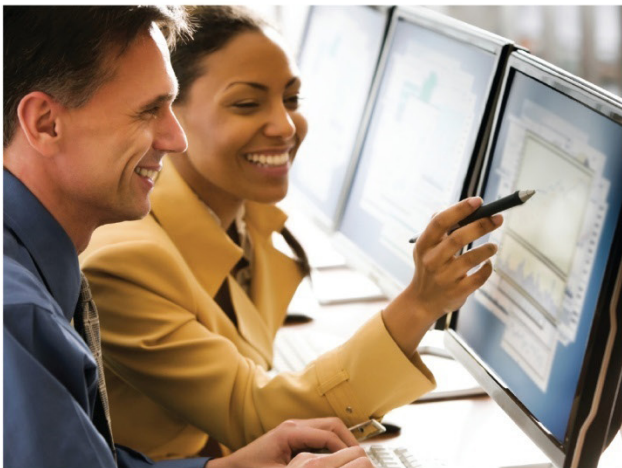


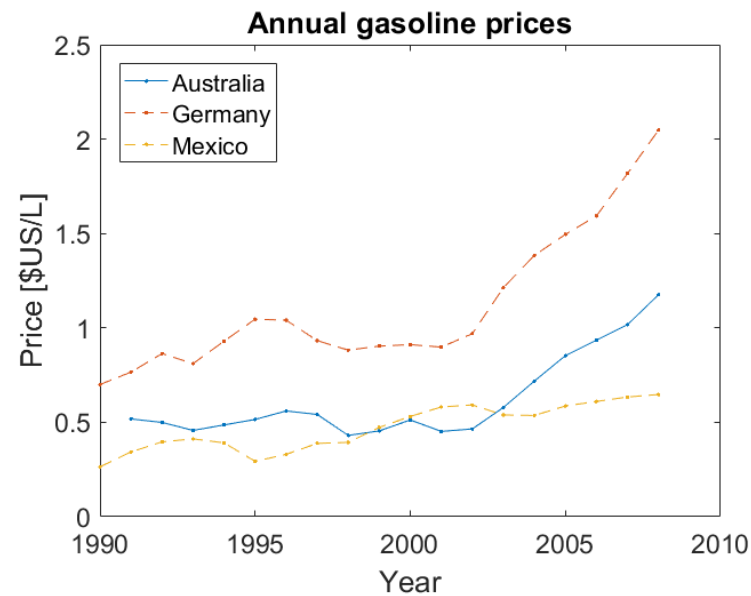
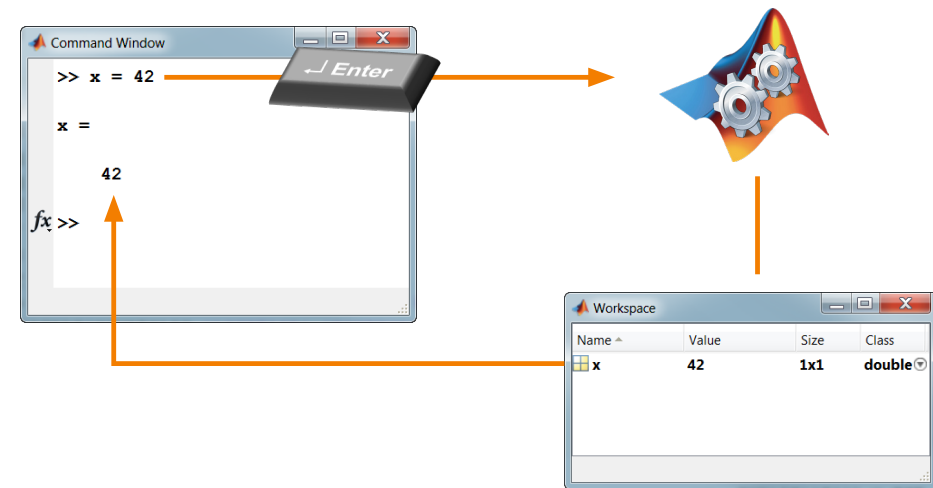
Variables and Commands

MATLAB® Fundamentals for Aerospace Applications



Outline

- Entering commands
- Creating numeric variables
- Creating character variables
- Making and annotating plots
- Getting help
- Creating and running scripts
- Formatting live scripts



Chapter Learning Outcomes

The attendee will be able to:

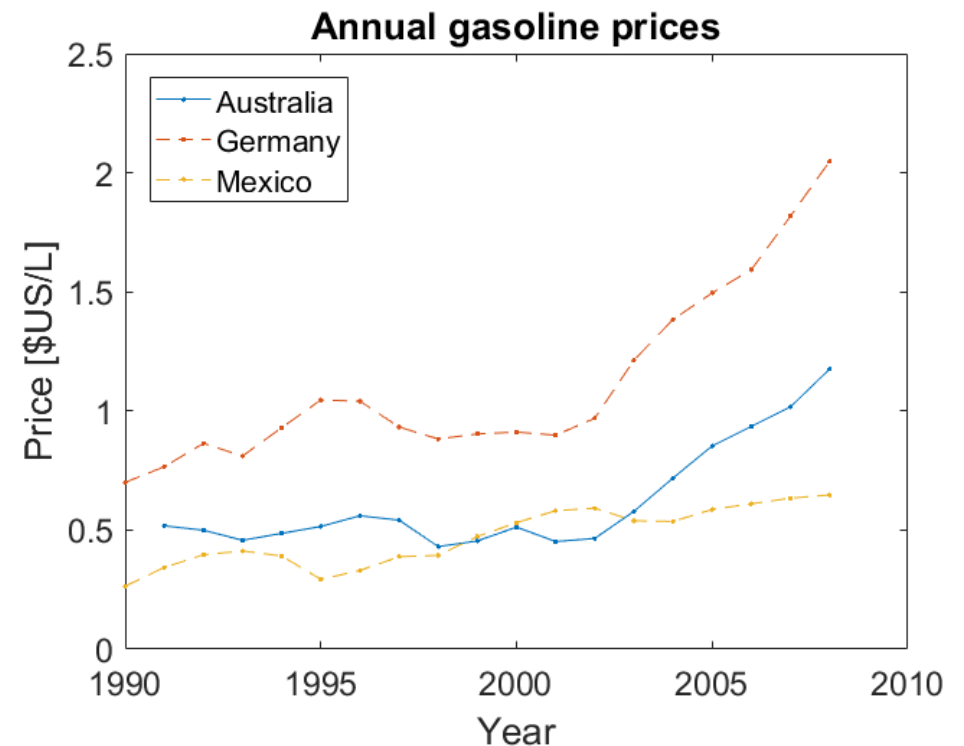
- Issue MATLAB® commands in the Command Window.
- Create new variables, and apply arithmetic operations and functions to existing variables.
- Create text variables.
- Create two-dimensional plots of vector data.
- Obtain help on MATLAB commands and navigate the documentation browser.
- Label plots and adjust plot elements such as line style and color.
- Use the Command History and the MATLAB® Live Editor to write, save, and run script files.
- Use code sections to partition large scripts into smaller parts.
- Format live scripts to provide user help and increase readability.

