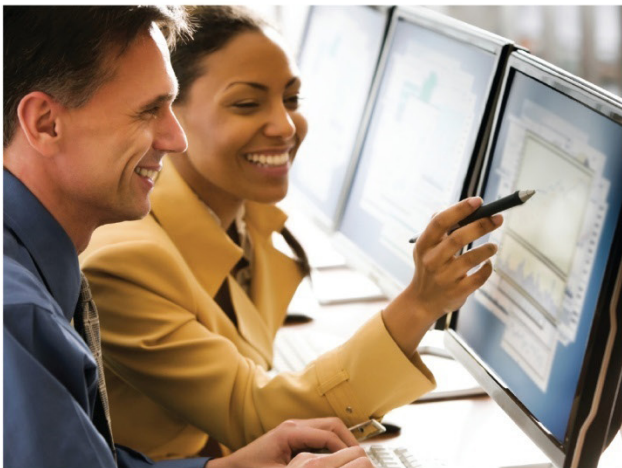


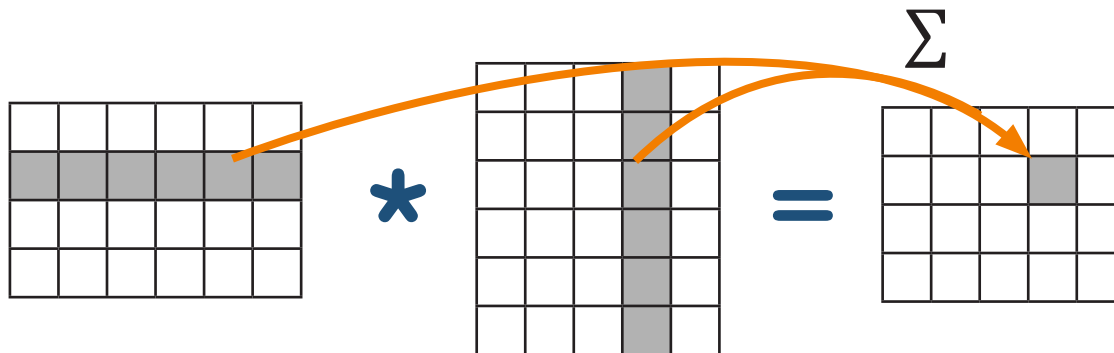
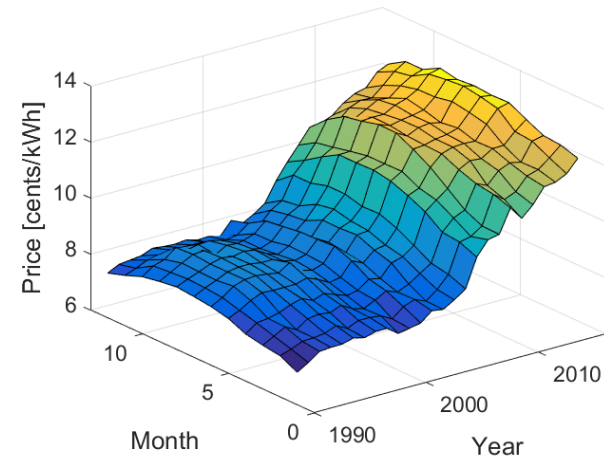
Analysis and Visualization with Matrices

MATLAB® Fundamentals for Aerospace Applications



Outline

- Creating and manipulating matrices
- Calculations with matrices
- Statistics with matrices of data
- Matrix visualization



Chapter Learning Outcomes

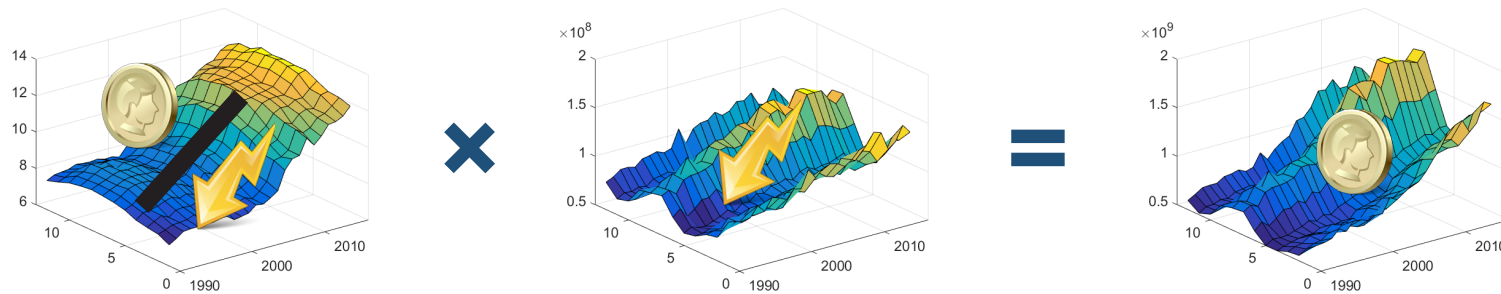
The attendee will be able to:

- Create matrix variables.
- Access and manipulate the data stored in matrices using row-column indexing.
- Create larger array variables by combining smaller elements and using matrix-creation functions.
- Perform matrix and array (element-wise) operations.
- Compute basic descriptive statistics for a matrix of data.
- Distinguish between the behavior of mathematical and statistical functions in MATLAB®.
- Plot columns of a matrix as independent variables.
- Create arrays of text.
- Visualize matrix data in two and three dimensions.

Course Example: Electricity Consumption

The file `electricity.mat` contains two 12-by-26 matrices, `usage` and `price`, that represent the amount of residential electricity (in MWh) sold in the U.S. and the retail price (in ¢) of a kWh of residential electricity, respectively. The 12 rows represent the 12 months of the year, and the 26 columns represent the 26 years from 1990 to 2015.

