INTRODUCTION TO MATLAB

1 First Steps in MATLAB

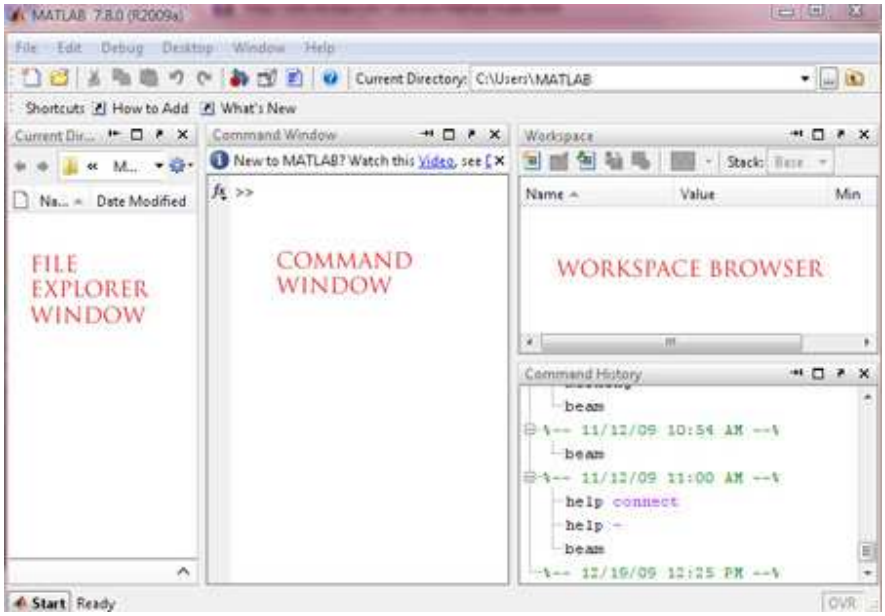
1.1 Starting MATLAB

Matlab stands for matrix laboratory.

Matlab is a program which allows you to manipulate, analyze and visualize data.

MATLAB allows easy matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages. The drawback to using Matlab is the specific syntax you will have to learn prior to being able to operate the software efficiently. The purpose of this tutorial is to introduce you to the basics of Matlab and give you the skills you will need to complete the homework in this class. It is important to note that Matlab has many additional functions and features which will not be discussed here, but may be helpful in your future work.

- You can start MATLAB by double-clicking on the MATLAB icon or invoking the application from the Start menu of Windows. The main MATLAB window, called the MATLAB Desktop, will then pop-up and it will look like this:



You get matlab to do things for you by typing in commands. Mat-lab prompts you with two greater-than signs (>>).when it is ready to accept a command from you .

- The ">>" is called the command prompt, and there will be a blinking cursor right after it waiting for you to type something. The idea is that you type commands at the command prompt for MATLAB to execute; after you type a command at a command prompt, MATLAB executes the command you typed in, then prints out the result. It then prints out another command prompt and waits for you to enter another command. In this way, you can interactively enter as many commands to MATLAB as you want.

- Another simple command you can try now is the clc command (clear command window). In the MATLAB command window, if you want to clear away all the visible text and have the cursor move to the top of the window, then type clc at the command prompt. Go ahead and try it now to see what happens. That was simple, wasn't it?

- To end a matlab session type quit or exit at the matlab prompt.
- You can type help at the matlab prompt or pull down the Help Menu on a PC.

The various forms of help available are

helpwin -Opens a matlab help GUI

helpdesk -Opens a hypertext help browser

demo- Starts the matlab demonstration

- The complete documentation for matlab can be accessed from the hypertext helpdesk.
- You can learn how to use any matlab command by typing help followed by the name of the command, for example, **help sin**.
- You can also use the **lookfor** command which searches the help entries for all matlab commands, for a particular word. For example if you want to know which matlab functions to use for spectral analysis, you could type **lookfor** spectrum. Matlab responds with the names of the commands that have the searched word in the first line of the help entry.
- You can search the entire help entry for all matlab commands by typing **lookfor** -all keyword.