Course Example: Comparing Prices Visually

Performing any analysis or modeling generally requires creating, extracting, and modifying data in variables. Issuing commands in the Command Window allows you to automate this process and makes it easier to control the details of your computations and visualizations.

The file `AuDeMx.mat` contains gasoline prices for Australia, Germany, and Mexico, as would be created by importing the data in `gasprices.xlsx` with the Import Tool.

The goal of this example is to use commands to create a visualization such as the one to the right. This requires creating and modifying variables and using built-in MATLAB functions. A script containing the commands is created to easily recreate or modify the analysis.

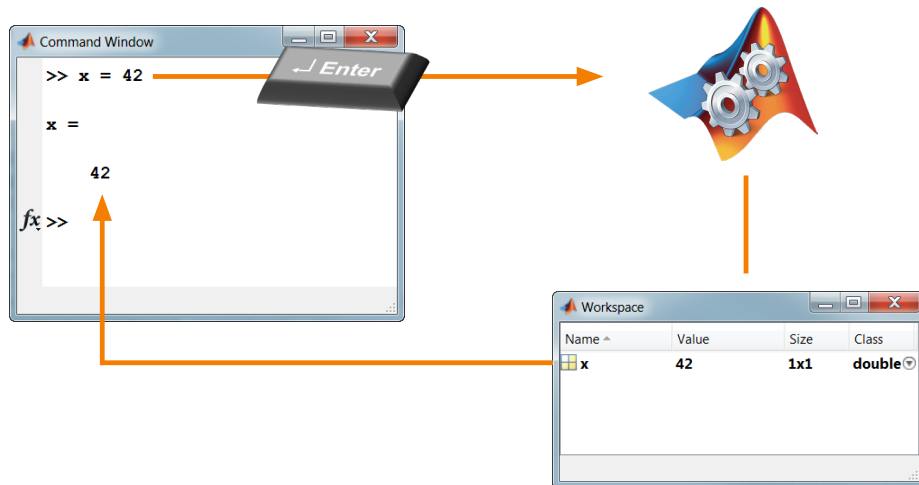


Entering Commands

Enter commands at the MATLAB prompt (`>>`) in the Command Window. MATLAB interprets and executes the command when you press the **Enter** key:

```
>> x = 42
x =
    42
```

If the command generates text output, the output is shown in the Command Window.



There is no required line-ending character in MATLAB. A semicolon (`;`) at the end of a command suppresses any output, although the command is still executed:

```
>> y = 3.14;
```

MATLAB is case sensitive: name is not the same as Name or NAME.

Try

Enter some commands.

```
>> x = 42
>> y = 3.14;
```

Verify that `y` now exists in the workspace and has the value of 3.14 (even though no output is displayed to the Command Window).

Although the Command Window shows a record of the commands issued and the output returned, this record is not editable: only the current line can be edited.

A number of keyboard shortcuts are available to assist in entering commands:

Home, End Move to the beginning, end of an expression

↑, ↓ Scroll through previous commands

Esc Deletes the expression at the prompt

Tab Displays commands that complete a partial expression

Useful commands for managing the Command Window are:

`clc` Clears the Command Window

`home` Moves the prompt to the top of the Command Window (without clearing commands)

Getting Data into MATLAB®

Each time MATLAB starts, the workspace is initially empty. You can import data from a file interactively, as shown in Chapter 2 (“Working with the MATLAB User Interface”). Alternatively you can use commands to load or import data or to create data directly in MATLAB.

Data is stored in MATLAB as *variables*. Each variable in memory has a unique name that:

- Is anywhere from 1 to 63 characters long.
- Consists of any combination of letters (a-z and A-Z), numbers (0-9), and the underscore (_) character.
- Starts with a letter.

For example:

Valid Variable Names	Invalid Variable Names
x	X-Y
Prices	month&year
dailyAvgPower	1998value
CO2_conc	CO2 conc

As with commands, variable names are case sensitive.

Try

Load gasoline prices from a MAT-file.

```
>> load AuDeMx
```

You can load data from a MAT-file with the `load` command:

```
load myfile
```

This loads the data from `myfile.mat`. Note that you do not need to provide the `.mat` extension.

By default, all variables in the MAT-file are loaded. You can specify particular variables to load:

```
load myfile var1 var3
```

Similarly, you can save variables in your workspace to a MAT-file with the `save` command:

```
save myfile var1 var3
```

This saves variables `var1` and `var3` to a MAT-file called `myfile.mat`. If this file does not exist, it is created. If it already exists, it is overwritten. To add variables to an existing MAT-file, rather than overwriting, use the `-append` option:

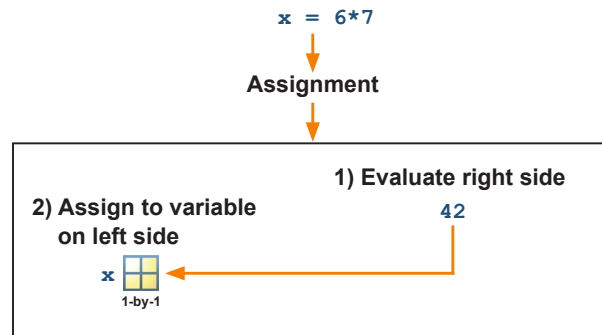
```
save myfile var2 -append
```

Assigning Values to Variables

You can assign values to a variable using the equals sign (=), which is the assignment operator in MATLAB:

```
x = 6*7;
```

MATLAB computes the value of the expression on the right of the equals sign and then assigns that value to the variable named on the left.



There is no requirement in MATLAB to declare the name, type, or size of a variable before making an assignment statement. Variables are created and modified as needed. If the variable on the left already exists, it is overwritten with the new value.