Multiplying each element of `usage` with the corresponding element of `price` results in the total revenue from the sale of electricity for each month from 1990 to 2015.



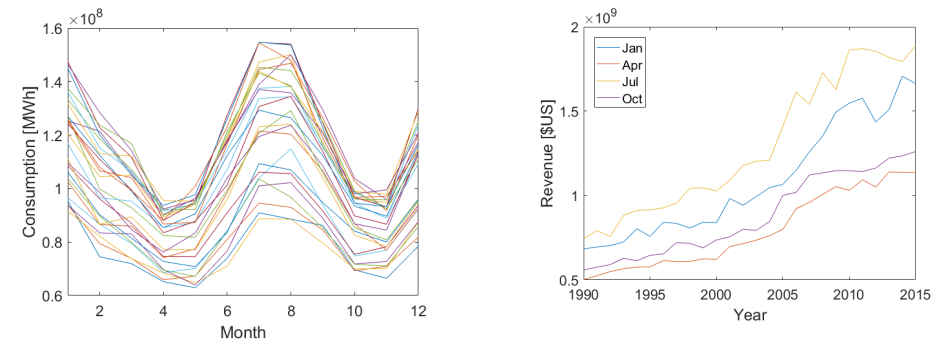
The goal of this example is to perform such calculations on matrix data and to use the matrix arrangement to investigate patterns and trends in the data.

Try

Load the electricity and price data.

```
load electricity  
whos
```

The data is available for download from the U.S. Energy Information Administration: <http://www.eia.gov/electricity/data.cfm>



Concatenating Arrays

Matrices of arbitrary data can be entered manually using the same syntax as for vectors. Use commas or spaces to separate columns (i.e., separate elements horizontally) and semicolons to separate rows (i.e., vertically):

```
A = [1 2 3; 4 5 6]
B = [1 2; 3 4; 5 6]
```


A	1	2	3
	4	5	6

B	1	2
	3	4
	5	6

All rows in a matrix must have the same number of columns (and vice versa). Empty spaces will result in an error:

```
C = [1 2 3; 4 5; 7 8 9] % No!
C = [1 2 3; 4 5 ; 7 8 9] % No!
```

1	2	3
4	5	
7	8	9



Instead, you can use the NaN (Not A Number) “value” as a placeholder for missing data:

```
C = [1 2 3; 4 5 NaN; 7 8 9]
```

1	2	3
4	5	NaN
7	8	9



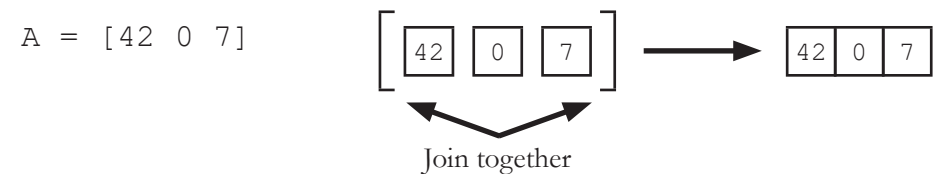
The square brackets ([]) are the concatenation operator in MATLAB. Concatenation is the act of joining arrays together to form larger arrays.

Try

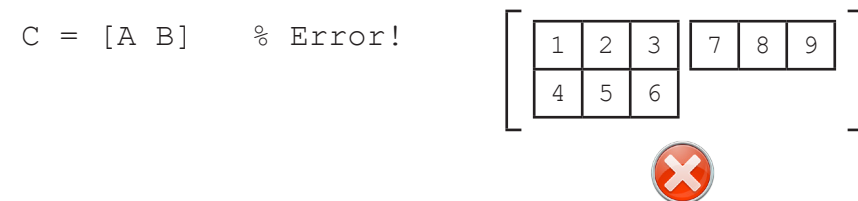
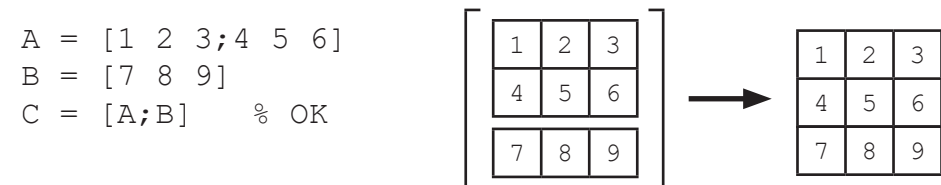
Create and concatenate some matrices.

```
A = [1,2,3; 4,5,6; 7,8,9];
B = [A; 10 11 12]
C = [A, 10;11;12] % error!
C = [A, [10;11;12]] % OK
```

When you create a vector or matrix with the square bracket syntax, you are actually concatenating the individual elements (which are scalars or 1-by-1 arrays) into a larger array:



You can use the same principles to join together any arrays, as long as the result has consistent dimensions (each row has the same number of columns, and vice versa). Use commas or spaces to separate elements horizontally and semicolons to separate elements vertically.



Creating Matrices with Functions

MATLAB has a large number of *matrix creation functions* that output variables with certain characteristics and avoid tedious data entry. These functions use the syntax

$$A = \text{function_name}(m,n)$$


or

$$A = \text{function_name}(n)$$

m

n


A


m-by-n

n

n

A


n-by-n

For example, the zeros function and the ones function create matrices with every element equal to 0 or 1, respectively:

$$X = \text{zeros}(2,3)$$

X

0

0

0

0

0

0

$$Y = \text{ones}(2)$$

Y

1

1

1

1

Typing `doc elmat` lists some of the elementary matrix functions in MATLAB, including a number of matrix creation functions.

