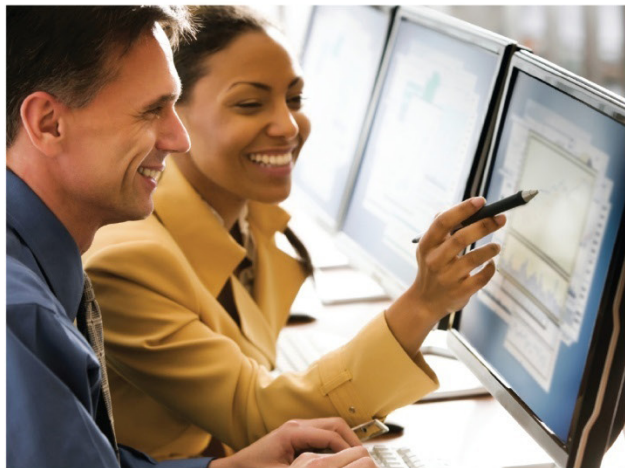


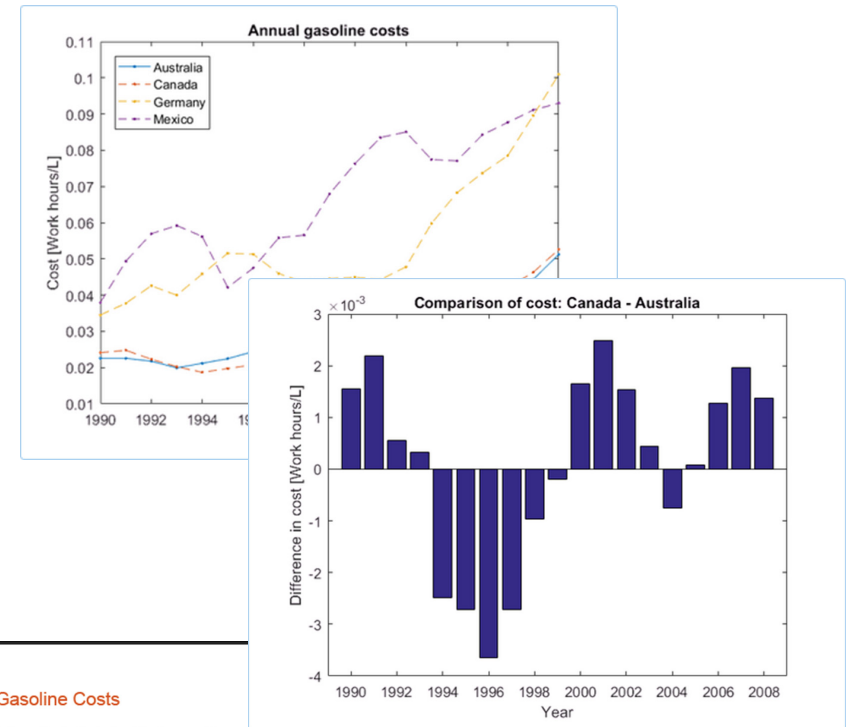
Analysis and Visualization with Vectors

MATLAB® Fundamentals for Aerospace Applications



Outline

- Calculations with vectors
- Creating vector variables
- Accessing and manipulating elements of vectors
- Sharing live scripts



Comparing Gasoline Costs

Compares the cost of gasoline in Australia with other countries.

The true cost (in terms of hours of labor) of a liter of gasoline is calculated from raw price data and average hourly wages. The cost in Australia is compared to Canada, Germany, and Mexico. Statistics are calculated for the difference between Australia and Canada.

Load gas price data from file

Replace the missing Australian price in 1990 with the price in 1991.

```
load gCosts
Australia(1) = Australia(2);
```

Plot the gas prices for all four countries.

```
plot(Year,Australia,'-')
hold on
plot(Year,Canada,'-')
plot(Year,Germany,'-')
plot(Year,Mexico,'-')
hold off
% Add annotations
legend('country','Location','northwest')
title('Annual gasoline prices')
xlabel('Year')
```

Chapter Learning Outcomes

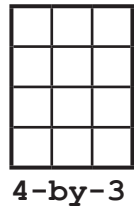
The attendee will be able to:

- Create vector variables, and apply functions and array operations to vectors.
- Access and manipulate the data stored in vectors.
- Share scripts in a variety of file formats.

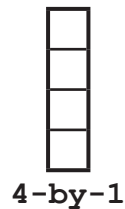
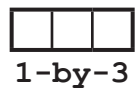
Vectors, Matrices, and Arrays

The name MATLAB® is an abbreviation of “MATrix LABoratory”. MATLAB was designed to work with matrices as the fundamental unit of data. A *matrix* is a regular two-dimensional grid, or table, of numbers.

A table of numbers with m rows and n columns is referred to as an m -by- n matrix.