The expression to the right of the equals sign can be anything that MATLAB can evaluate. It can therefore reference any variable in memory. There is no retroactive calculation if the value of that variable changes subsequently:

```
x = 42;
y = x/6;    % y has the value 7
x = 3.14;   % y still has the value 7
y = y - 1;  % y is now 6
```

When a computation is carried out but not explicitly assigned to a variable, MATLAB assigns the output to the default variable `ans`.

```
>> 6*7
ans =
    42
```

Try

Create some variables.

```
>> x = 6*7
>> a = 0;
>> A = 1;
```

Redefine an existing variable.

```
>> x = a + 2*A
```

Change the value of an existing variable.

```
>> x = x + 1
```

Define a conversion constant for converting from gallons to liters.

```
>> gal2lit = 0.2642;
```

Convert Australian prices from gallons to liters.

```
>> Australia = gal2lit*Australia;
```

Entering just the name of a variable at the command prompt, with no assignment or calculation, returns the current value of that variable:

```
>> x
x =
    3.1400
```

Variables remain in memory until explicitly cleared:

```
clear x    % deletes just x
clear      % deletes all variables
```

Using Built-In Functions and Constants

MATLAB contains a large number of standard mathematical functions, such as square root (`sqrt`), natural logarithm (`log`), exponential (`exp`), trigonometric functions (`sin`, `cos`, `tan`), inverse trigonometric functions (`asin`, `acos`, `atan`), and so on.

Functions in MATLAB are evaluated by including the input value(s) in parentheses (`()`):

```
x = sqrt(42)
y = atan(x)
```

Calculations can be nested:

```
z = exp(-2*sqrt(42)/3)
```

Calculations in MATLAB obey the normal mathematical rules of precedence. You can also use parentheses to group terms for precedence or clarity:

```
z = log((42 - exp(-x))/2)
```

MATLAB provides the mathematical constants `pi` (π) and `i` ($i = \sqrt{-1}$). You can also use the notations `j`, `1i`, and `1j` to refer to the imaginary unit i . MATLAB works with complex numbers naturally:

```
x = sqrt(-1)    % x is complex
y = x + 2       % y is complex
z = x^2         % z is real
```

There is no built-in constant e . To calculate e^x , use `exp(x)`.

Try

Perform some calculations with built-in functions and constants.

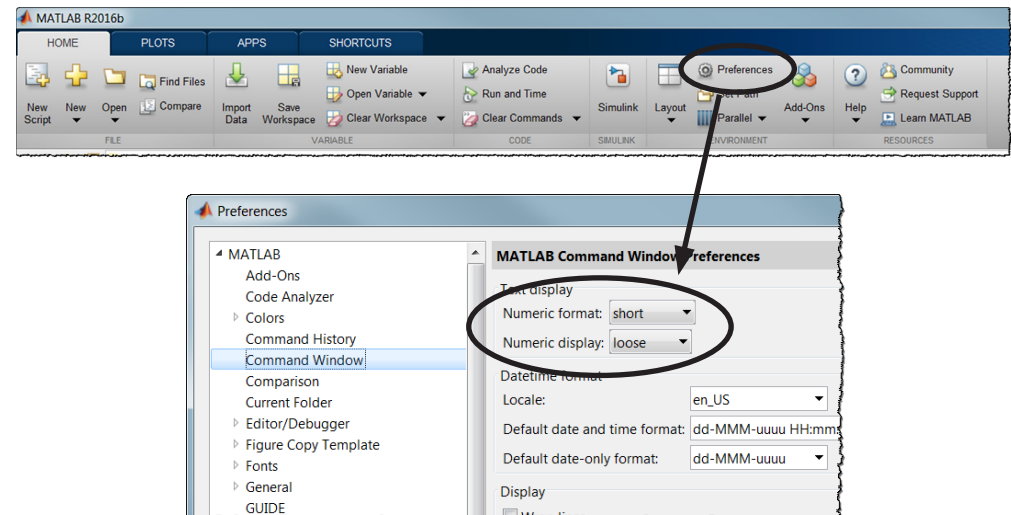
```
>> x = pi/2
>> y = cos(x)
>> z = exp(1i*x)
```

View the real and imaginary components of `z`.

```
>> real(z)
>> imag(z)
```

By default, numerical data is stored and manipulated in memory in double precision, which provides approximately 16 digits of precision. How the data is stored is independent of how it is displayed in the Command Window. By default, five digits of precision are displayed.

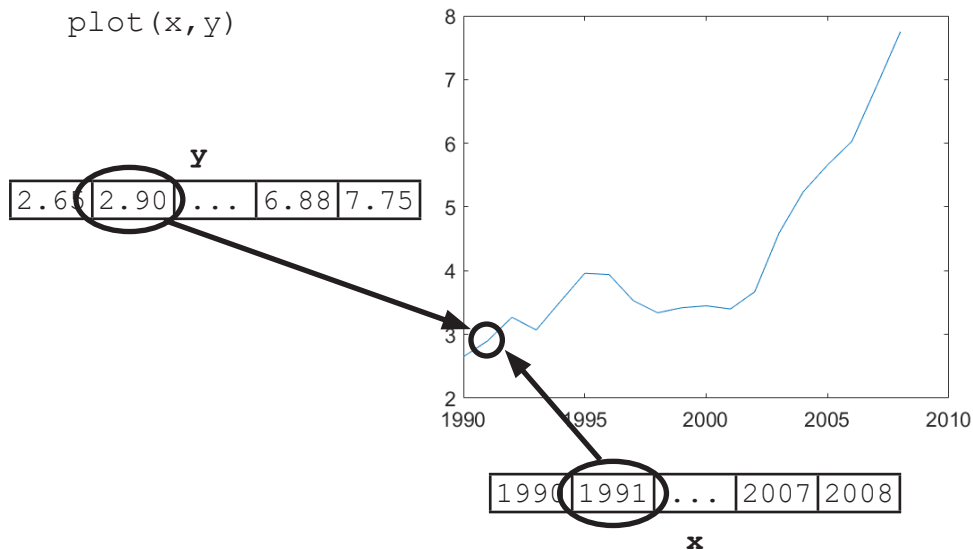
To control the display, you can use the `format` command, or click the **Preferences** button in the **Environment** section of the **Home** tab of the toolstrip. Various display options, including **Numeric Format**, appear under the **Command Window** node of the Preferences dialog.



Plotting

Two vectors of the same length can be plotted against each other with the `plot` function:

```
plot(x,y)
```



This plots each element of `x` on the horizontal axis against the corresponding element of `y` on the vertical axis. Neither vector is sorted before plotting.

The default appearance of the plot is to join the data points with straight line segments and not to use a visible marker at the data points. These behaviors, and the color of the line, can be changed with optional arguments to `plot` (see page 3-10).

If the input vectors `x` and `y` are not the same length, an error will occur. Any NaN values are simply omitted from the plot.