Try

Create some basic matrices.

```
A = ones(2,3)
B = zeros(2)
C = eye(3)
D = magic(3)
```

Generate random numbers.

```
x1 = rand(1,1e5);
histogram(x1)
x2 = randi(6,1,1e5);
histogram(x2)
x3 = randn(1,1e5);
histogram(x3)
```

Many simulation applications require the creation of random numbers. In MATLAB, a whole matrix of pseudorandom numbers can be created at once, rather than one at a time, using an appropriate matrix creation function:

Random Number Creation Functions	
rand	Uniformly distributed between 0 and 1
randi(k)	Uniformly distributed integer between 1 and k.
randn	Normally distributed from the standard normal distribution (mean 0 and standard deviation 1)

You can generate more kinds of random numbers from other distributions with Statistics Toolbox™.

Accessing Data in Matrices

As with vectors, you can use indexing to access data in matrices. MATLAB references matrix elements with *row, column indexing*. That is, the syntax

$$x = A(j, k)$$

extracts the value in the j^{th} row and k^{th} column of the variable A and assigns it to the (scalar) variable x .

You can use assignment to change matrix elements, in the same way as vectors:

$$A(4, 3) = 0;$$

As with vectors, you can use the keyword `end` as either a row or column index:

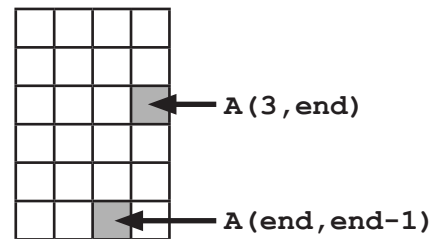
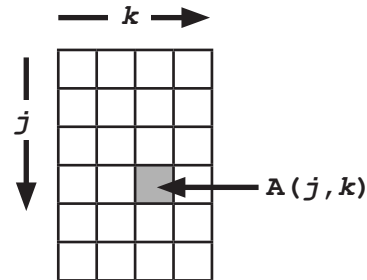
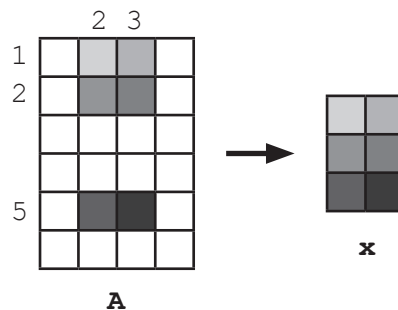
$$x = A(3, \text{end});$$

$$A(\text{end}, \text{end}-1) = 0;$$

Multiple indices reference a submatrix formed by the intersection of the row and column indices. For example,

$$x = A([1, 2, 5], [2, 3])$$

creates a 3-by-2 matrix of six elements of A .



Try

Compare electricity usage in June and October (from 1990 to 2015).

```
June = usage(6, :);
Oct = usage(10, :);
plot(yr, June)
hold on
plot(yr, Oct)
hold off
```

View the monthly electricity prices in 2015.

```
price2015 = price(:, end);
bar(mth, price2015)
```

Multiple indices are typically used to reference a range of rows or columns in a variable using the colon (`:`) notation. For example,

$$x = A(2:5, 3)$$

extracts a portion of the third column of A , and

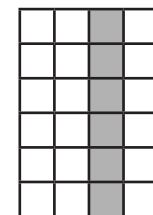
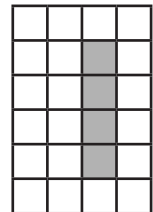
$$x = A(1:\text{end}, 3)$$

extracts the entire third column.

You can use the colon operator on its own as a shorthand for all rows or columns (`1:end`):

$$x = A(:, 3)$$

This extracts the entire third column.



```
x = A(1:6, 3);
x = A(1:end, 3);
x = A(:, 3);
```

Matrix Operations

Arithmetic operations on matrices are defined in the same way as for vectors:

$$C = A + B \quad \begin{matrix} \mathbf{A} & \begin{bmatrix} 1 & 0 & 7 \\ 3 & 2 & 5 \end{bmatrix} & + & \mathbf{B} & \begin{bmatrix} 6 & 4 & 2 \\ 1 & 4 & 0 \end{bmatrix} & = & \mathbf{C} & \begin{bmatrix} 7 & 4 & 9 \\ 4 & 6 & 5 \end{bmatrix} \end{matrix}$$

$$B = 2 * A \quad \begin{matrix} \mathbf{A} & \begin{bmatrix} 1 & 0 & 7 \\ 3 & 2 & 5 \end{bmatrix} & \times & \begin{bmatrix} 2 \end{bmatrix} & = & \mathbf{B} & \begin{bmatrix} 2 & 0 & 14 \\ 6 & 4 & 10 \end{bmatrix} \end{matrix}$$

$$B = A + 2 \quad \begin{matrix} \mathbf{A} & \begin{bmatrix} 1 & 0 & 7 \\ 3 & 2 & 5 \end{bmatrix} & + & \begin{bmatrix} 2 \end{bmatrix} & = & \mathbf{B} & \begin{bmatrix} 3 & 2 & 9 \\ 5 & 4 & 7 \end{bmatrix} \end{matrix}$$

For any nonscalar (i.e., array) operands, multiplication, division and exponentiation are ambiguous: they can be interpreted as array (element-by-element) operations or matrix operations in a mathematical sense. For this reason, MATLAB has two operators each for multiplication, division and exponentiation:

	Matrix Operation	Array Operation
Multiplication	*	.*
Division	/	./
Exponentiation	^	.^

Try

Convert prices from ¢/kWh to \$/MWh.

```
price = price*1000/100;
```

Add simulated noise to the price data.

```
noise = randn(12,26);  
nPrice = price + noise;
```

Attempt to calculate total revenue.

```
revenue = price*usage
```

Check the dimensions of the variables.

```
whos price usage
```

The dimensions are the same. Why didn't this work?



$$\begin{matrix} \text{m-by-n} & + & \text{m-by-n} & = & \text{m-by-n} \end{matrix}$$

Array Operations

The default multiplication, division, and exponentiation operators are interpreted as *matrix* operations in MATLAB. To perform an *array* (or *element-by-element*) operation, rather than a matrix operation, simply prefix the operator with a dot (.):

$$C = A \text{ .* } B;$$

1	0	7
3	2	5

 \times

6	4	2
1	4	0

 $=$

6	0	14
3	8	0

If the operands A and B are the same size, the result C will also be the same size.