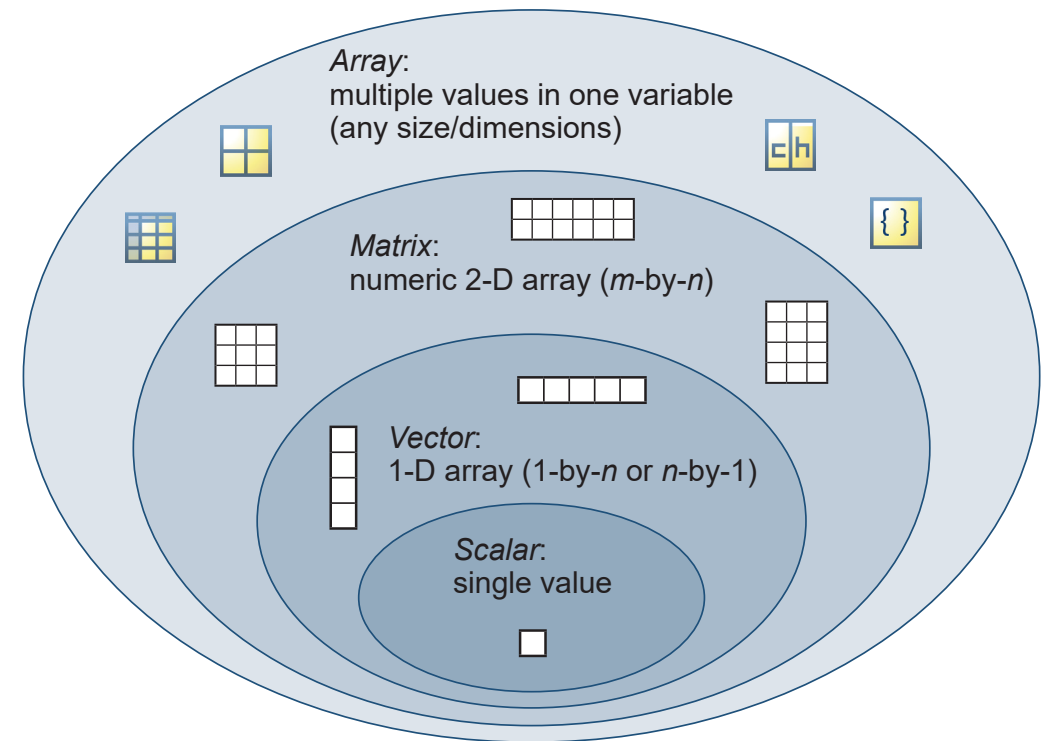
A *vector* is a single row or a single column of numbers. It is therefore a special case of a matrix, where either m or n is equal to 1. MATLAB distinguishes between a *row vector* (a single row) and a *column vector* (a single column). Hence, the dimensions of an n -element vector are given as either 1-by- n or n -by-1.



A single number is referred to as a *scalar*. It is a 1-by-1 matrix and, equivalently, a 1-element vector.



In general, a variable that contains multiple values in a regular n -dimensional arrangement is called an *array*. Scalars, vectors, and matrices are therefore (numeric) arrays. By default, variables in MATLAB are matrices. However, MATLAB does not restrict the size or dimension of variables.



Course Example: Comparing Real Cost

The gasoline price data is in \$U.S. per gallon. The value of \$1, however, differs from one country to another. Therefore, to make truly fair comparisons between gasoline prices in different countries, the prices should be expressed in terms of cost, such as the amount of work required to earn a given quantity of gasoline.

Dividing the price (in \$U.S./unit volume) by the average hourly wage (in \$U.S./hour) results in the cost of gasoline expressed as hours of labor per unit volume.

Once the cost is expressed in comparable units, you can perform calculations to compare countries, such as finding the average difference in cost through time.

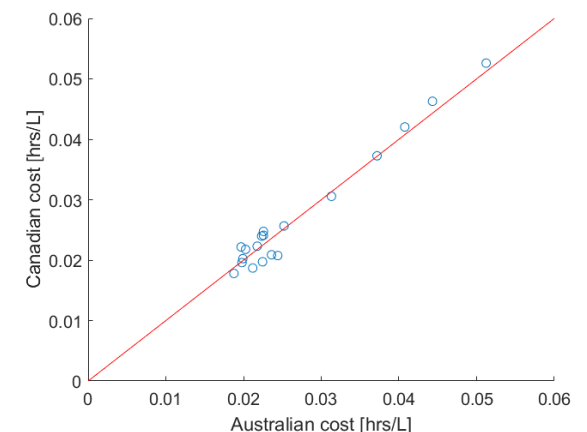
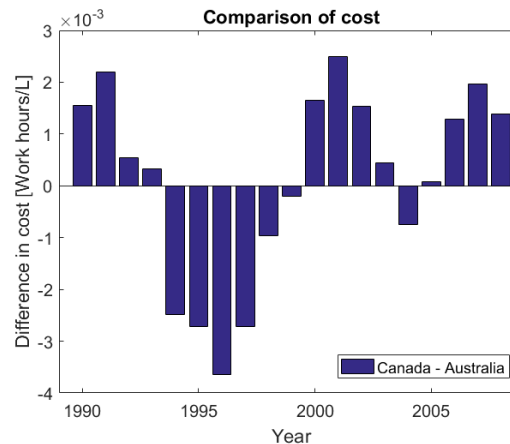
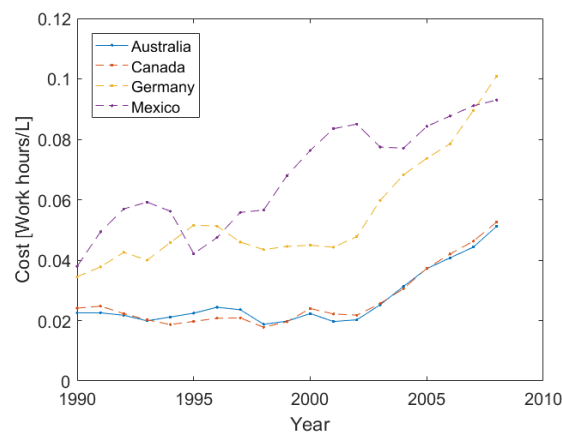
The file `gCosts.mat` contains the price data for Australia, Canada, Germany, and Mexico, as well as average hourly wage data.

