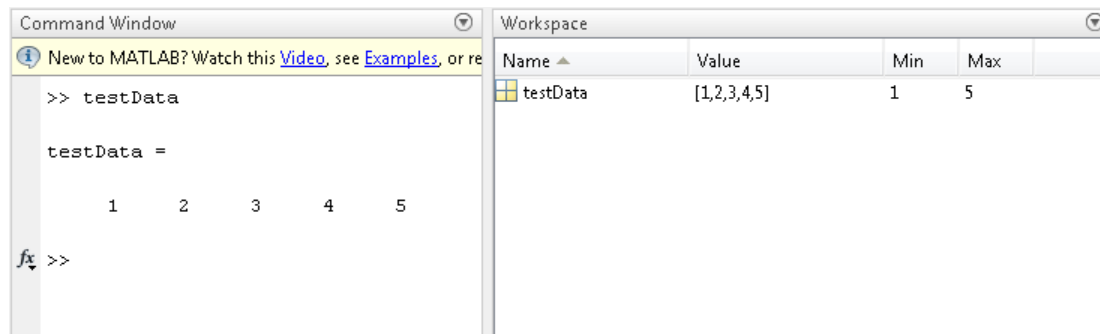

To send data in the named range in the current worksheet to MATLAB, execute the function inside a worksheet cell.

```
= MLPutRanges()
```

After pressing **Enter**, Microsoft Excel exports the named range `testData` to the MATLAB variable `testData` in the MATLAB workspace.

Export Microsoft Excel Named Ranges to MATLAB in Microsoft Visual Basic Macro Without Output

Call the function to send data in the named ranges in the current worksheet to MATLAB.

MLPutRanges

Export Microsoft Excel Named Ranges to MATLAB in Microsoft Visual Basic Macro with Output

Call the function to send data in the named ranges in the current worksheet to MATLAB.

```
out = MLPutRanges()
```

`out` returns 0 if the function succeeded or a string with the corresponding error code if the function failed.

Output Arguments

out — Status
0 | string

Status for execution of `MLPutRanges`, returned as 0 if the function succeeded, or a string containing an error code.

See Also

`MLGetMatrix` | `MLPutMatrix`

Topics
"Create Diagonal Matrix Using Microsoft Excel Ribbon" on page 1-18
"Create Diagonal Matrix Using Microsoft Excel Context Menu" on page 1-21
"Create Diagonal Matrix Using Worksheet Cells" on page 1-25
"Create Diagonal Matrix Using VBA Macro" on page 1-28
"Find and Execute MATLAB Function Using MATLAB Function Wizard" on page 1-30
"Executing Spreadsheet Link Functions" on page 1-42

Introduced in R2013b

MLPutVar

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

Syntax

```
MLPutVar ML_var_name, VBA_var_name
out = MLPutVar ML_var_name, VBA_var_name
```

Description

`MLPutVar ML_var_name, VBA_var_name` creates or overwrites matrix `ML_var_name` in MATLAB workspace with data in `VBA_var_name`. Creates `ML_var_name` if it does not exist. If `ML_var_name` exists, this function replaces the contents with data from `VBA_var_name`.

`out = MLPutVar ML_var_name, VBA_var_name` lets you catch errors when executing `MLPutVar`. If `MLPutVar` fails, then `out` is a string containing error code. Otherwise, `out` is `0`.

Input Arguments

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 specifies a VBA variable that contains 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.

Output Arguments

out

`0` if the command succeeded. Otherwise, a string containing error code.

Examples

Create a MATLAB Matrix Using Data Stored in a VBA Variable

Create (or overwrite) the MATLAB matrix K with the data in the VBA variable `DataK`:

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

Tips

- 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.

See Also
`MLGetVar`

Topics
"Create Diagonal Matrix Using VBA Macro" on page 1-28
"Executing Spreadsheet Link Functions" on page 1-42

Introduced before R2006a

MLShowMatlabErrors

Return standard Spreadsheet Link errors or full MATLAB errors

Syntax

```
= MLShowMatlabErrors(flag)
MLShowMatlabErrors flag
```

Description

`= MLShowMatlabErrors(flag)` sets the error message type when executing MATLAB commands using `MLEvalString` in Microsoft Excel. An error displays when `MLEvalString` fails. You can display two different types of error messages. At any point, you can switch between displaying Spreadsheet Link and MATLAB errors. Without this function, worksheet cells display only Excel error messages. Use this syntax when working in a worksheet cell.

`MLShowMatlabErrors flag` works in a VBA macro.

Examples

Display Spreadsheet Link Errors in Worksheet Cell

This Spreadsheet Link code assumes `MLEvalString` returns MATLAB errors upon failure.

Enter this text in any worksheet cell to display `MLEvalString` failures as Spreadsheet Link errors.

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

Enter this invalid text in any worksheet cell.

```
=MLEvalString("sum(2+b);")
```

This Spreadsheet Link error appears in the calling cell.

```
#COMMAND!
```

Display MATLAB Errors in VBA Macro

This VBA code assumes `MLEvalString` returns Spreadsheet Link errors upon failure.