If A or B is a scalar, then $C = A \text{ .* } B$ is a scalar multiplication, and is therefore equivalent to $C = A * B$.

Because matrix addition and subtraction are defined as element-by-element operations, there is no need to have separate matrix and array addition operators. Hence there is no .+ (or .-) operator in MATLAB.

Array operations can also be performed on operands of different, but compatible, sizes. For example, if A is a matrix and B a row vector with the same number of columns as A, then the vector B is expanded to match the size of A before the element-by-element operation is performed.

$$A \text{ .* } B$$

1	0	7
3	2	5
3	1	1
2	1	0

 \times

3	2	1
3	2	1
3	2	1
3	2	1

 $=$

3	0	7
9	4	5
9	2	1
6	2	0

Try

Calculate electricity revenues.

```
revenue = price.*usage
```

Calculate a ratio of prices.

```
priceInflation = price(:,end)./price(:,1)
```

Convert prices to 2015 equivalents using the value of \$1 relative to 2015.

Note the size of the result.

```
price2015 = price.*dollar2015
```

Add a row vector and a column vector. Note the size of the result.

```
A = [1 2]
```

```
B = [0;1;2;3]
```

```
C = A + B
```

In general, if the operands A and B have *compatible sizes*, then each input is expanded as needed to match the size of the other. Two arrays have compatible sizes if the size of each dimension is either the same or 1. The following table lists arrays with compatible sizes and the size of the result.

A	B	Result
<i>m-by-n</i>	<i>m-by-n</i>	<i>m-by-n</i>
<i>m-by-n</i>	<i>m-by-1</i>	<i>m-by-n</i>
<i>m-by-n</i>	1-by- <i>n</i>	<i>m-by-n</i>
<i>m-by-1</i>	1-by- <i>n</i>	<i>m-by-n</i>

Matrix Mathematics

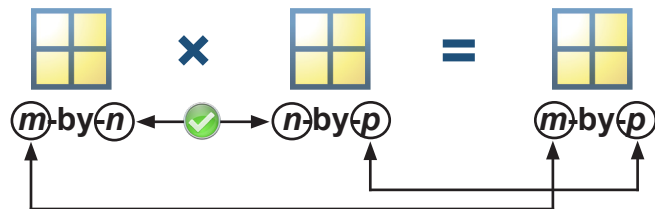
MATLAB was originally designed as a tool for performing powerful matrix calculations simply. Multiplication, division, and exponentiation operators are therefore interpreted in a mathematical matrix sense.

Mathematically, matrix multiplication of the form

$$C = A * B$$

Diagram illustrating matrix multiplication $C = A * B$. Matrix A is 3×3 , matrix B is 3×4 , and matrix C is 3×4 . The calculation $3*2 + 2*0 + 5*1 = 11$ is shown, indicating the dot product of the first row of A and the first column of B .

requires that the *inner dimensions* – the number of columns in A and the number of rows in B – agree.



If A is m -by- n and B is n -by- p , then C will be m -by- p .

Exponentiation is interpreted as the appropriate matrix operation, depending on the dimensions of the operands:

```
doc mpower
```

Try

Use matrix-vector multiplication to calculate the weighted average of prices, weighted by the value of \$1 relative to 2015.

```
w = dollar2015/sum(dollar2015)
avgWithInflation = price*w'
plot(mth,avgWithInflation)
```

Division is interpreted as the solution to a matrix equation. Because matrix multiplication is not commutative ($A*B \neq B*A$), MATLAB uses both the forward slash (/) and backslash (\) characters as division operators:

Expression	Interpretation
$x = B/A$	Solve $x*A = B$ (for x)
$x = A \setminus B$	Solve $A*x = B$ (for x)

Linear systems of equations, such as $Ax = b$, may have no solution, a unique solution, or an infinite family of solutions. MATLAB uses sophisticated algorithms to deal with all possible cases. The solution returned from the command

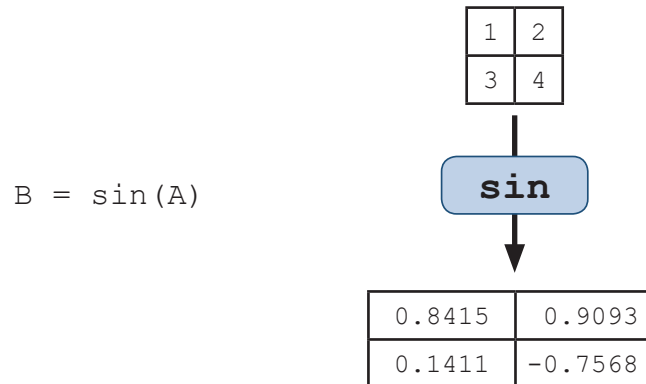
```
x = A \ b
```

is the vector (or matrix) x that minimizes the *norm of the residual* (or error), $\|Ax - b\|$, over all possible vectors (or matrices) x .

MATLAB also provides many functions for matrix algebra, such as `eig` (eigenvalues and eigenvectors), `cond` (condition number), `det` (determinant), `expm` (matrix exponential), `lu` (LU decomposition), etc.

Mathematical Functions

As with vectors, you can perform mathematical operations on a whole matrix of data with a single command:



Mathematical functions are interpreted in an element-by-element manner for vector and matrix inputs.

Try

Convert revenues to billions of dollars, and round to three decimal places (i.e., the nearest million dollars).