Try

Plot the prices in time.

```
>> plot(Year,Australia)
```

```
>> plot(Year,Germany)
```

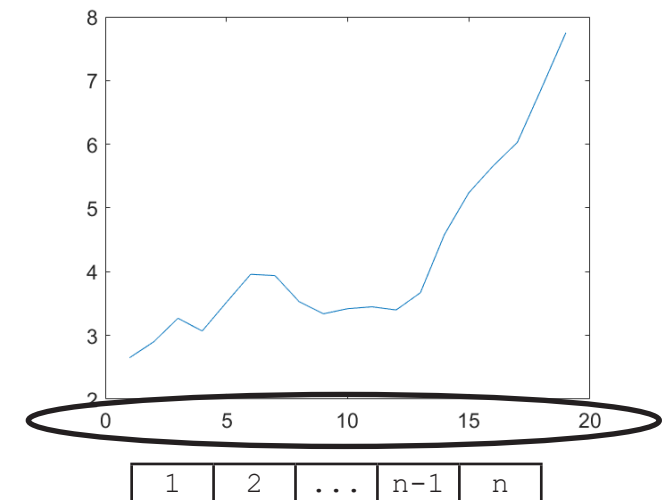
Attempt to make a scatter plot.

```
>> plot(Australia,Germany)
```

Is this what you wanted?

If only a single input vector is passed to `plot`, the vector is plotted on the vertical axis, using the array index as the horizontal axis.

```
plot(y)
```



(where `n` is the number of elements of `y`)

If a figure window is already open, the graph is plotted in that window, overwriting any previous plot. If no figure window is currently open, a new one is created.

Plot Options

The `plot` function accepts an optional third argument that allows you to specify the basic properties of the plot's appearance: line style, line color, and marker style.

These options are set using a character code. This code can consist of one, two, or three characters – one for each property.

Color	Marker	Line Style
b blue	. point	- solid
g green	o circle	-- dashed
r red	x (diagonal) cross	: dotted
c cyan	+ (vertical) cross	-. dash-dot
m magenta	* star	Note If a marker style is given, but no line style is given, no line will be plotted.
y yellow	s square	
k black	d diamond	
w white	v triangle (down)	
	^ triangle (up)	
	< triangle (left)	
	> triangle (right)	
	p pentagram	
	h hexagram	

Try

Make a scatter plot of German prices versus Australian prices.

```
>> plot(Australia,Germany,'mo')
```

Plot Australian prices with a line and point marker.

```
>> plot(Year,Australia,'.-')
```

The characters can be combined in any order, so

```
plot(Year,Mexico,'k^:')
```

and

```
plot(Year,Mexico,'^:k')
```

both plot Mexican prices with a black dotted line and triangular markers.

Obtaining Help

MATLAB help and documentation can show you

- The various ways to call a function
- The algorithm implemented by a function
- Examples of how to use a function
- Links to related functions
- Tutorials and background information

To display help on a particular function to the Command Window, type

```
help plot
```

For complete documentation on the function, type

```
doc plot
```

The MATLAB Help browser opens to the appropriate documentation. If you want to search or browse the documentation, enter

```
doc
```

to open the Help browser to its main page. You can then search by keywords or browse by topic.

Try

Find the various ways `plot` can be called.

```
>> doc plot
```

View the examples available in the documentation.

MATLAB → MATLAB Examples

Help is also available “on the fly”. You can highlight any command or function name and press **F1** to bring up a pop-up help window on that function.

Basic syntactic help appears automatically (after a short pause) after you enter the name of a function and an open parenthesis:

```
plot (
plot(X,Y)
plot(X,Y,LineSpec)
plot(X1,Y1,...,Xn,Yn)
plot(X1,Y1,LineSpec1,...,Xn,Yn,LineSpecn)
plot(Y)
plot(Y,LineSpec)
plot(____,Name,Value)
plot(ax,____)
plot(____)
More Help...
```

You can also bring up this syntax help by pressing **Ctrl+F1**. You can remove it by pressing **Esc**.

Creating Characters and Text

When MATLAB encounters alphanumeric characters in a command, it will attempt to interpret them as a variable name, function, command, or numeric value.

y = x → **variable called x**
y = 'x' → **character "x"**

Some functions expect text as input, such as plot annotation functions.

To specify characters as literal text, enclose them in single quotation marks:





```
a = 'MATLAB'
b = 'Simulink'
c = sin(pi/2)    % Numeric value (1)
d = 'sin(pi/2)' % Text "sin(pi/2)"
```

Since MATLAB treats all variables as arrays, a is actually an array of the letters “M”, “A”, “T”, “L”, “A”, and “B”. Hence, it is a row vector of size 1-by-6. Because it contains character data, it is of type char, not double.

Try

Compare how the symbol “Australia” is interpreted as a variable reference or literal text.

```
>> country = Australia
>> country = 'Australia'
```

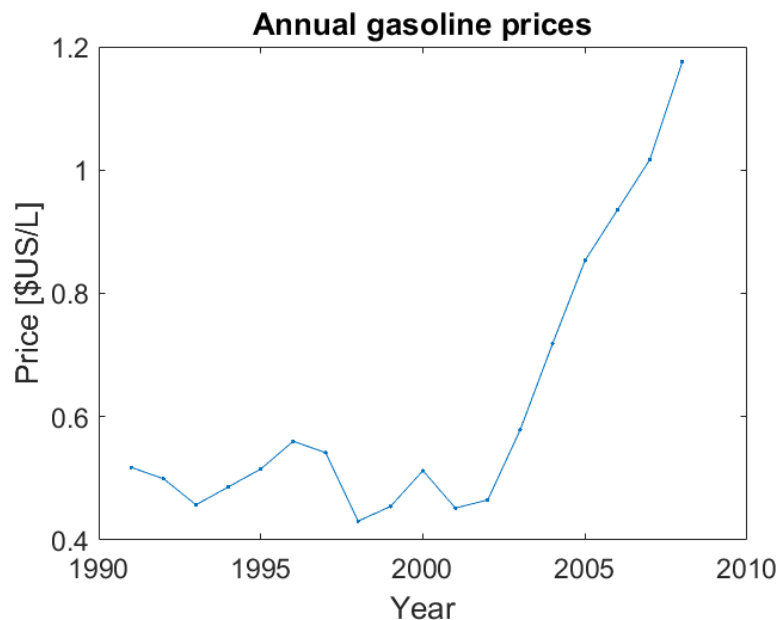
Workspace			
Name ^	Value	Size	Class
 a	'MATLAB'	1x6	char
 b	'Simulink'	1x8	char
 c	1	1x1	double
 d	'sin(pi/2)'	1x9	char

M	A	T	L	A	B
1	2	3	4	5	6

Annotating Plots

Textual information can be added to plots using separate annotation functions: `title`, `xlabel`, and `ylabel`. All functions take text as input:

```
plot(Year,Australia,'.-')
title('Annual gasoline prices')
xlabel('Year')
ylabel('Price [$US/L]')
```



MATLAB text annotation functions use basic TeX markup interpretation. Hence `'x^2'` will be rendered as x^2 , and `'x_2'` will be rendered as x_2 .

Try

Add axis labels and a title to your plot.