Enter this text at the beginning of a VBA macro to display `MLEvalString` failures as MATLAB errors.

```
MLShowMatlabErrors "yes"
```

Enter this invalid text in the VBA macro.

```
out = MLEvalString("sum(2+b);")
MsgBox (out)
```

When running this macro, this MATLAB error appears in a dialog box: `??? Undefined function or variable 'b'`. For details on running macros, see Excel Help.

Input Arguments

`flag` — Error message indicator
`"no"` (default) | `"yes"`

Error message indicator, specified as `"no"` or `"yes"` to determine the type of error message displayed when `MLEvalString` fails. To display the standard Spreadsheet Link errors, specify `"no"`. To display the full MATLAB errors, specify `"yes"`.

See Also
`MLEvalString`

Topics

Introduced in R2006b

MLStartDir

Specify MATLAB current working folder after startup

Syntax

```
= MLStartDir(path)
MLStartDir path
out = MLStartDir(path)
```

Description

`= MLStartDir(path)` sets the MATLAB working folder after startup. *Use this syntax when working directly in a worksheet.*

`MLStartDir path` sets the MATLAB working folder after startup. *Use this syntax in a VBA macro.*

`out = MLStartDir(path)` lets you catch errors when executing `MLStartDir` in a VBA macro. If `MLStartDir` fails, then `out` is a string containing error code. Otherwise, `out` is `0`.

Input Arguments

path

Path to the new MATLAB working folder after startup.

Output Arguments

out

Execution status that contains `0` if the command succeeded. Otherwise, `out` contains a string with an error code.

Examples

Specify MATLAB Working Folder

Set the MATLAB working folder to `d:\work` after start up in a worksheet cell.

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

Specify MATLAB Working Folder That Includes Spaces

If your folder path includes a space, embed the path in single quotation marks within double quotation marks.

Set the MATLAB working folder to `d:\my work` in a VBA macro.

```
MLStartDir "'d:\my work'"
```

Specify MATLAB Working Folder with Execution Status

Set the MATLAB working folder to `d:\work` after start up in a VBA macro. Return the execution status `out`.

```
out = MLStartDir("d:\work")
```

Tips

- This function does not work like the standard Microsoft Windows **Start In** setting, because it does not automatically run `startup.m` or `matlabrc.m` in the specified folder.
- The working folder changes only if you run MATLAB *after* you run this function. Running this function while MATLAB is running does not change the working folder for the current session. In this case, MATLAB uses the specified folder as the working folder when it is restarted.

See Also

`MLAutoStart`

Introduced in R2006b

MLUseCellArray

Toggle `MLPutMatrix` to use MATLAB cell arrays

Syntax

```
= MLUseCellArray(flag)
MLUseCellArray flag
out = MLUseCellArray(flag)
```

Description

`= MLUseCellArray(flag)` specifies whether `MLPutMatrix` must use cell arrays for transfer of data (for example, dates). When the Spreadsheet Link software is installed, the default is `"no"`. If you change the value of `MLUseCellArray` to `"yes"`, the change remains in effect the next time a Microsoft Excel session starts. *Use this syntax when working directly in a worksheet.*

`MLUseCellArray flag` specifies whether `MLPutMatrix` must use cell arrays for transfer of data. *Use this syntax in a VBA macro.*

`out = MLUseCellArray(flag)` lets you catch errors when executing `MLUseCellArray` in a VBA macro. If `MLUseCellArray` fails, then `out` is a string containing error code. Otherwise, `out` is 0.

Input Arguments

flag

Either `"yes"` or `"no"`.

Specify `"yes"` to automatically uses cell arrays for transfer of data structures. Specify `"no"` to stop using cell arrays for transfer of data structures.

Default: `"no"`

Output Arguments

out

`0` if the command succeeded. Otherwise, a string containing error code.

Examples

Stop Using Cell Arrays When Transferring Data Structures

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

`MLUseCellArray("no")`

See Also
`MLPutMatrix`

Introduced in R2007a

MLUseFullDesktop

Specify whether to use full MATLAB desktop or Command Window

Syntax

```
= MLUseFullDesktop(flag)
MLUseFullDesktop flag
out = MLUseFullDesktop(flag)
```

Description

`= MLUseFullDesktop(flag)` sets the MATLAB session to start with the full desktop or Command Window only. *Use this syntax when working directly in a worksheet.*

`MLUseFullDesktop flag` sets the MATLAB session to start with the full desktop or Command Window only. *Use this syntax in a VBA macro.*

`out = MLUseFullDesktop(flag)` lets you catch errors when executing `MLUseFullDesktop` in a VBA macro. If `MLUseFullDesktop` fails, then `out` is a string containing error code. Otherwise, `out` is `0`.

Input Arguments

flag

Either `"yes"` or `"no"`.

Specify `"yes"` to start full MATLAB desktop. Specify `"no"` to start the Command Window only.

Default: `"yes"`

Output Arguments

out

`0` if the command succeeded. Otherwise, a string containing error code.

Examples

Start Only the Command Window

Set the MATLAB session to start with the command window only:

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

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

Introduced in R2006b