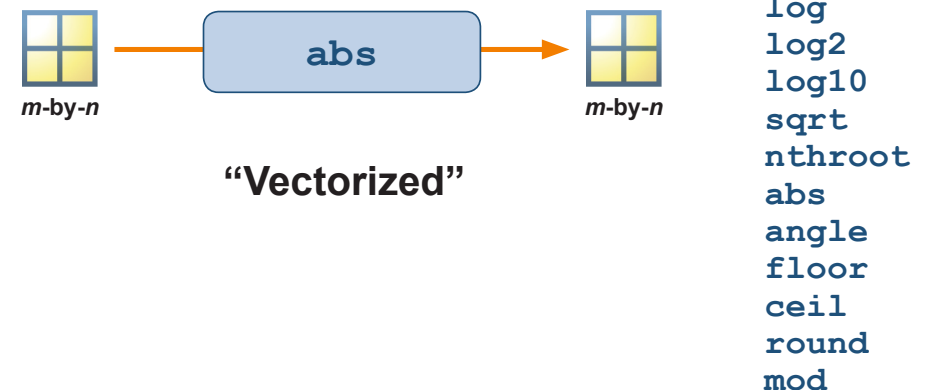
```
revBillion = round(revenue/1e9,3)
```

Calculate the geometric mean of the prices from 1990 and 2015.

```
GMprice = sqrt(price(:,1).*price(:,end))
```

Compare with the arithmetic mean.

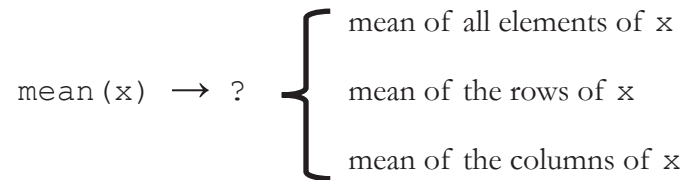
```
AMprice = (price(:,1) + price(:,end))/2
```



Data in the MATLAB® Environment


Statistical functions such as `max` and `mean` are intended to work on sets of data, rather than single values.

If `x` is a vector, `mean(x)` is unambiguous. However, if `x` is a matrix, this command could be interpreted in several ways:

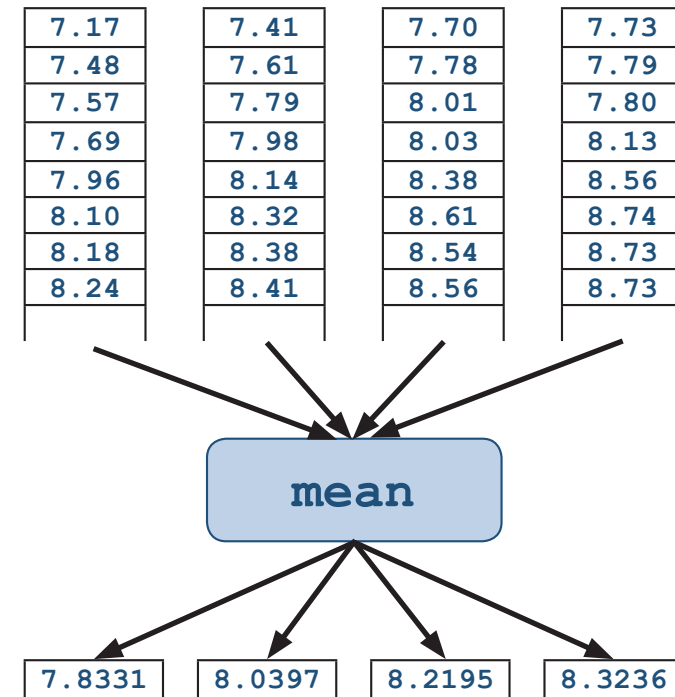


For statistical purposes in MATLAB, the rows of a matrix represent different observations or cases and the columns represent different variables.

		Variables			
		1990	1991	1992	1993
Observations or cases	Jan	7.17	7.41	7.70	7.73
	Feb	7.48	7.61	7.78	7.79
	Mar	7.57	7.79	8.01	7.80
	Apr	7.69	7.98	8.03	8.13
	May	7.96	8.14	8.38	8.56
	Jun	8.10	8.32	8.61	8.74
	Jul	8.18	8.38	8.54	8.73
	Aug	8.24	8.41	8.56	8.73

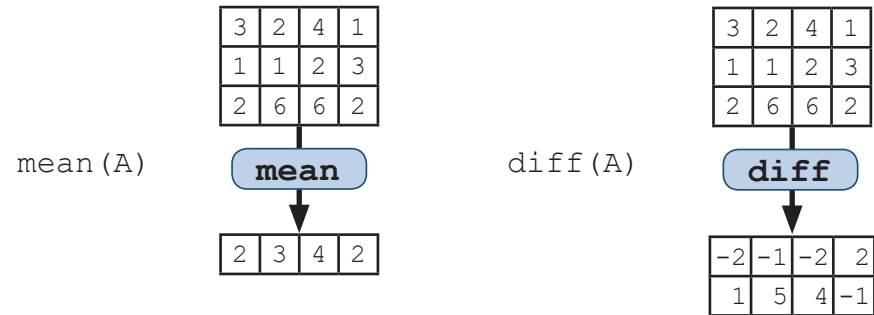

m-by-n

Hence, statistical functions in MATLAB treat the columns independently, as separate variables.



