Array Operations
Array Operations

- Using MATLAB as a glorified calculator is OK, but its real strength is in matrix manipulations
Definitions

- **Scalars**: Variables that represent single numbers. Note that complex numbers are also scalars, even though they have two components.

- **Arrays**: Variables that represent more than one number. Each number is called an **element** of the array. Array operations allow operating on multiple numbers at once.

- **Row and Column Arrays (Vector)**: A row of numbers (called a row vector) or a column of numbers (called a column vector).

- **Two-Dimensional Arrays (Matrix)**: A two-dimensional table of numbers, called a matrix.
To create a row vector, enclose a list of values in brackets.
You may use either a space or a comma as a “delimiter” in a row vector.
Use a semicolon as a delimiter to create a new row.
Use a semicolon as a delimiter to create a new row.
Hint: It’s easier to keep track of how many values you’ve entered into a matrix, if you enter each row on a separate line. The semicolons are optional.
Shortcuts

- While a complicated matrix might have to be entered by hand, evenly spaced matrices can be entered much more readily. The command
  \[
  \mathbf{b} = 1:5
  \]
  or the command
  \[
  \mathbf{b} = [1:5]
  \]
  both return a row matrix
The default increment is 1, but if you want to use a different increment put it between the first and final values.
To calculate spacing between elements use…

- `linspace()`
- `logspace()`
The increment is computed internally, having the value:

\[ \text{increment} = \frac{\text{end} - \text{start}}{\text{number} - 1} \]
MATLAB 7.12.0 (R2011a)

Command Window:

```
>> d=linspace(1,10,3)
  d =
      1.0000    5.5000    10.0000
>> e=logspace(1,3,3)
```

- **Initial value in the array expressed as a power of 10**: 1.0000
- **Final value in the array expressed as a power of 10**: 10.0000
- **Number of elements in the array**: 3

Detail: initial value 1.0000, final value 10.0000, number of elements 3.
It is a common mistake to enter the initial and final values into the logspace command, instead of entering the corresponding power of 10.
Hint

• You can include mathematical operations inside a matrix definition statement.

• For example

\[ a = [0: \pi/10: \pi] \]
Mixed calculations between scalars and arrays

- Matrices can be used in many calculations with scalars
- There is no confusion when we perform addition and subtraction
- Multiplication and division are a little different
- In matrix mathematics the multiplication operator (*) has a very specific meaning
$\text{Rate of temperature change, degrees/hour}$

```
>> a=[1 2 3]
  a =
       1   2   3
>> b=a+5
  b =
       6   7   8
```
Addition between arrays is performed on corresponding elements.
Multiplication between arrays is performed on corresponding elements if the .* operator is used.
MATLAB interprets * to mean matrix multiplication. The arrays a and b are not the correct size for matrix multiplication in this example.
Array Operations

• Array multiplication \( .* \)
• Array division \( ./ \)
• Array exponentiation \( .^ \)

In each case the size of the arrays must match
Array Operations

Scalar-Array Mathematics
Addition, subtraction, multiplication, and division of an array by a scalar simply apply the operation to all elements of the array.

```
>> f = [1 2 3; 4 5 6]
f =
1 2 3
4 5 6
>> g = 2*f - 1
g =
1 3 5
7 9 11
```
Matrix Operations

**Matrix Addition (Subtraction)**

\[ A \pm B = \begin{pmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
\end{pmatrix}
\pm
\begin{pmatrix}
  b_{11} & b_{12} & b_{13} \\
  b_{21} & b_{22} & b_{23} \\
  b_{31} & b_{32} & b_{33}
\end{pmatrix} =
\begin{pmatrix}
  a_{11} \mp b_{11} & a_{12} \mp b_{12} & a_{13} \mp b_{13} \\
  a_{21} \mp b_{21} & a_{22} \mp b_{22} & a_{23} \mp b_{23} \\
  a_{31} \mp b_{31} & a_{32} \mp b_{32} & a_{33} \mp b_{33}
\end{pmatrix} \]
Array Operations

Examples: Addition & Subtraction

\[ A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \]

\[ A + B = \begin{pmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{pmatrix} \]

\[ A - B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \]
Array Operations

Element-by-Element Array-Array Mathematics

\[
\begin{align*}
\text{A} &= [2 \ 5 \ 6] \\
\text{B} &= [2 \ 3 \ 5] \\
\text{C} &= \text{A} \times \text{B} \\
\text{D} &= \text{A} / \text{B} \\
\text{E} &= \text{A} ^ \text{B} \\
\text{F} &= 3.0 ^ \text{A}
\end{align*}
\]
Array Operations