1.2 First Steps

To get matlab to work out 1+1,

1. Type the following at the prompt:

>> 1+1

2. Hit the enter key.

Matlab responds with

>> ans=2

- In its simplest mode of use, MATLAB can be used just like a pocket calculator. For example, here is how you would do some simple, calculator-like operations with MATLAB:
- 4 + 10
  ans = 14
  5 *10 + 6
  ans = 56
  (6 + 6) / 3
  ans = 4
  9^2
  ans = 81

  The ans Variable

- As you can see, MATLAB supports all the basic arithmetic operations: +, -, *, /, ^, etc.; and you can group and order operations by enclosing them in parentheses. However, what exactly is ans above? In short ans is short for "answer", and is used in MATLAB as

the default variable name when none is specified. You can refer to that value by just typing ans:

- refer to that value by just typing ans:
- 4 + 5
  ans = 9
  ans
  ans = 9
- However, if a new equation is entered the value of ans would change.
- ans + 10
  ans = 19

The spacing of operators in formulas does not matter. The following formulas both give the same answer:

1+3*2-1/2*4

1 + 3 * 2 - 1 / 2 * 4

1.3 Matrices

The basic object that MATLAB deals with is a matrix. Matrix is an array of numbers.

- The size of a matrix is the number of rows by the number of columns.
- The following syntax finds the size of a matrix:
- ```
  d = size(X)
  [m,n] = size(X)
   m = size(X,dim)
  [d1,d2,d3,...,dn] = size(X),
  ```

Description

- `d = size(X)` returns the sizes of each dimension of array `X` in a vector `d` with `ndims(X)` elements. If `X` is a scalar, which MATLAB software regards as a 1-by-1 array, `size(X)` returns the vector `[1 1]`.
- `[m,n] = size(X)` returns the size of matrix `X` in separate variables `m` and `n`.

  Where m=number of rows and n=number of columns.

- `m = size(X, dim)` returns the size of the dimension of `X` specified by scalar `dim`.
  ```

1.4 Variables

Your own variable

Variables in MATLAB are named objects that are assigned using the equals sign (=).They are limited to 31characters and can contain upper and lower case letters.

- Matlab gives the user the facility to define and use their own variables by assigning a variable name to an expression by using the equal (=) sign as statement operator

>>a=10

a=10

Variable names

- A variable name can be a combination of:
  - ✓ Alphabet characters (a to z), numbers and the underscore character (_)
- You cannot have spaces in your variable name

  e.g. first one =1
  - ✓ Use the underscore whenever you want to use space in your variable name

    e.g. some_text =54
- Matlab is case sensitive
  - ✓ The same word with different capitalization refer to different variable names

    e.g. Timex, timex, tiMex and TIMEX are different variable names
- NB: Do not start variable names with numerals.

e.g. 2for1='1' , 7tim =84.

Suppressing Results with Semicolons

- Semicolons typed after commands can be used to hide the printing out of results. If you type an expression (such as "b = 4 + 5") and then follow it with a semicolon, then MATLAB will evaluate the expression and store the result internally, but it will not print out OR **display** the results in the MATLAB command window for you to see. For example:

```
a = 10;
b = 20;
c = 30;
d = 40;
the_average = (a + b + c + d) / 4

    the_average =
    25
```

Listing Currently Defined Variables and Clearing Variables

- Let's say you have defined a lot of different variables. You probably can't remember all the variable names you have defined, and so it would be nice to get a list of all the variables currently defined. This is exactly what the whos command does. Simply typing whos at the command prompt will return to you the names of all variables that are currently defined. For example:
- clear % clear variables from memory - see notes below
  a = 5
  a = 5
  b = 6
  b = 6
  whos

NAME	SIZE	BYTES	CLASS
a	1x1	8	Double array
b	1x1	8	Double array

- Grand total is 2 elements using 16 bytes
- Once you are done with the variables that you have defined how do you remove them from memory? This is exactly what the clear command is for. Typing clear at the command prompt will remove all variables and values that were stored up to that point. For example, continuing from the above example:
- whos

NAME	SIZE	BYTES	CLASS
a	1x1	8	Double array
b	1x1	8	Double array

Grand total is 2 elements using 16 bytes

Clear

Whos

1.5 The Colon Operator

To generate a vector of equally – spaced elements MATLAB provides the

Colon operator. Try the following commands:

1:5

0:2:10

0:0.1:2*pi

- Positive increment

>>X=0.0:0.5:2:0

X =

0 0.5000 1.0000 1.5000 2.0000

- Negative increment

>>Y=10:-1:4

Y =

10 9 8 7 6 5 4

The syntax x: y means roughly ‘’generate the ordered set of numbers from x to y with increment 1 between them’’. The syntax x: d: y means roughly ‘’generate the ordered set of numbers from x to y with increment d between them’’.

1.6 LinSpace

To generate a vector of evenly spaced points between two end points,

You can use the function linspace (start, stop, № of points):

>> x=linspace(0,1,10)

x =

Columns 1 through 6

0 0.1111 0.2222 0.3333 0.4444 0.5556

Columns 7 through 10

0.6667 0.7778 0.8889 1.0000

Generates 10 evenly spaced points from0to1.Typing linspace (start, stop) will generate a vector of 100 points.

2 .0 Typing into MATLAB

2.1 Command Line Editing

If you make a mistake when entering a MATLAB command ,you do not

Have to type the whole line again. The arrow keys can be used to save much typing.

↑	Ctrl-p	Recall previous line
↓	Ctrl-n	Recall next line

←	Ctrl-b	Move back one character
→	Ctrl-f	Move forward one character
Ctrl - →	Ctrl-r	Move right one word
Ctrl - ←	Ctrl- l	Move left one word
Home	Ctrl-a	Move to beginning of line
End	Ctrl-e	Move to end of line
Esc	Ctrl-u	Clear line
Del	Ctrl-d	Delete character at cursor
Backspace	Ctrl-h	Delete character before cursor
	Ctrl-k	Delete (kill) to end of line

If you finish editing in the middle of a line ,you do not have to put the cursor at the end of the line before pressing the return key ; you can press return when the cursor is any where on the command line.

2.2 Smart Recall

Repeated use of the ↑ key recalls earlier commands .If you type the first few characters of a previous command and then press the ↑key MATLAB will recall the last command that began with those characters.

Subsequent use of ↑will recalls earlier commands that began with those characters.

2.3 Long Lines

When writing a long Matlab statement that becomes to long for a single line, leave a space followed by three full stops ("**...**") at the end of the line to continue on the next line. e.g.

1. A = [1, 2; ...
   3, 4];

2. Final_answer = Bigmatrix (row_indices, column_indices) + **…**
Another_vector*SomethingElse;

USE OF THE PERCENTAGE SIGN

The symbol "%" is used to indicate a comment (for the remainder of the line).

Interrupting calculations

If MATLAB is hung up in a calculation, or is just taking too long to perform an operation, you can usually abort it by typing CTRL+C (that is, hold down the key labeled CTRL, or CONTROL, and press C).

3.0 Matrices

One of the easy ways to learn MATLAB is to understand how MATLAB handles matrices. Think in terms of arrays and vectors when you work with MATLAB. The basic types in Matlab are **scalars (**1x1 matrix**), vectors** (row vector: 1xn matrix, column vector: mx1 matrix), and **matrices (m x n): a set of numbers arranged in rows (m) and columns (n):**