Statistical Operations

The organization of matrix data into columns of independent variables allows you to perform statistical analysis on several variables with a single command:

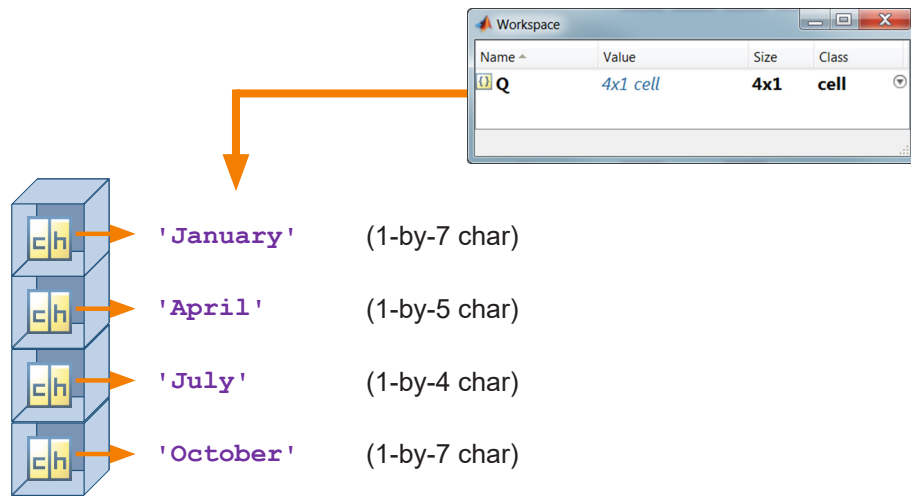


Creating Arrays of Text

To store multiple strings as an array, use curly braces (`{ }`) to create a *cell array* of character vectors:

```
Q = {'January'; 'April'; 'July'; 'October'}
```

This will create a 4-by-1 cell array where each element (cell) contains a character vector.



Many MATLAB functions for processing text accept cell arrays of text as input, as do some functions that expect character input.

The `xlabel`, `ylabel`, and `title` functions interpret a cell array input as a multiline annotation:

```
xlabel({'hello', 'world'})
```



Try

Create a cell array of 12 strings. Note the size and type of the result.

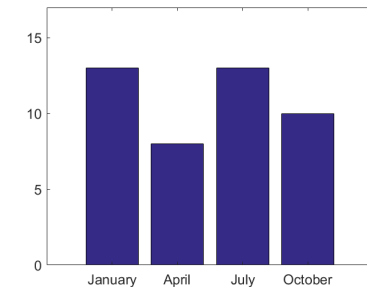
```
M = {'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', ...  
     'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'};
```

Change the x-axis tick labels to display the months as text.

```
monthly = mean(usage, 2);  
bar(mth, monthly)  
xticks(1:12)  
xticklabels(M)
```

You can use a cell array of text as input to the `xticklabels` function to modify the x-axis tick labels:

```
bar(y)  
xticklabels(Q)
```

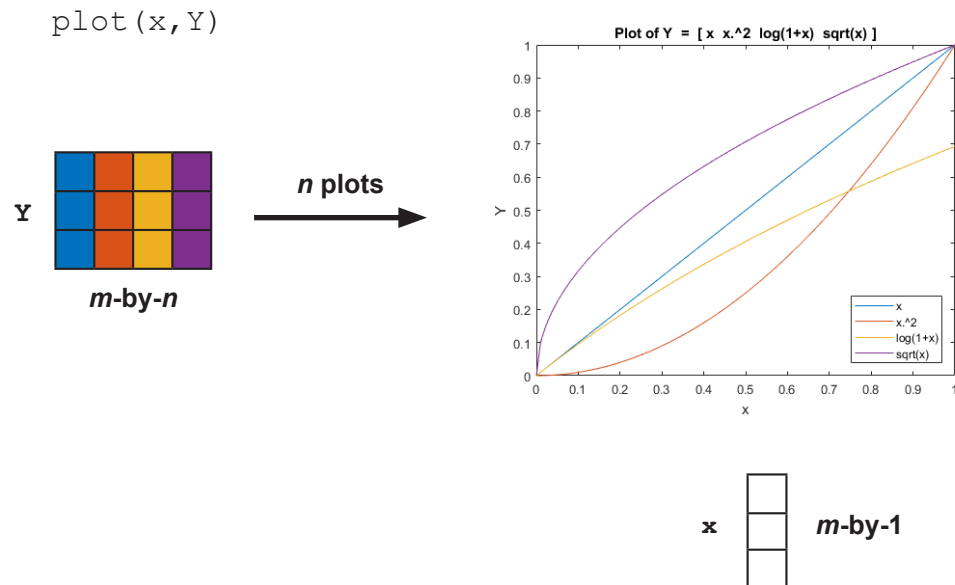


The text in each character vector within the cell array will be identified with each x-axis tick, in order. As such, it is good practice to also specify the tick locations using the `xticks` function.

```
xticks(1:4)  
xticklabels(Q)
```

Plotting Multiple Columns

Two-dimensional plotting functions (`plot`, `bar`, `stem`, etc.) behave in the same manner as statistical functions when dealing with matrices: the columns are assumed to be independent variables:



To distinguish the various variables, each column is plotted with a different color, cycling through a default sequence.

The same marker and line style options are available as for vector plots. However, only one set of options is allowed; the same options are applied to every column.

```
plot(yr, price, 'p')
```

It is possible to include a color specification, but it will also apply to all columns, thus removing any distinction between the different variables.

```
plot(yr, price, 'r:p')
```

Try

Plot each year's electricity usage simultaneously as a function of month.