Try

Create a script to load and plot gasoline price data.

```
load gCosts

plot(Year,Australia,'.-')
hold on
plot(Year,Canada,'.--')
plot(Year,Germany,'.--')
plot(Year,Mexico,'.--')
hold off
legend(country,'Location','northwest')
title('Annual gasoline prices')
xlabel('Year')
ylabel('Price [US$/gallon]')
```



Array Operations

MATLAB is designed to work with matrices and vectors as mathematical objects. Array addition and subtraction is defined, both mathematically and in MATLAB, as an element-by-element operation:

$$z = x + y$$

1
3
5
7

 $+$

0
2
1
2

 $=$

1
5
6
9

x **y** **z**

Similarly, scalar multiplication or division of an array is defined mathematically:

$$y = 2 * x$$

1
3
5
7

 \times

2

 $=$

2
6
10
14

x **y**

Although not strictly defined mathematically, MATLAB performs scalar *expansion* to interpret expressions such as

$$y = x + 2$$

In this case, y is the result of adding 2 to each element of x .

1
3
5
7

 $+$

2

 $=$

1
3
5
7

 $+$

2
2
2
2

 $=$

3
5
7
9

x **y**

Try

Create and run a section in your script to convert the gasoline prices to comparable units – work hours (based on average income) per liter.

```
gal2lit = 0.2642;
Australia = gal2lit*Australia/hourlyAus;
Canada = gal2lit*Canada/hourlyCan;
Germany = gal2lit*Germany/hourlyGer;
Mexico = gal2lit*Mexico/hourlyMex;
```

Calculate the difference between Australian and Canadian cost.

```
AuCaDiff = Canada - Australia;
```

Calculate the average of Canadian and Mexican cost.

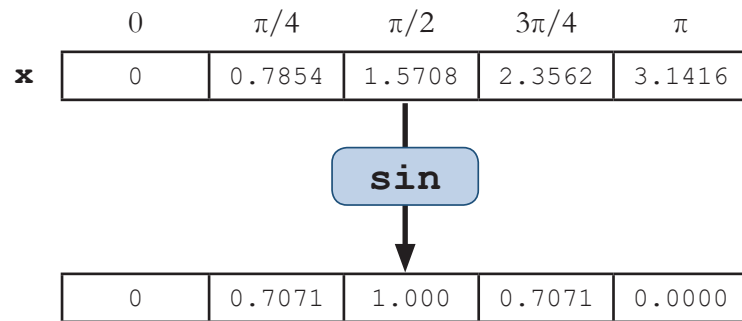
```
CaMxAvg = (Canada + Mexico)/2;
```

Mathematical Functions

One of the principal advantages of working in MATLAB is that commands are *vectorized*. They perform mathematical operations on an entire vector or array of values in a single command. Most mathematical operations in MATLAB are vectorized.

For example, to compute the sine of a sequence of values, first create a vector containing the sequence of values, and then pass the entire vector to the `sin` function:

```
y = sin(x)
```



MATLAB performs the five calculations in a single command.

Note that array operations such as scalar multiplication are also vectorized operations.