```
A = [1 2; 3 4];          % Creates a 2x2 matrix
N = 5                    % A scalar (1x1 matrix)
v = [1 0 0]              % A row vector
v = [1; 2; 3]            % A column vector
```

3.1 Typing Matrices

To type a matrix into MATLAB you must

- Begin with a square bracket [
- Separate elements in a row with commas or spaces

- Use a semi colon **(;)** to separate rows
- End the matrix with another square bracket]

For example, a 3x5 matrix **B** with the following elements:

first row: 1, 0, 9, 4, 3
second row: 0, 8, 4, 2, 7
third row: 14, 90, 0, 43, 25

would be entered in MATLAB as follow:

```
» B=[1,0,9,4,3;0,8,4,2,7;14,90,0,43,25]

B =

     1     0     9     4     3
     0     8     4     2     7
    14    90     0    43    25
```

Note that you may use a space in place of the comma in separating the column entries.

Matrix Concatenation

>>a=[1 2 3], b=[7 9 4], c=[1 3 5 8]

then

>>d=[a b c]

D =

1 2 3 7 9 4 1 3 5

>> d=[a;b;c]

d =

1 2 3

7 9 4

1 3 5

Z= [(0:2:10);(5:-0.2:4)] results in 2x6 matrix

```
Z =

         0    2.0000    4.0000    6.0000    8.0000   10.0000

    5.0000    4.8000    4.6000    4.4000    4.2000    4.0000
```

Indexing vectors and matrices.

You may extract a certain group or element from an existing matrix. Say you wish to create a new array C from the second row of matrix B. Specify the row number and ":" for all columns in that row as shown below:

```
» C=B(2,:)

C =

     0     8     4     2     7
```

Similarly, You may also form a matrix from the element of an existing matrix:

```
» D=[B(1,2),B(1,4);B(3,2),B(3,5)]

D =

     0     4
    90    25
```

>> my_matrix = [8, 12, 19; 7, 3, 2; 12, 4, 23; 8, 1, 1]
my_matrix =
8 12 19
7 3 2
12 4 23
8 1 1

- To access the value of 4 you would type in
- my_matrix(3,2)
  ans = 4
- **Note:** The row number is first, followed by the column number.
- **The colon (:**) operator means ‘ALL’.
- You can also extract any contiguous subset of a matrix, by referring to the row range and column range you want. For example, if mat is a matrix with 5 rows and 8 columns, then typing
- mat(2:4,4:7) would extract all elements in rows 2 to 4 and in columns 4 to 7.

>> c = [1 3 6; 2 7 9; 4 3 1]

c =

1	3	6
2	7	9
4	3	1

- >>c(3,1) is the element in the third row, 1st column, which is 4.
- >>c(2:3,1:2) gives you the elements in rows 2-3, and columns 1-2, so you will get
  2 7
  4 3
- >>c(1:3,2) gives you the elements in rows 1-3, and the second column, that is, the entire second column. Which is:

  3
  7
  3

  You can shortcut this to:

  >>c(:,2)

  Literally means instructing matlab to use the 2nd column.

- You can get a whole row of a matrix with
  >>c(1,:)
  This literally instructs matlab to take the first row, all columns

- >> B = [1 2 3;4 5 6;7 8 9]

  B =

1	2	3
4	5	6
7	8	9

- >>B(2,2) =5
- >>B(:,2) =

  ans =

  2

  5

  8

- >>B([1 3],:) % selects the first and third rows, all the columns.

  ans =

  1 2 3

  7 8 9

- U= B([3 2 1],:) % reverses the rows of B

  U =

  7 8 9

  4 5 6

  1 2 3

Ends as a Subscript

To access the last element of a matrix along a given dimension, use **end** as a subscript .This allows you to go to the final element without knowing in advance how big the matrix is. For example:

>> q=4:10

q =

4 5 6 7 8 9 10

>> q(end)

ans =

10

q(end-4:end)

ans =

6 7 8 9 10

3.6.1 Deleting Rows or Columns

To get rid of a row or column set it equal to the empty matrix [].

a=[1 2 3;4 5 6;7 8 9]

a =

1 2 3

4 5 6

7 8 9

>> a(:,2)=[] % deletes the all the rows the second column.

a =

1 3

4 6

7 9

Using MATLAB built-in functions to Concatenation matrices

The functions to be used are vertcat and horzcat.

1. **vertcat**

Concatenate arrays vertically

Syntax