MATRIX Multiplication (element by element)

\[
A = \begin{pmatrix}
    a_{11} & a_{12} & a_{13} \\
    a_{21} & a_{22} & a_{23} \\
    a_{31} & a_{32} & a_{33}
\end{pmatrix}
\]

\[
B = \begin{pmatrix}
    b_{11} & b_{12} & b_{13} \\
    b_{21} & b_{22} & b_{23} \\
    b_{31} & b_{32} & b_{33}
\end{pmatrix}
\]

\[
A \times B = \begin{pmatrix}
    a_{11} \times b_{11} & a_{12} \times b_{12} & a_{13} \times b_{13} \\
    a_{21} \times b_{21} & a_{22} \times b_{22} & a_{23} \times b_{23} \\
    a_{31} \times b_{31} & a_{32} \times b_{32} & a_{33} \times b_{33}
\end{pmatrix}
\]
Array Operations

Examples: Multiplication & Division (element by element)

\[ A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \]

\[ B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \]

\[ A \cdot B = \begin{pmatrix} 1 & 4 & 9 \\ 16 & 25 & 36 \\ 49 & 64 & 81 \end{pmatrix} \]

\[ A \cdot / B = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \]
The **matrix multiplication** of \( m \times n \) matrix \( A \) and \( n \times p \) matrix \( B \) yields \( m \times p \) matrix \( C \), denoted by 

\[
C = AB
\]

Element \( c_{ij} \) is the inner product of row \( i \) of \( A \) and column \( j \) of \( B \)

\[
c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}
\]

**Note** \( AxB \neq BxA \)
Array Operations

Matrix Multiplication

\[ A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \]

\[ B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} \]

\[ A \times B = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} \end{pmatrix} \]

\[ \begin{pmatrix} a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \\ a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} \end{pmatrix} \]

\[ \begin{pmatrix} a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} \\ a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} \\ a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33} \end{pmatrix} \]
Array Operations

Example: Matrix Multiplication

\[ A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 3 & 1 & 2 \end{pmatrix} \]

\[ B = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 3 & 1 & 2 \end{pmatrix} \]

\[ A \times B = \begin{pmatrix} 16 & 9 & 11 \\ 12 & 11 & 13 \\ 12 & 10 & 14 \end{pmatrix} \]
The matrix capability of MATLAB makes it easy to do repetitive calculations

- For example, assume you have a list of angles in degrees that you would like to convert to radians.
  - First put the values into a matrix.
  - Perform the calculation
```matlab
>> degrees = [10 15 70 90];
>> radians = degrees*pi/180
radians =
    0.1745    0.2618    1.2217    1.5708
```
The value of pi is built into MATLAB as a floating point number, called pi.

Either the * or the .* operator can be used for this problem, because it is composed of scalars and a single matrix.
The built-in MATLAB functions such as $\sqrt{x}$ and $\exp(x)$ automatically operate on array arguments to produce an array result the same size as the array argument $x$.

Thus these functions are said to be *vectorized* functions.

For example, in the following session the result $y$ has the same size as the argument $x$.

```matlab
>> x = [4, 16, 25];
>> y = sqrt(x)
```

$y = \begin{bmatrix} 2 & 4 & 5 \end{bmatrix}$
Using arrays in functions

However, when multiplying or dividing these functions, or when raising them to a power, **you must use element-by-element operations if the arguments are arrays.**

For example, to compute $z = (e^y \sin x) \cos^2 x$, you must type

$$z = \exp(y) \cdot \sin(x) \cdot (\cos(x))^\cdot 2.$$ 

You will get an error message if the size of $x$ is not the same as the size of $y$. The result $z$ will have the same size as $x$ and $y$. 
Using arrays in functions

We can raise a scalar to an array power. For example, if \( p = [2, 4, 5] \), then typing \( 3 \cdot p \) produces the array \([3^2, 3^4, 3^5] = [9, 81, 243]\).

Remember that \( \cdot^p \) is a single symbol. The dot in \( 3 \cdot p \) is not a decimal point associated with the number 3.