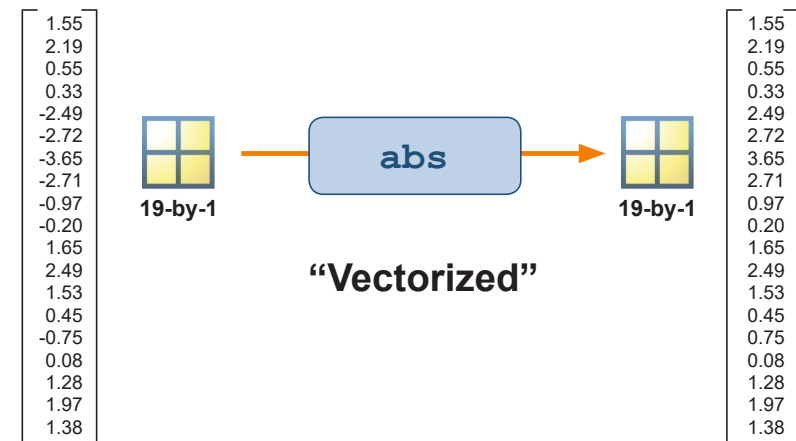
Try

Calculate the absolute value of the difference between Australian and Canadian cost.

```
absDiff = abs(AuCaDiff);
```

View the elementary math functions.

```
doc elfun
```



sin	sqrt
sind	nthroot
sinh	abs
asin	angle
exp	floor
log	ceil
log2	round
log10	mod

Statistical Functions

Basic statistical functions in MATLAB can be applied to a vector of data to produce a single output:

```
maxCan = max(Canada)
```

Some of these functions can return extra optional arguments. For example, suppose you want to find the year that had the lowest gasoline prices in Australia. The default behavior of the `min` function returns the smallest value, but not its location. However, the index (location) where the smallest value occurred can be returned as an optional argument:

```
[AusMin, AMIndex] = min(Australia)
```

```
AusMin = 1.6300  
AMIndex = 9
```

Note the use of the square brackets construction for multiple function outputs. The first output is the minimum value; the second is the index.

Try

Find the greatest cost difference between Australia and Canada.

```
biggestDiff = max(absDiff)
```

What was the average difference in gasoline cost between Australia and Canada?

```
avgDiff = mean(AuCaDiff)
```

Is the result what you expected? Modify the command to ignore NaNs.

```
avgDiff = mean(AuCaDiff, 'omitNaN')
```

View functions for data analysis.

```
doc datafun
```

```
[maxDiff,idx] = max(absDiff)
```

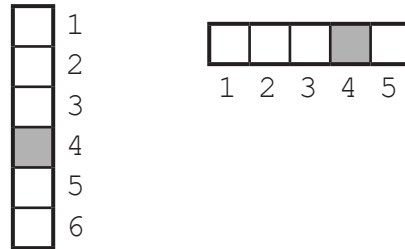
1.55	1
2.19	2
0.55	3
0.33	4
2.49	5
2.72	6
3.65	7
2.71	8
0.97	9
0.20	10
1.65	11
2.49	12
1.53	13
0.45	14
0.75	15
0.08	16
1.28	17
1.97	18
1.38	19

Indexing into Vectors

Data in arrays is accessed using indexing. Elements are indexed by number, using parentheses after the variable name:

```
y = x(4)
```

This command extracts the fourth element of `x` and assigns it to the (scalar) variable `y`.



Indices in MATLAB variables always begin with 1.

Where they end, of course, depends on the size of the variable. You can use the `length` function to determine the number of elements in a vector:

```
n = length(x)
y = x(n)
```

You can also use the MATLAB keyword `end` as an index:

```
y = x(end)      % last element
z = x(end-1)    % second-to-last element
```

Because parentheses are used for both indexing and function inputs, does `foo(n)` refer to the `n`th element of the variable `foo` or to the `foo` function with `n` as an input? MATLAB will first attempt to interpret such an expression as an index into a variable. If no such variable exists, the expression is then interpreted as a function call.

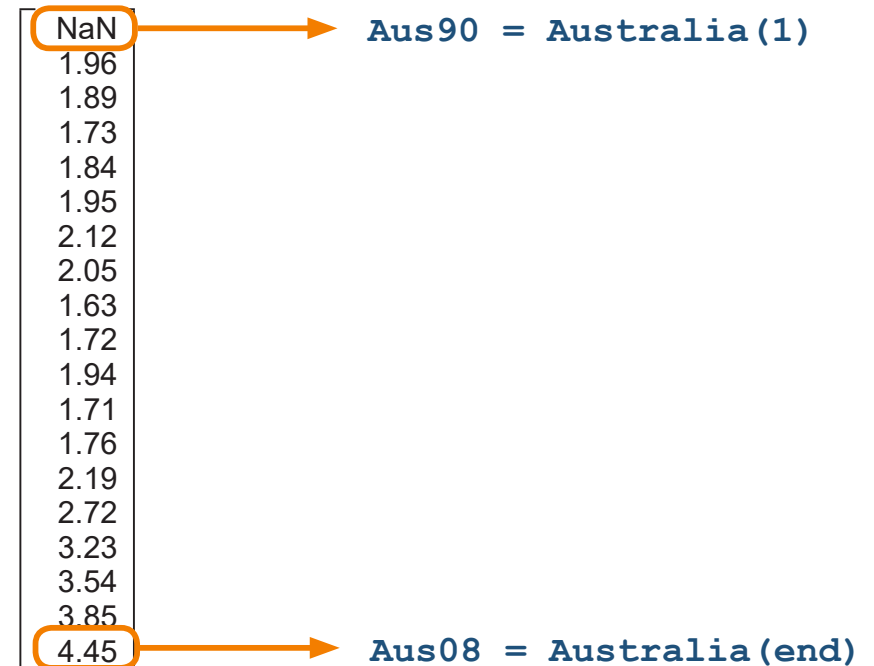
Try

Extract the missing Australian price.

```
Aus90 = Australia(1)
```

When did the greatest cost difference between Australia and Canada occur?

```
[biggestDiff,idx] = max(absDiff);
Year(idx)
```