```
C = vertcat(A1, A2, ...)
```

Description

`C = vertcat(A1, A2, ...)` vertically concatenates matrices `A1`, `A2`, and so on. All matrices in the argument list must have the same number of columns.

`vertcat` concatenates N-dimensional arrays along the first dimension. The remaining dimensions must match.

MATLAB calls `C = vertcat(A1, A2, ...)` for the syntax `C = [A1; A2; ...]` when any of `A1`, `A2`, etc. is an object.

Examples

Create a 5-by-3 matrix, `A`, and a 3-by-3 matrix, `B`. Then vertically concatenate `A` and `B`.

```
A = magic(5);           % Create 5-by-3 matrix, A
A(:, 4:5) = []

A =

    17    24     1
    23     5     7
     4     6    13
    10    12    19
    11    18    25


B = magic(3)*100        % Create 3-by-3 matrix, B

B =

   800   100   600
   300   500   700
```

```
   400   900   200
```

```
C = vertcat(A,B)         % Vertically concatenate A and B
```

```
C =

    17    24     1
    23     5     7
     4     6    13
    10    12    19
    11    18    25
   800   100   600
   300   500   700
   400   900   200
```

2. horzcat

Concatenate arrays horizontally

Syntax

```
C = horzcat(A1, A2, ...)
```

Description

`C = horzcat(A1, A2, ...)` horizontally concatenates matrices `A1`, `A2`, and so on. All matrices in the argument list must have the same number of rows.

`horzcat` concatenates N-dimensional arrays along the second dimension. The first and remaining dimensions must match.

MATLAB calls `C = horzcat(A1, A2,...)` for the syntax `C = [A1 A2 ...]` when any of `A1`, `A2`, etc., is an object.

Examples

Create a 3-by-5 matrix, `A`, and a 3-by-3 matrix, `B`. Then horizontally concatenate `A` and `B`.

```
A = magic(5);            % Create 3-by-5 matrix, A
A(4:5,:) = []
```

```
A =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
```

```
B = magic(3)*100       % Create 3-by-3 matrix, B

B =

   800   100   600
   300   500   700
   400   900   200


C = horzcat(A, B)      % Horizontally concatenate A and B

C =

    17    24     1     8    15   800   100   600
    23     5     7    14    16   300   500   700
     4     6    13    20    22   400   900   200
```

Special Matrices in Matlab

1. eye

Identity matrix

Syntax

```
Y = eye(n)
Y = eye(m,n)
eye([m n])
Y = eye(size(A))
eye(m, n, classname)
eye([m,n],classname)
```

Description

`Y = eye(n)` returns the `n`-by-`n` identity matrix.

`Y = eye(m,n)` or `eye([m n])` returns an `m`-by-`n` matrix with `1`'s on the diagonal and `0`'s elsewhere.

`Y = eye(size(A))` returns an identity matrix the same size as `A`.

`eye(m, n, classname)` or `eye([m,n],classname)` is an m-by-n matrix with 1's of class `classname` on the diagonal and zeros of class `classname` elsewhere.

`classname` can have the following values: `'double'`, `'single'`, `'int8'`, `'uint8'`, `'int16'`, `'uint16'`, `'int32'`, `'uint32'`, `'int64'`, or `'uint64'`.

Example:

- eye (3)

  ans =

  1 0 0

  0 1 0

  0 0 1

  eye(3,4)

  ans =

  1 0 0 0

  0 1 0 0

  0 0 1 0

  x = eye(2,3,'int8')

  x =

  1 0 0

  0 1 0

2. zeros

Create array of all zeros

Syntax