The following operations, with the value of \( p \) given here, are equivalent and give the correct answer:

\[
3^p \\
3.0^p \\
(3)^p \\
3^\{2,4,5\}
\]
 transpose operator changes rows to columns or vice versa.
The transpose operator makes it easy to create tables.
table = [degrees'; radians'] would have given the same result
The transpose operator works on both one dimensional and two dimensional arrays.

```
>> degrees = [10 15 70 90]
degrees =
     10    15    70    90
>> radians = degrees.*pi/180
radians =
    0.1745   0.2618   1.2217   1.5708
>> table = [degrees', radians']
table =
      10.0000   0.1745
      15.0000   0.2618
      70.0000   1.2217
      90.0000   1.5708
>> table'
an =
    10.0000   15.0000   70.0000   90.0000
    0.1745   0.2618   1.2217   1.5708
```
Data Analysis Functions

- max(x)
- min(x)
- mean(x)
- median(x)
- sum(x)
- prod(x)
- sort(x)
- sortrows(x)

- std(x)
- var(x)
3.5.1 Max and Min

When the max function is used with a vector (either a row or a column), it returns the maximum value in the vector.
When x is a matrix, the max is found for each column.
The max function can also be used to determine where the maximum occurs.
```matlab
>> x=[1,5,3];
>> [a,b]=max(x)
a =
    5
b =
    2
>> x=[1,5,3;2,4,6]
x =
    1  5  3
    2  4  6
>> [a,b]=max(x)
a =
    2  5  6
b =
    2  1  2
```
3.5.4 Sorting Values

It’s easy to sort data in MATLAB, using the sort function.

The default is to sort in ascending order.
To sort in descending order, just add the word ‘descend’ in the second input field.
MATLAB is column dominant, so when sort is used with a 2-D matrix, each column is sorted in ascending order.
The sortrows function allows you to sort entire rows, based on the value in a specified column.

The default sorting column is #1
In this example the matrix is sorted in ascending order, based on the second column.
To sort based on descending order, place a negative sign in front of the column number.

Notice that this is a different strategy than that used by the sort function!
3.5.5 Determining Matrix Size

- `size(x)` number of rows and columns
- `length(x)` biggest dimension
- `numel(x)` total number of elements
```matlab
>> x=[1,5,3;2,4,6]
x =
    1   5   3
    2   4   6
>> size(x)
ans =
    2   3
>> length(x)
ans =
    3
>> numel(x)
ans =
    6
```
3.5.6 Variance and Standard Deviation

- \( \text{std}(x) = \sigma \)
- \( \text{var}(x) = \sigma^2 \)

\[
\sigma^2 = \frac{\sum_{k=1}^{N} (x_k - \mu)^2}{N - 1}
\]
Standard Deviation
3.6 Random Numbers

- **rand(x)**
  - Returns an \( x \) by \( x \) matrix of random numbers between 0 and 1

- **rand(n,m)**
  - Returns an \( n \) by \( m \) matrix of random numbers

- These random numbers are evenly distributed
>> rand(3)
ans =
 0.8147   0.9134   0.2785
 0.9058   0.6324   0.5469
 0.1270   0.0975   0.9575

>>
If you create a very large matrix of random numbers using the \texttt{rand} function, the average value will be 0.5

Notice that we created a 1 by $10^7$ matrix, which required 2 inputs (\texttt{rand(1,10e6)}). If we had entered a single value (\texttt{rand(10e6)}) the result would have been a $1 \times 10^7$ by $1 \times 10^7$ matrix.
Gaussian Random numbers

- `randn(n)`
- Also called a normal distribution
- Generates numbers with a mean of 0 and a standard deviation of 1
First generate an array of 10 million gaussian random numbers.

Use MATLAB to take the mean, and notice that it is very close to 0.

Use MATLAB to find the standard deviation, and notice that it is very close to 1.
The hist function creates a histogram of the input data.
To generate random numbers between other bounds...

\[ x = (b - a) \cdot r + a \]

- \( a \) and \( b \) are the upper and lower bounds
- \( r \) is the array of random numbers
Although the average is the same for each of these data sets, they have a different standard deviation.
More about Manipulating Matrices

• $M(\cdot)$
  – Converts a two dimensional matrix to a single column
```
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{array}
\end{array}
```
```
% MATLAB code

>> std(x)
ans =
    3    3    3

>> x(:)
ans =
    1
    2
    3
    4
    5
    6
    7
    8
    9

>> mean(x(:))
ans =
    5

>> std(x(:))
ans =
    2.7386
```
• **Scientific Notation**
  
  – Although you can enter any number in decimal notation, it isn’t always the best way to represent very large or very small numbers
  
  – In MATLAB, values in scientific notation are designated with an e between the decimal number and exponent. (Your calculator probably uses similar notation.)
It is important to omit blanks between the decimal number and the exponent. For example, MATLAB will interpret \( 6.022 \, \text{e}^{23} \) as two values (6.022 and \( 10^{23} \)).
Display Format