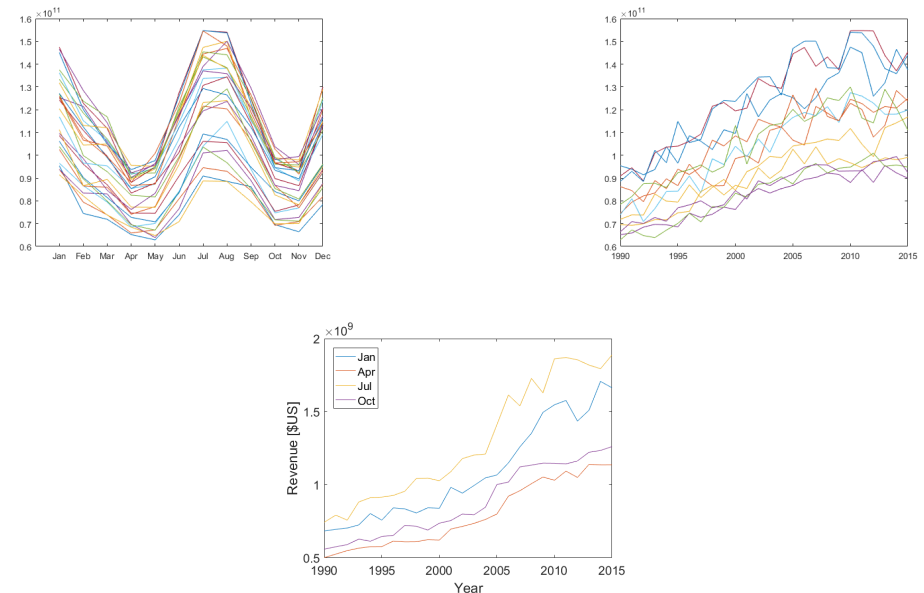
```
plot(mth, usage)
xticks(1:12)
xticklabels(M)
```

Plot each month's electricity usage simultaneously as a function of year.

```
plot(yr, usage')
```

Plot a time series of revenue for the first month of each quarter.

```
firstmonth = revenue(1:3:12, :)';
plot(yr, firstmonth)
q = {'Jan', 'Apr', 'Jul', 'Oct'};
legend(q, 'Location', 'northwest')
xlabel('Year')
ylabel('Revenue [$US]')
```



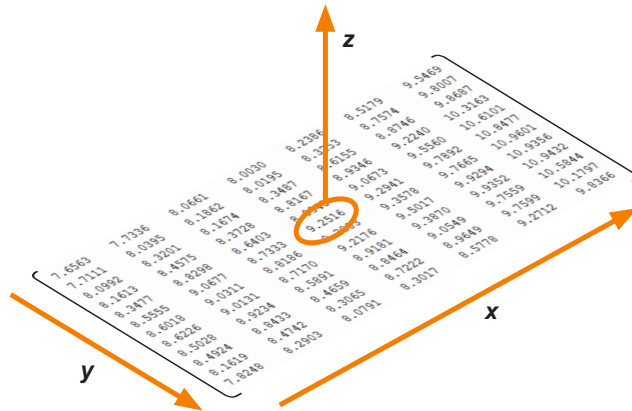
Matrix Visualization

You can visualize an m -by- n matrix of values Z by representing each value of Z as the height of a surface above the x - y plane. The command

```
surf(Z)
```

plots $Z(j, k)$ above the point $x = k, y = j$. To specify x and y coordinates (rather than row and column indices), use the syntax

```
surf(X, Y, Z)
```



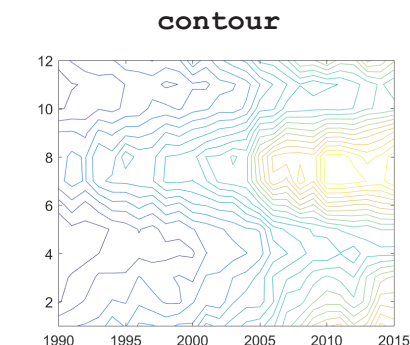
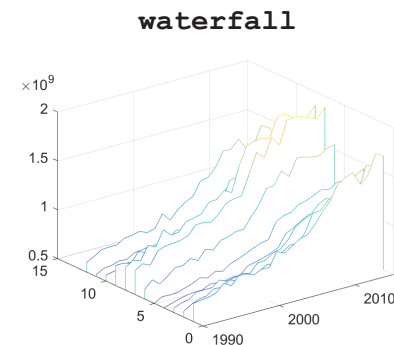
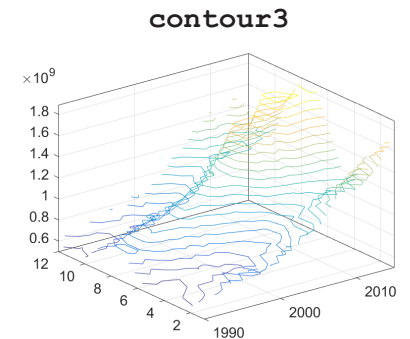
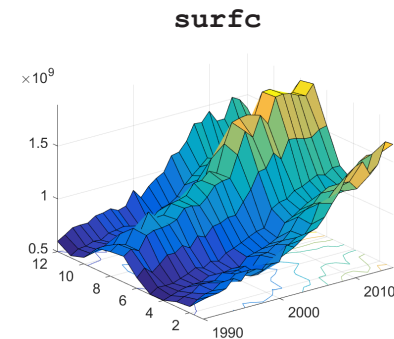
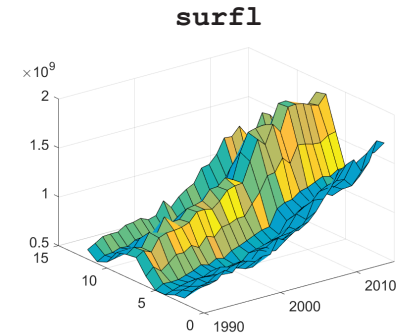
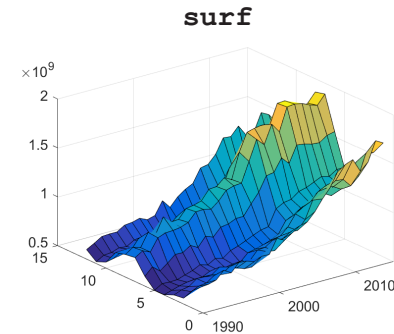
where either X is an n -element vector and Y is an m -element vector, or X and Y are both m -by- n matrices.

Some additional plot types for visualizing matrix data are shown. They all have the same syntax as `surf`.