```
B = zeros(n)
B = zeros(m,n)
B = zeros([m n])
B = zeros(m,n,p,...)
B = zeros([m n p ...])
B = zeros(size(A))
zeros(m, n,...,classname)
zeros([m,n,...],classname)
```

Description

`B = zeros(n)` returns an `n`-by-`n` matrix of zeros. An error message appears if `n` is not a scalar.

`B = zeros(m,n)` or `B = zeros([m n])` returns an `m`-by-`n` matrix of zeros.

Example

```
s=zeros(3)          % 3x3 matrix


s =


     0     0     0

     0     0     0

     0     0     0
```

3. ones

Create array of all ones

Syntax

```
Y = ones(n)
Y = ones(m,n)
Y = ones([m n])
Y = ones(m,n,p,...)
Y = ones([m n p ...])
Y = ones(size(A))
ones(m, n,...,classname)
ones([m,n,...],classname)
```

Description

`Y = ones(n)` returns an `n`-by-`n` matrix of `1`s. An error message appears if `n` is not a scalar.

`Y = ones(m,n)` or `Y = ones([m n])` returns an `m`-by-`n` matrix of ones.

`Y = ones(m,n,p,...)` or `Y = ones([m n p ...])` returns an `m`-by-`n`-by-`p`-by-`...` array of `1`s.

> **Note** The size inputs `m`, `n`, `p`, `...` should be nonnegative integers. Negative integers are treated as 0.

`Y = ones(size(A))` returns an array of `1`s that is the same size as `A`.

`ones(m, n,...,classname)` or `ones([m,n,...],classname)` is an `m`-by-`n`-by-... array of ones of data type `classname`. `classname` is a string specifying the data type of the output. `classname` can have the following values: `'double'`, `'single'`, `'int8'`, `'uint8'`, `'int16'`, `'uint16'`, `'int32'`, `'uint32'`, `'int64'`, or `'uint64'`.

Example

- ones(2,4)

  ans =

  1 1 1 1

  1 1 1 1

5. rand

Uniformly distributed pseudorandom numbers

Syntax

```
r = rand(n)
rand(m,n)
rand([m,n])
rand(m,n,p,...)
rand([m,n,p,...])
rand
rand(size(A))
r = rand(..., 'double')
r = rand(..., 'single')
```

Description

`r = rand(n)` returns an `n`-by-`n` matrix containing pseudorandom values drawn from the standard uniform distribution on the open interval (0,1). `rand(m,n)` or `rand([m,n])` returns an `m`-by-`n` matrix. `rand(m,n,p,...)` or `rand([m,n,p,...])` returns an `m`-by-`n`-by-`p`-by-... array. `rand` returns a scalar. `rand(size(A))` returns an array the same size as **`A`**.

`r = rand(..., 'double')` or `r = rand(..., 'single')` returns an array of uniform values of the specified class.

> **Note** Note: The size inputs `m`, `n`, `p`, ... should be nonnegative integers. Negative integers are treated as 0.

Examples

rand(3)

ans =

0.8147	0.9134	0.2785
0.9058	0.6324	0.5469
0.1270	0.0975	0.9575

6. randn

Normally distributed pseudorandom numbers

Syntax

```
r = randn(n)
randn(m,n)
randn([m,n])
randn(m,n,p,...)
randn([m,n,p,...])
randn(size(A))
```

```
r = randn(..., 'double')
r = randn(..., 'single')
```

Description

`r = randn(n)` returns an `n`-by-`n` matrix containing pseudorandom values drawn from the standard normal distribution. `randn(m,n)` or `randn([m,n])` returns an `m`-by-`n` matrix. `randn(m,n,p,...)` or `randn([m,n,p,...])` returns an `m`-by-`n`-by-`p`-by-... array. `randn` returns a scalar. `randn(size(A))` returns an array the same size as `A`.

`r = randn(..., 'double')` or `r = randn(..., 'single')` returns an array of normal values of the specified class.

> **Note** The size inputs `m`, `n`, `p`, ... should be nonnegative integers. Negative integers are treated as 0.

Examples