- Multiple display formats are available
- No matter what display format you choose, MATLAB uses double precision floating point numbers in its calculations
- MATLAB handles both integers and decimal numbers as floating point numbers
Default

• The default format is called short
• If an integer is entered it is displayed without trailing zeros
• If a floating point number is entered four decimal digits are displayed
Other formats:

- Changing the format affects all subsequent displays.
  - `format long` results in 14 decimal digits.
  - `format bank` results in 2 decimal digits.
  - `format short` returns the display to the default 4 decimal digits.
Really Big and Really Small

- When numbers become too large or too small for MATLAB to display using the default format, it automatically expresses them in scientific notation.
- You can force scientific notation with:
  - `format short e`
  - `format long e`
Common Scale Factor

- For long and short formats, a common scale factor is applied to the entire matrix if some of the elements become very large, or very small. This scale factor is printed along with the scaled values.
```matlab
>> drag=20000;
>> r=0.000001;
>> V=100*0.4470;
>> A=1;
>> cd=drag*2/(r*V^2*A)
   cd =  
       2.0019e+007
>> V=0:20:200;
>> V=V*0.447;
>> drag=cd*r*V.^2*A/2;
>> table = [V',drag']

Table =  
       0    0.0009    0.0018    0.0027    0.0036    0.0045    0.0054    0.0063
       0    0.0800    0.3200    0.7200    1.2800    2.0000    2.8800    3.9200
```

**Common Scale Factor**

-5 -4.5 -4 -3.5 -3 -2.5 -2 -1.5 -1 -0.5 0

**Rate of Change**

- time, hour
- Rate of temperature change, degrees/hour

**Command Window**
Two other formats

- `format +`
- `format rat`
Spacing in the command window

- The `format` command allows us to control how tightly information is spaced in the command window:
  - `format compact`
  - `format loose` (default)

- Most of the examples in this presentation use `format compact`

```
>> A
A =
   51/10
```

```
>> format loose
>> A
A =
   51/10
```

Notice that the value of A is still being displayed using the `rat` format, because we haven’t changed it back to `format short`
Section 2.4
Saving Your Work

• If you save a MATLAB session performed in the command window, all that is saved are the values of the variables you have named
Variables are saved, not the commands in the command window
Save with a command in the command window

```matlab
>> A=3
A =
    3
>> B=1:10;
>> save my_neat_matlab_file
```
MATLAB automatically saves to a .mat file

- If you want to save to another format, such as .dat, you need to explicitly tell the program

```matlab
save <file_name> <variable_list> -ascii
```

Again – Remember that the only things being saved are the values stored in the workspace window – not the commands from the command window
Script M-files

- If you want to save your work, (the commands you entered) you need to create an M-file
- File->New->M-file
- Type your commands in the edit window that opens
% Holly Moore
% Date
% Section
% An Example Problem

% Commands to create two vectors
x = [1 2 3 4 5];
y = [10 20 30 40 50];
% Command to create a plot
plot(x,y)
• The file can be saved into the current folder/directory
• It runs in the command window
Save the file using the save icon, or the file menu

% Holly Moore
% Date
% Section
% An Example Problem

% Commands to create two vectors
x = [1 2 3 4 5];
y = [10 20 30 40 50];
% Command to create a plot
plot(x,y)
You can dock the editing window with the MATLAB desktop, by using the docking arrow.

% Holly Moore
% Date
% Section
% An Example Problem

% Commands to create two vectors
x = [1 2 3 4 5];
y = [10 20 30 40 50];
% Command to create a plot
plot(x,y)
This arrangement is often easier to use.
I saved this file as example.m

Notice that it now appears in the current directory

When I execute the file, the figure appears on top of the MATLAB desktop
The figure window can also be docked onto the MATLAB desktop, using the docking arrow.
Notice that the command history window is hidden underneath the figure, but can be accessed with the tab.
Comments

- Be sure to comment your code
  - Add your name
  - Date
  - Section #
  - Assignment #
  - Descriptions of what you are doing and why
The % sign identifies comments
You need one on each line
Cell Mode

Cell Mode

- Enables the user to execute one section of code at a time
- Especially useful as you debug your code
- To use cell mode you must activate the cell toolbar
Cell Toolbar

Cell Dividers

Be sure to include a space after the %%
Summary

• Introduced the MATLAB Windows
• Basic matrix definition
• Save and retrieve MATLAB data
• Create and use script M-files
• The use of cell mode