Other symbols can be created using markup commands, which start with a backslash. Greek letters, for example, are created by typing the full name of the letter, preceded by the backslash:

`xlabel('\gamma^2')` → γ^2

Curly braces (`{}`) are used as delimiters in TeX:

`title('\sigma \approx \pi^{-2}')`

→ $\sigma \approx \pi^{-2}$

`xlabel('{\bf German} prices')`

→ **German prices**

Because `\`, `^`, `_`, `{`, and `}` are all interpreted as markups by the TeX interpreter, use `\\`, `\^`, `_`, `\{`, and `\}` to render them as literal characters.

See A-11 for a list of basic TeX markup commands in MATLAB.

You can use the `grid` command to add grid lines to your plot:

```
grid on
grid off
```

The Command History

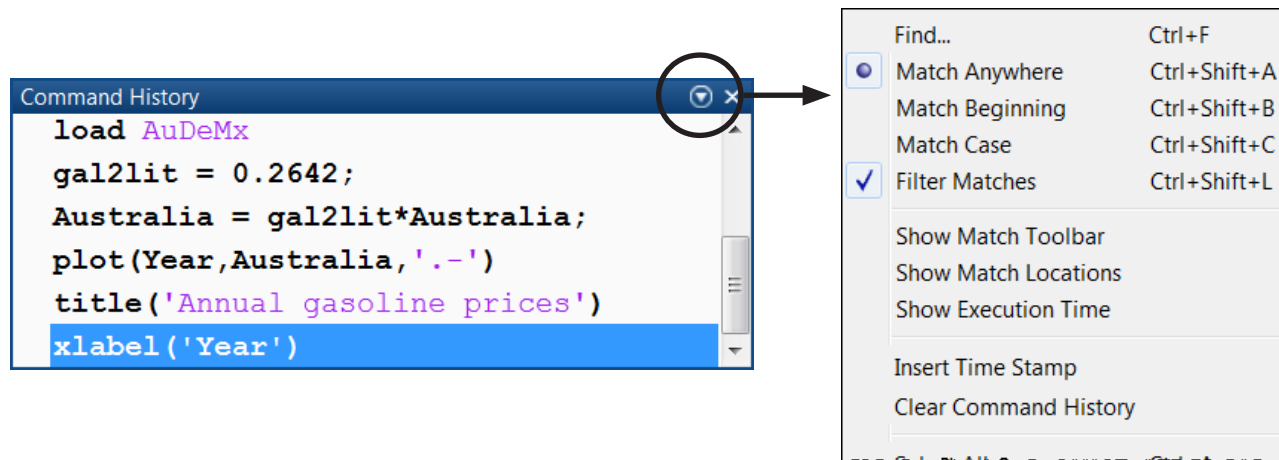
As the complexity of your tasks increases, repeatedly entering long sequences of commands in the Command Window becomes impractical. You can use the Command History to recall commands and help create scripts to automate your tasks.

MATLAB automatically keeps a history of the commands you type. These commands remain from session to session until you choose to delete them.

Pressing the up arrow key at the command prompt brings up the Command History window. In the Command History you can:

- Scroll through commands with the up and down arrow keys.
- Select commands with the mouse.
- Select multiple commands (use **Shift**-click or **Ctrl**-click).
- Edit recalled commands.
- Reenter commands.

If you partially enter a command, the Command History will be filtered to show only commands that match the portion entered. Use the window action menu to adjust the filtering options.



Try

Use the Command History to enter the sequence of commands below to plot annual gasoline prices in liters in Germany. Recall and edit commands as necessary.

```
>> load AuDeMx

>> gal2lit = 0.2642;
>> Germany = gal2lit*Germany;

>> plot(Year,Germany,'.-')
>> title('Annual gas prices in Germany')
>> xlabel('Year')
>> ylabel('Price [$US/L]')
```

The MATLAB® Live Editor

After selecting commands in the Command History, right-clicking displays a context menu. When you choose **Create Live Script** from the context menu, the selected commands are copied into a new live script file in the MATLAB Live Editor.

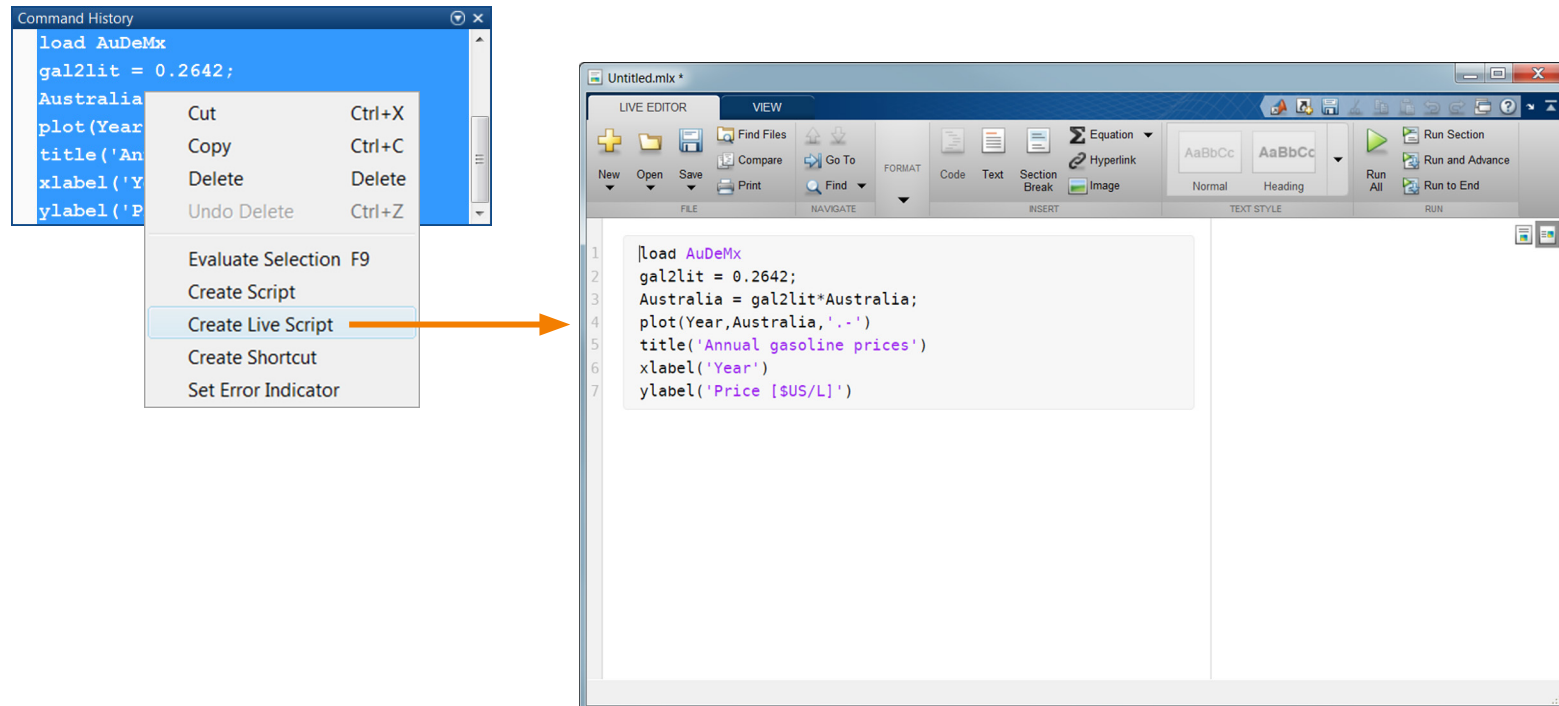
The Live Editor is used to write, edit, run, and share MATLAB live scripts, which are sequences of MATLAB commands together with embedded output, formatted text, equations, and images. Live scripts have a `.mlx` extension.