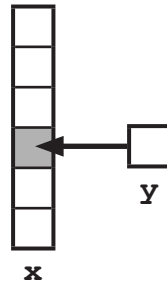


Changing Values in a Vector

Elements of a variable can be altered by combining indexing with assignment:

$$x(4) = y$$

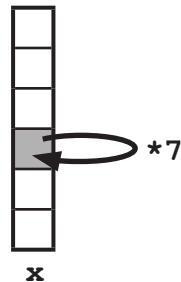
This is called an *indexed* (or subscripted) assignment, and it assigns the value y to the 4th element of x .



Referencing and assignment can be combined in a single statement:

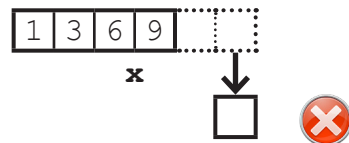
$$x(4) = 7 * x(4)$$

This replaces the 4th element of x with seven times its current value.

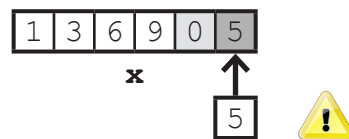


Although referencing an index beyond the bounds of an array will produce an error, assigning to such an index will not. Instead the array will be resized to make the assignment possible:

```
y = x(6)    % Error!
```



```
x(6) = 5    % no error, but x is resized!
```



Note that unassigned elements are given a default value of 0.

Try

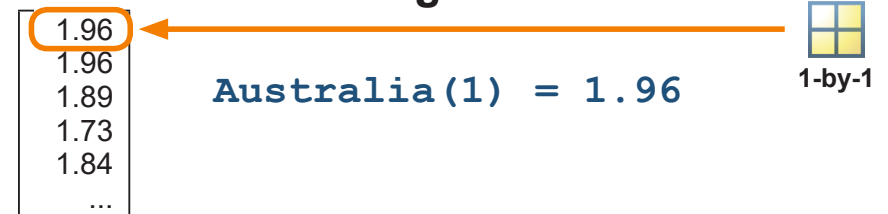
Replace the missing Australian cost in 1990 with the Australian cost in 1991.

```
Australia(1) = Australia(2);
```

reference



assignment



Entering Vectors Manually

You can create vectors and matrices in MATLAB by manually entering values. However, if the values follow a simple pattern, you can use MATLAB functions to create arrays.

Vectors of arbitrary data can be entered manually using the syntax:

```
x1 = [1,0,-42,pi,7]    % using commas
x2 = [1 0 -42 pi 7]    % using spaces
```

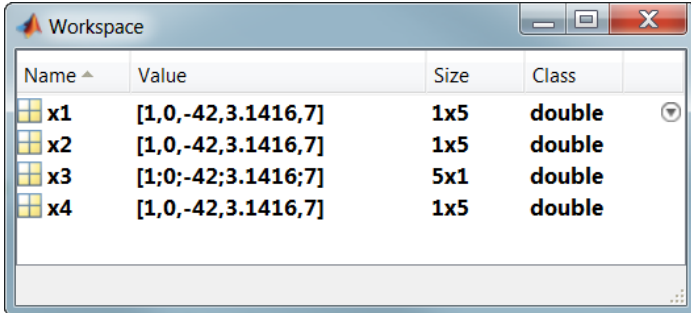
The square brackets concatenate the entries into an array. You can separate entries by either commas or spaces to create a row vector.

Separating entries by semicolons creates a column vector:

```
x3 = [1;0;-42;pi;7]
```

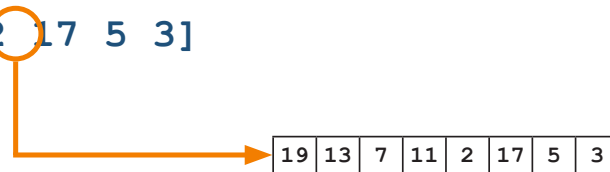
You can also transform a row vector into a column (or vice versa) using the transpose operator (`.'`) or the complex conjugate transpose operator (`'`):

```
x4 = x3'
```

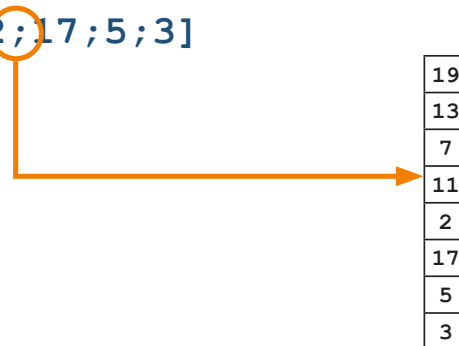


Name	Value	Size	Class
x1	[1,0,-42,3.1416,7]	1x5	double
x2	[1,0,-42,3.1416,7]	1x5	double
x3	[1;-42;3.1416;7]	5x1	double
x4	[1,0,-42,3.1416,7]	1x5	double

x = [19 13 7 11 2 17 5 3]