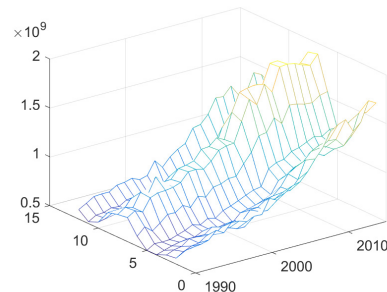
Try

Visualize the electricity revenue as a surface.

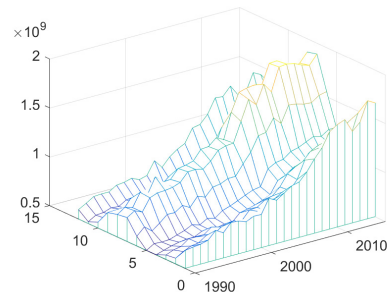
```
surf(yr, mth, revenue)
```



mesh



meshz

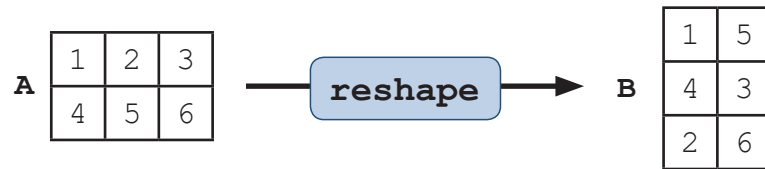


waterfall

Reshaping

You can use the `reshape` function to reshape an m -by- n matrix into a p -by- q matrix, as long as the total number of elements remains the same (i.e., $m*n = p*q$):

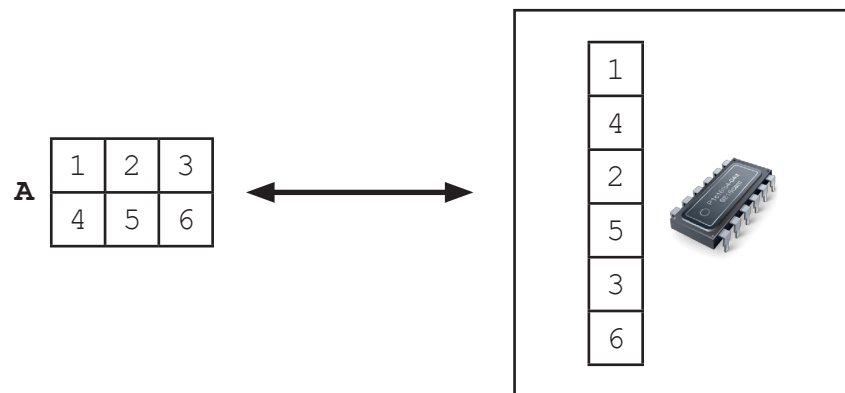
```
A = [1,2,3;4,5,6]    % Start as 2-by-3
B = reshape(A,3,2)    % Change to 3-by-2
```



For convenience, you can leave one of the dimensions blank when calling `reshape`:

```
reshape(A,3,[])    % Automatically 2 columns
reshape(A,[],2)    % Automatically 3 rows
```

Note that it is not necessary to specify the order in which the elements of **A** are extracted into **B**. This is because MATLAB stores data in memory in a *column-major* format. That is, **A** is stored sequentially in memory as 1, 4, 2, 5, 3, 6.



Try

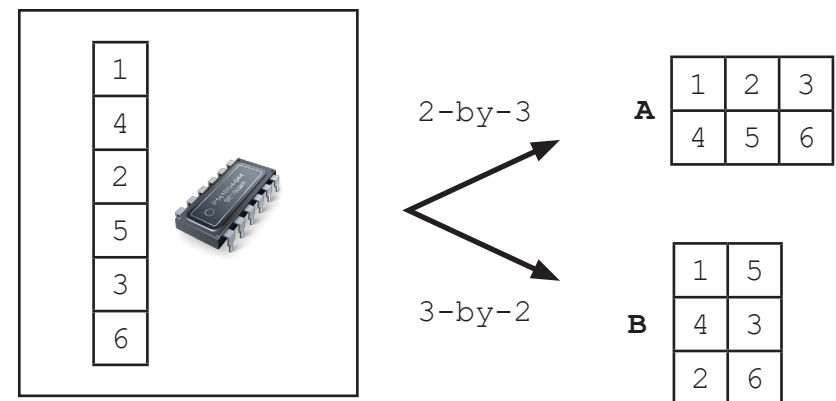
Calculate the average electricity usage over all time.

```
usageVec = reshape(usage,[],1);
mean(usageVec)
```

Plot electricity prices as a single time series.

```
priceVec = reshape(price,[],1);
plot(priceVec,'.-')
```

This means that reshaping a matrix does not change what is stored in memory, only the dimensionality information that MATLAB uses to interpret the variable. This provides a powerful way to group and regroup data without performing calculations or copying memory.



Summary

- Creating and manipulating matrices
- Calculations with matrices
- Statistics with matrices of data
- Matrix visualization

Function	Use
[]	Concatenation
ones zeros rand randi randn	Create matrices
* / ^	Matrix operations
.* ./ .^	Array (elementwise) operations
reshape	Change dimensions of an array
surf surfl surfc mesh waterfall contour contour3	Create various matrix visualizations
{ }	Create a cell array
x/y/zticks x/y/zticklabels	Set axis tick locations and labels

Test Your Knowledge

Name: _____

1. (Select all that apply): Given a 2-by-3 matrix A and a 3-by-2 matrix B, which of the following operations are valid?
 - A. $A+B$
 - B. $A. +B$
 - C. $A*B$
 - D. $A. *B$

2. If A is a 15-by-7 matrix, which of the following commands will result in five line plot on the same axes?
 - A. `plot(A)`
 - B. `plot(A(:,3:end))`
 - C. `plot(A(11:end,:),:)`
 - D. `plot(A(2:6))`