You can open the Live Editor and begin a new file by selecting **New → Live Script** in the **File** section of the **Home** tab on the toolbar. To open an existing live script in the Live Editor, simply double-click the file in the Current Folder browser.

Try

In the Command History, select the commands needed to plot the annual gas prices in liters in Australia, right-click, and choose **Create Live Script** from the menu.

The Live Editor window has its own toolbar. If the Live Editor is docked, the toolbar appears as extra tabs in the main MATLAB toolbar. The **Live Editor** tab contains controls for editing, navigating, running, and formatting your code.



Live Scripts

The sequence of commands you have just created in the Live Editor is a live script. When you run a live script, MATLAB executes the commands as a batch, in order. Any visible outputs from the commands are embedded directly in the live script.

Scripts are used to automate useful blocks of commands. Scripts do not have inputs and outputs. They operate on data in the workspace, and they can create new data on which to operate. Variables they create remain in the workspace and can be used in subsequent computations. Scripts can also produce graphical output by calling appropriate plotting functions.

Live scripts display output with the line of code that creates it. By default, MATLAB displays the output to the right of the code. When you select an output, the cursor automatically moves to the line of code that generated it.

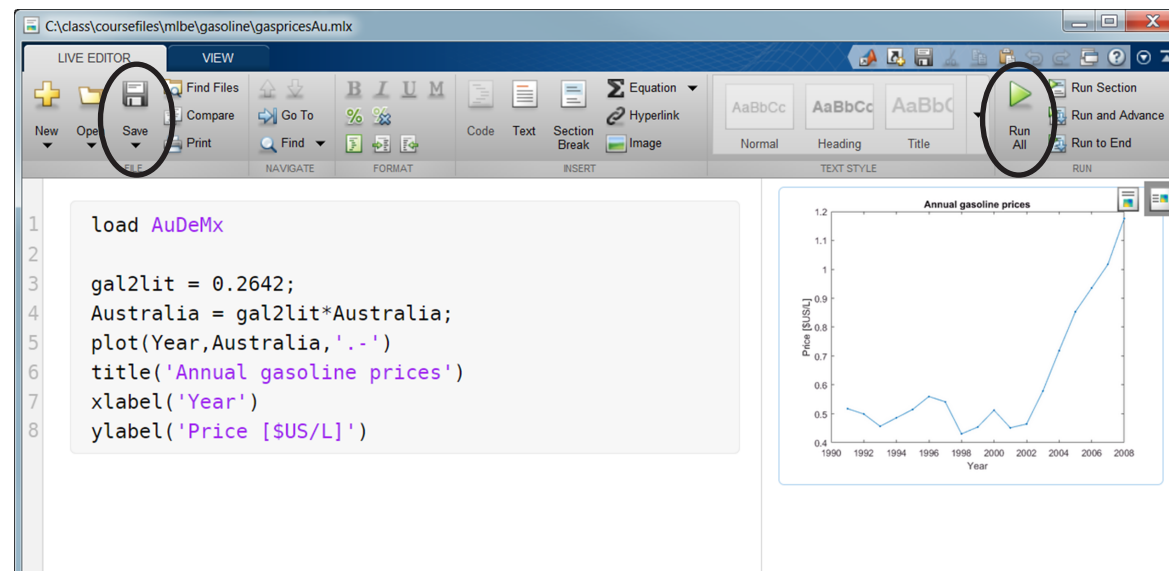
Try

Click **Run All** to run your live script that plots the gasoline prices in Australia in \$US/L. Save your live script. Close and reopen your script to verify the plot also saved.

If a script is displayed in the Live Editor, you can run it by clicking the **Run All** button in the **Run** section of the **Live Editor** tab on the toolbar. Equivalently, you can press **F5**. Running a live script runs the code currently shown in the Live Editor, even if the file contains unsaved changes.

When you save a live script, MATLAB automatically saves it with a `.mlx` extension. The file name in the Live Editor title bar is appended with an asterisk if the file has been modified.

Saving a live script also saves all displayed output. To clear the output in a live script, click **Clear all Output** in the **View** tab.



Adding Plots

By default, each `plot` command displays its own figure output. To change this behavior, so that you can plot multiple plots on the same axes, you can issue the `hold` command:

```
hold on
```

All subsequent plots will be added to the current axes until the command

```
hold off
```

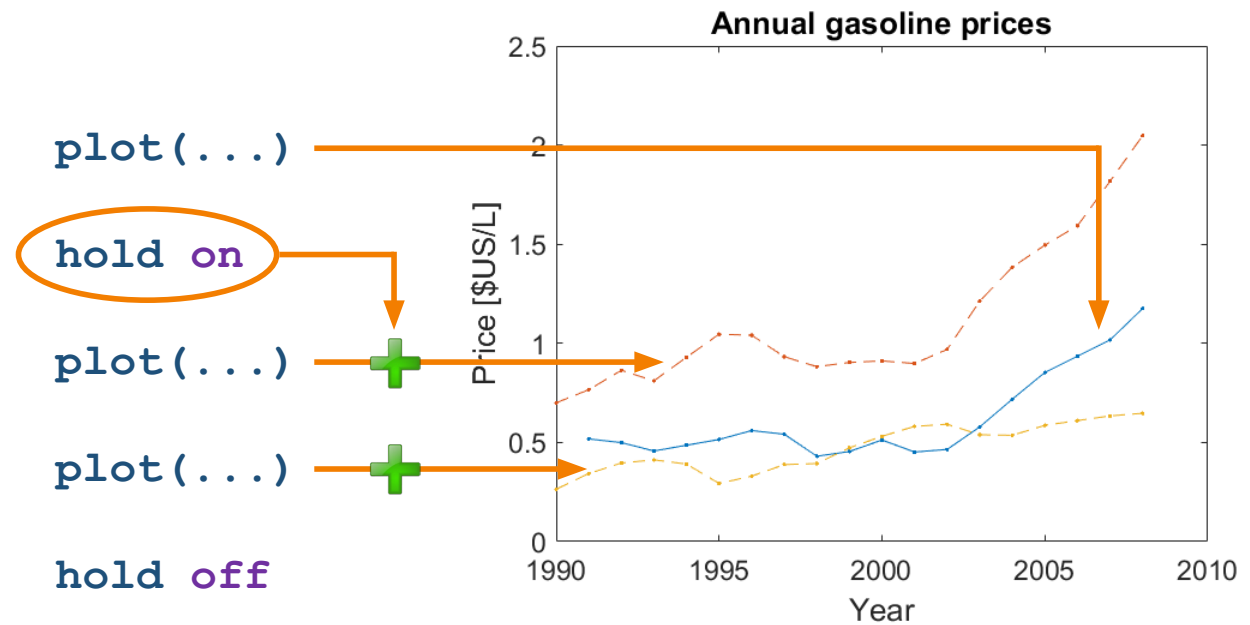
is issued.