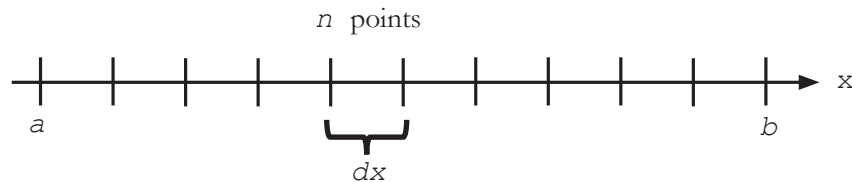


x = [19;13;7;11;2;17;5;3]



Creating Vectors of Equally Spaced Values

There are two basic methods for creating vectors of equally spaced values from a to b . The method you choose depends on the information available.



If you know the spacing, dx , between the values, use the range operator (`:`):

```
x = a:dx:b
```

If dx is not specified, it is assumed to be 1:

```
x = 2:2:8
y = 2:8
```

The values of x will be a , $a+dx$, $a+2dx$, and so on, until the largest value of $a+mdx$ that is not greater than b .

```
z = 1:pi
```

If $a+dx$ is greater than b , the result will consist of the single value a . If a itself is greater than b , the output will be an empty array (unless dx is negative).

Name	Value	Size
x	[2,4,6,8]	1x4
y	[2,3,4,5,6,7,8]	1x7
z	[1,2,3]	1x3

Try

Create a vector of years using the colon operator.

```
yrs = 1990:2008
```

Create a column vector of years.

```
yrs = (1990:2008)'
```

If you know the number of values n , use the `linspace` function:

```
x = linspace(a,b,n)
```

If n is not specified, it is assumed to be 100. The result will be an n -element row vector, including the two end values a and b :

```
u = linspace(3,6,4)
v = linspace(0,1,5)
w = linspace(pi,2*pi)
```

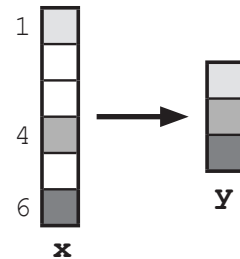
Name	Value	Size
u	[3,4,5,6]	1x4
v	[0,0.2500,0.5000,0.7500,1]	1x5
w	1x100 double	1x100

Note that the range (colon) operator and the `linspace` function both create row vectors.

Accessing Data in Vectors

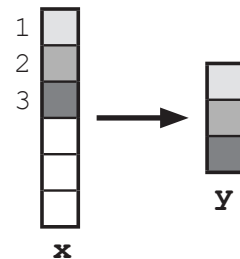
You can reference multiple elements of a vector using a vector of indices:

```
y = x([1 4 6])
```



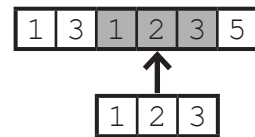
This is commonly done in conjunction with the range operator (:):

```
n = floor(length(x)/2)
y = x(1:n)
```



You can perform indexed assignment on multiple entries, in the same manner as multiple-index referencing:

```
x = [1 3 6 9 2 5]
y = 1:3
x(3:5) = y
```



Try

Extract the prices in Mexico up to 2000, and from 2000 on.

```
Mex90s = Mexico(1:10)
```

```
Mex00s = Mexico(11:end)
```

Find the average annual gasoline cost change in Mexico in the 1990s and the 2000s.

```
MexAvgChange90s = mean(diff(Mex90s))
```

```
MexAvgChange00s = mean(diff(Mex00s))
```

Which were the five cheapest years for gasoline in Germany?

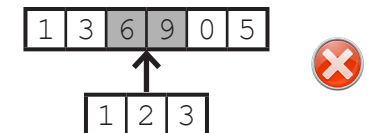
```
[sGer,sIdx] = sort(Germany);
```

```
sYear = Year(sIdx);
```

```
sYear(1:5)
```

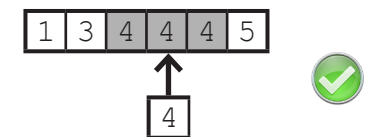
When performing multiple assignments in this way, the number of elements indexed on the left of the equals sign must match the number of elements in the expression on the right:

```
x = [1 3 6 9 2 5]
y = 1:3
x(3:4) = y % Error!
```



However, if the value on the right of the equals sign is a scalar, all the values referenced on the left will be assigned that same value:

```
x(3:5) = 4 % OK
```



Additional Vector Plot Types

MATLAB has many specialized plotting commands to allow you to visualize and present your data with the emphasis on the data features you find important. Some of the additional plot types for vector data include

<code>scatter</code>	Scatter plot, with variable marker size and color
<code>bar</code>	Bar graph (vertical and horizontal)
<code>stem</code>	Discrete sequence (signal) plot
<code>stairs</code>	Stairstep graph
<code>area</code>	Filled area plot
<code>pie</code>	Pie chart
<code>histogram</code>	Histogram

You can interactively create a new plot by selecting variables in the Workspace browser. You can then browse the available plot types in the **Plots** tab of the toolstrip. When you create a plot this way, MATLAB automatically generates the corresponding plot command and displays it in the Command Window.

Try

Visualize the difference between Australian and Canadian gasoline costs.

```
bar (Year,AuCaDiff)
```

Try some alternative visualizations.

```
stem (Year,AuCaDiff)
```

```
stairs (Year,AuCaDiff)
```

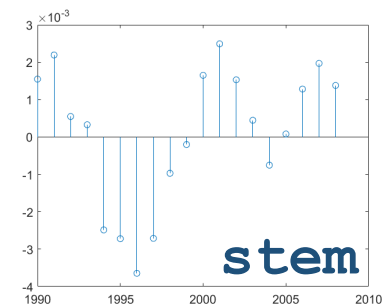
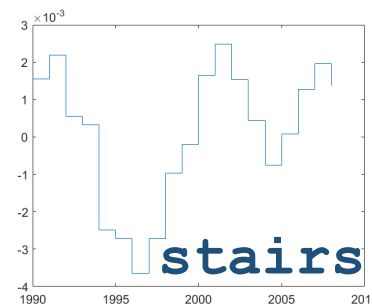
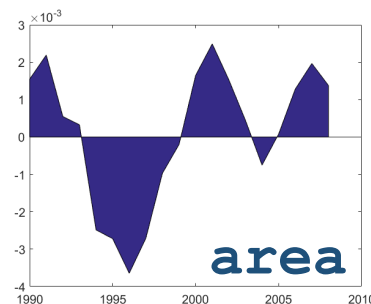
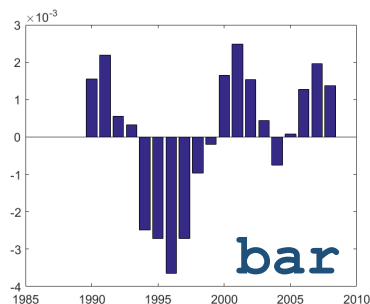
```
area (Year,AuCaDiff)
```

See the relationship between Australian and Canadian costs.

```
scatter (Australia,Canada)
```

Use the year to color the markers.

```
scatter (Australia,Canada,20,Year,'Filled')
```



Axis Control

MATLAB automatically chooses the limits and tick markings for plot axes. Although the automatic values are usually aesthetically reasonable, you may want more precise control of the axis appearance.

The `axis` function gives direct control of the appearance of the current figure's axes:

<code>v = axis</code>	Returns vector (v) of current limits
<code>axis(v)</code>	Sets limits to values specified in vector (v)
<code>axis tight</code>	Changes limits to the range of the data
<code>axis off</code>	Turns off all axes and background

The `x/y/zlim` functions allow you to change the limits of any individual axis without needing to use `axis` to specify all limits. They all take a 2-element vector as input, which is interpreted as the minimum and maximum axis values:

```
ylim([-42 pi]) % y axis will be from -42 to pi
```

When called with an output but no input, the `x/y/zlim` functions return the current axis limits.

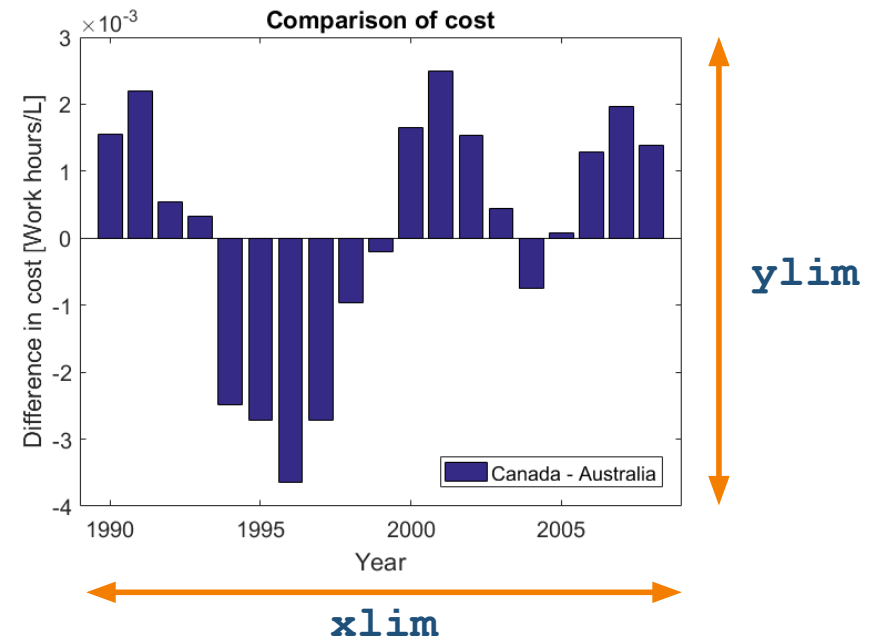
Try

Adjust the limits on the horizontal axis.

```
bar(Year,AuCaDiff)  
xlim([1989 2009])
```