Try

In your script, add other countries to your plot.

```
Germany = gal2lit*Germany;  
Mexico = gal2lit*Mexico;
```

```
hold on  
plot(Year,Germany,'.-')  
plot(Year,Mexico,'.-')  
hold off
```



Code Sections

The overall structure of most scripts naturally divides into sections. Especially for larger files, you may want to focus on one section at a time. To facilitate this, the Live Editor allows you to create code sections in your file.

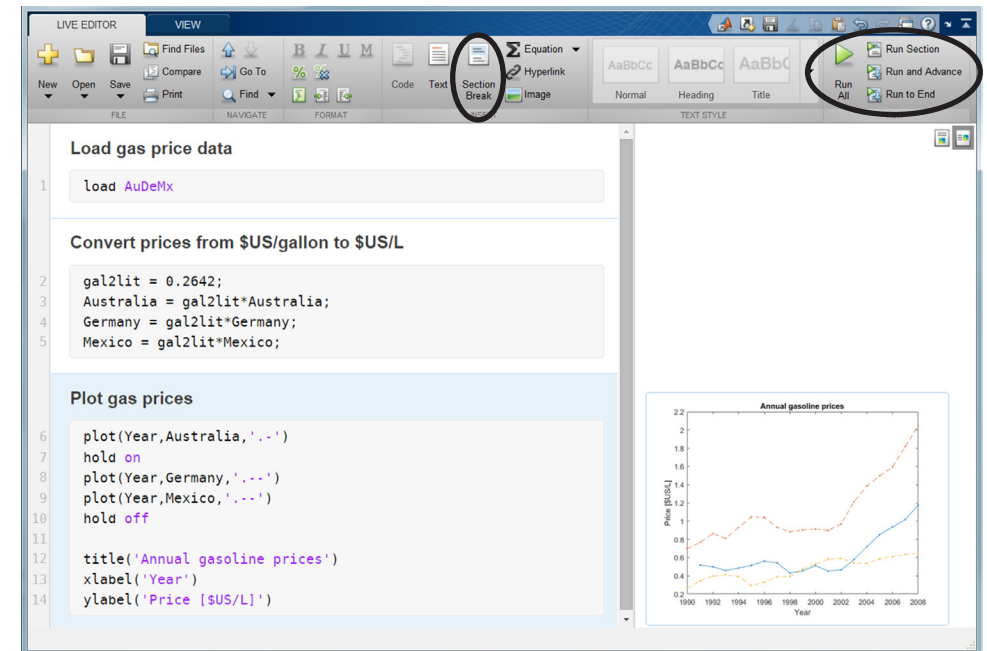
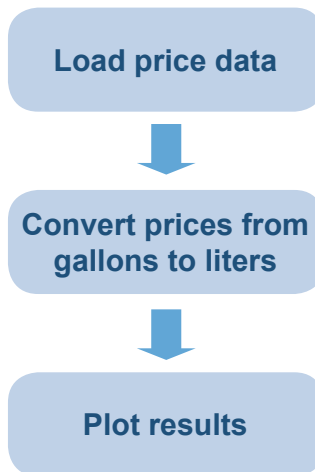
A section consists of a section break and the lines that follow, up to the start of the next section, which is identified by another section break. You can create a section break by clicking the **Section Break** button in the **Insert** section of the **Live Editor** tab on the toolbar, or by pressing **Ctrl+Alt+Enter**.

You can use the controls on the **Live Editor** tab of the toolbar to navigate from section to section and evaluate the code in a section.

Sections allow you to modify and reevaluate the code in that section to see how the changes affect the output. You do not need to commit to the change or rerun the entire file. This enables you to experiment rapidly with your code.

Try

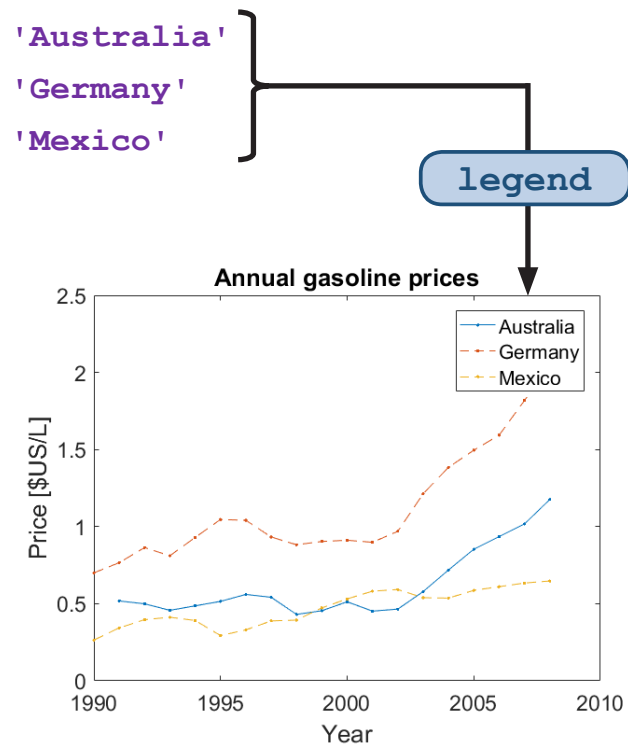
Reorganize your script to group similar commands together and insert section breaks. Step through the sections.



Adding A Plot Legend

You can use text inputs to the legend function to identify the components of a plot with multiple elements. Each input will be identified with each plot, in order.

```
legend('Australia','Germany','Mexico')
```



Try

Add a legend to your plot.