Add a 45-degree line to a scatter plot.

```
scatter(Australia,Canada)  
xlim([0 0.06])  
AuLimits = xlim;  
hold on  
plot(AuLimits,AuLimits,'r')  
hold off
```

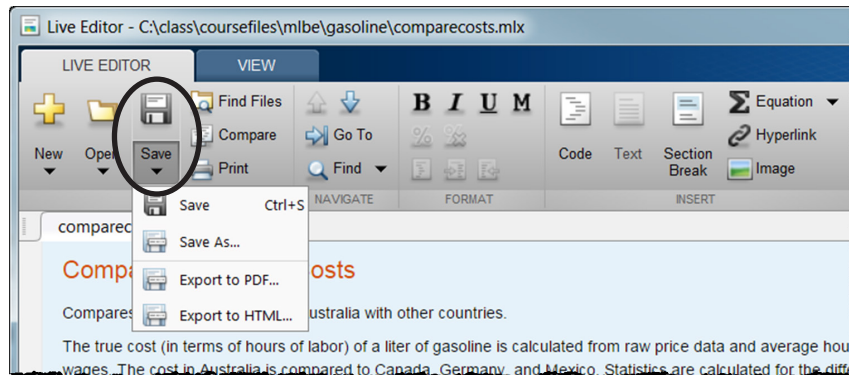


Sharing Live Scripts

You can share your live script and its results with other MATLAB users as an interactive document, or as a static file for viewing outside of MATLAB.

To share an interactive document with other MATLAB users, simply distribute the live script file. Recipients can open and view the file in the same state you saved it in, including generated output.

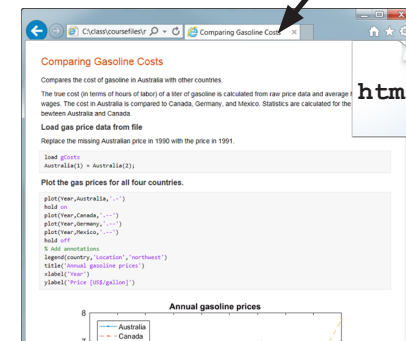
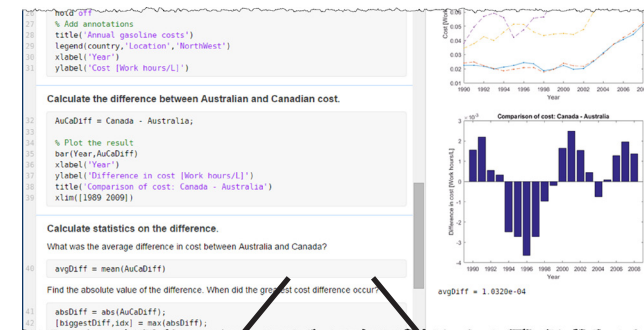
To share your live script as a static document that can be viewed outside of MATLAB, you can export the script to either PDF or HTML format. The exported file closely resembles your live script with the output viewed inline, including all code, comments, images, equations, and formatted text. To export your live script in one of these formats, select **Save** → **Export to PDF** or **Save** → **Export to HTML** on the **Live Editor** tab.



Try

Save `comparecosts.mlx` in various formats.

If you are sharing with users of older MATLAB versions without the Live Editor, you can save the live script as a plain text code (.m) file. To create a plain code file from a live script, open the live script and select **Save** → **Save As...** from the **File** section of the **Live Editor** tab. In the dialog box, select “MATLAB Code files (*.m)” as the **Save as type**. MATLAB converts formatted live script content to text and disregards images.



Summary

- Calculations with vectors
- Creating vector variables
- Accessing and manipulating elements of vectors
- Sharing live scripts

Function	Use
()	Index into array
[]	Multiple function outputs Create (numeric) array
' '	Transpose array
:	Range (step)
linspace	Create array of equally spaced values
figure close	Create and close figure windows
scatter bar stem stairs area pie histogram	Create various 2-D graphics
axis xlim ylim	Control plot axis limits

Test Your Knowledge

Name: _____

1. Given a vector `x`, what is the command to add 3 to each element, double that value, then sum all the resulting values?
 - A. `sum(2*x+3)`
 - B. `sum(2*[x(k)+3])`
 - C. `sum[2*x+3]`
 - D. `sum(2*(x+3))`

2. (Select all that apply) Which commands are equivalent to the command `x = 1.4:2:6.8`?
 - A. `x = [1.4 2 6.8]`
 - B. `x = [1.4 6.8]`
 - C. `x = [1.4 3.4 5.4]`
 - D. `x = [1.4 3.4 5.4 6.8]`
 - E. `x = [3.4 5.4]`