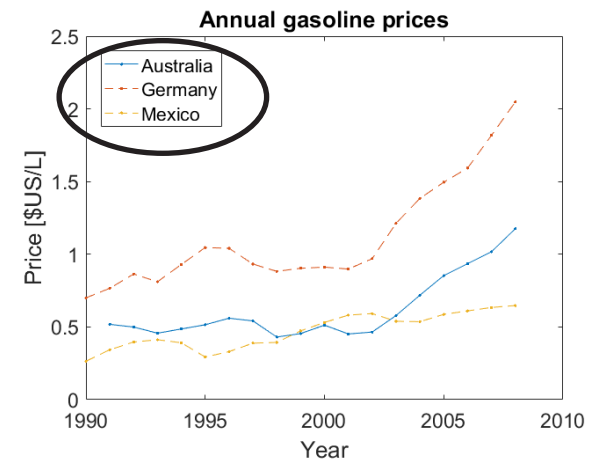
```
legend('Australia','Germany','Mexico')
```

Change the location of the legend.

```
legend('Australia','Germany','Mexico',...  
      'Location','northwest')
```

The legend function also accepts two optional arguments: the keyword 'Location' and a text description of the location, such as 'north' or 'southeast':

```
legend('Australia','Germany','Mexico',...  
      'Location','northwest')
```



Providing Documentation

Code files should contain nonexecutable text and other supporting items that explain the code. The Live Editor lets you add formatted text, hyperlinks, images, and equations to enhance the readability of your MATLAB code and create a presentable, interactive document to share with others.

You can add nonexecutable text in a live script by clicking **Text** in the **Insert** section of the **Live Editor** tab, or by pressing **Alt+Enter**.

Formatting text in a live script is similar to formatting text in a word processor. Use the controls in the **Live Editor** tab make text bold, italic, underlined, or monospaced, or to add headings, titles, and bulleted or numbered lists.

The **Insert** section of the **Live Editor** tab also contains buttons to insert equations, images, and hyperlinks. When you insert one of these items, MATLAB places the item on a new text line directly below the selected line.

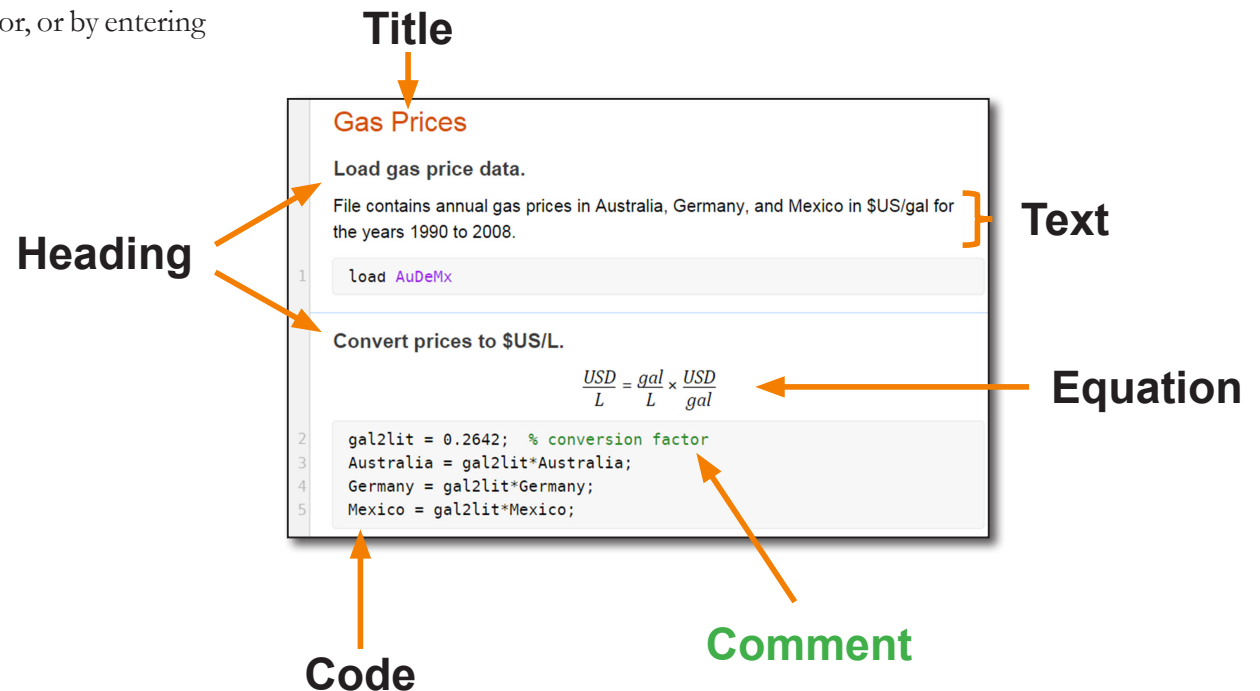
Equations can be inserted by using an interactive equation editor, or by entering a LaTeX equation. Only text lines can contain equations.

Try

Add a title, headers, and text to your code.

You may want to include text within block of code so you can keep related lines of code together in a single code block. Text following a percent sign (%) is a *comment*, or nonexecutable code. Comments can appear on lines by themselves, or they can be appended to the end of a line of code.

```
plot(x,y,'o')    % make scatter plot
% Add annotations
title('My plot')
```



Summary

- Entering commands
- Creating numeric variables
- Creating character variables
- Making and annotating plots
- Getting help
- Creating and running scripts
- Formatting live scripts

Try

View and run the live script `gasprices.mlx` that creates an annotated plot of annual gas prices in Australia, Germany, and Mexico.

Function	Use
<code>=</code>	Assign value to a variable
<code>()</code>	Function inputs Order of operations
<code>' '</code>	Literal text
<code>%</code>	Comment code
<code>clc</code>	Clear the Command Window
<code>home</code>	Move prompt to the top of the Command Window
<code>save</code> <code>load</code>	Save and load data to/from MAT-files
<code>clear</code>	Clear variables from workspace
<code>plot</code>	Create 2-D line plot
<code>xlabel</code> <code>ylabel</code> <code>title</code> <code>legend</code>	Annotate plot
<code>hold</code>	Keep/replace existing plot
<code>help</code> <code>doc</code>	Open documentation

Test Your Knowledge

Name: _____

1. (Select all that apply) Which of the following will create a scatter plot of `frogs` on the horizontal axis and `GDP` on the vertical axis, with red markers at the data points?
 - A. `plot(GDP,frogs,'ro')`
 - B. `plot(GDP,frogs,'o','r')`
 - C. `plot(frogs,GDP,'ro')`
 - D. `plot(frogs,GDP,'red')`

2. T/F: Script files can access and modify any variables already in the base MATLAB workspace.