```
randn(4)


ans =


    2.7694   -0.0631    1.4897   -1.2075
   -1.3499    0.7147    1.4090    0.7172
    3.0349   -0.2050    1.4172    1.6302
    0.7254   -0.1241    0.6715    0.4889
```


magic

Magic square

Syntax

```
M = magic(n)
```

Description

`M = magic(n)` returns an `n`-by-`n` matrix constructed from the integers `1` through `n^2` with equal row and column sums. The order `n` must be a scalar greater than or equal to `3`.

Remarks

A magic square, scaled by its magic sum, is doubly stochastic.

Examples

The magic square of order 3 is

```
M = magic(3)

M =

    8    1    6
    3    5    7
    4    9    2
```

This is called a magic square because the sum of the elements in each column is the same.

```
 sum(M) =

     15    15    15
```

And the sum of the elements in each row, obtained by transposing twice, is the same.

```
 sum(M')' =

     15
     15
     15
```

This is also a special magic square because the diagonal elements have the same sum.

```
sum(diag(M)) =

     15
```

MATRIX ARITHMETIC

MATLAB treats arithmetic operations on arrays in an element-by-element manner. This operation is accomplished by including a dot or a period before the arithmetic operator such as

multiplication, division, etc. The table below gives a list of such operators with examples of how these operators are being used.

For examples in the table below, the following matrices are used:

```
A =

     1     2     3
     4     5     6
     7     8     0
B =

     2     4     6
     0     3     7
     9     8     1
```

Operation	Description	Example (MATLAB actual inputs and outputs)

+	Addition	`>> C=A+B`
		
		`C =`
		
		`3 6 9`
		`4 8 13`
		`16 16 1`

-	Substraction	`>> C=A-B`
		
		`C =`
		
		`-1 -2 -3`
		`4 2 -1`
		`-2 0 -1`

*	Multiplication	`>> C=A*B`
		
		`C =`
		
		`29 34 23`
		`62 79 65`
		`14 52 98`

.*	Element-by-element multiplication.
	Note that this is different from multiplication of two matrices.
		`» C=A.*B`
		
		`C =`
		
		`2 8 18`
		`0 15 42`
		`63 64 0`

		Note that this is not the same as **C = A*B**
/	Right matrix division. Dividing matrix **B** into matrix **A**	>> C=A/B C = 0.5000 0 0 3.6875 -2.2500 -0.3750 -8.3125 6.7500 2.6250
\	Left matrix division. Dividing **A** into **B**. This is equivalent to **inv(A)*B**. Note that **X** = **C** is the solution to **A***X=**B**	>> C=A\B C = -4.5556 -5.3333 -4.5556 5.1111 5.6667 4.1111 -1.2222 -0.6667 0.7778 >> X=inv(A)*B X = -4.5556 -5.3333 -4.5556 5.1111 5.6667 4.1111 -1.2222 -0.6667 0.7778
./	Element-by-element division note that **D**(2,1) is undefined due to the zero at **B**(2,1)	» D=A./B Warning: Divide by zero. D = 0.5000 0.5000 0.5000 Inf 1.6667 0.8571 0.7778 1.0000 0
.\	Element-by-element left division.Note that left division in this particular example means elements of B divided by the corresponding elements of A.	» E=A.\B Warning: Divide by zero. E = 2.0000 2.0000 2.0000 0 0.6000 1.1667 1.2857 1.0000 Inf

.^	Element-by-element power	» F=A.^B F = 1 16 729 1 125 279936 40353607 16777216 0

Transposing a matrix in MATLAB involves a simple prime notation (') after the defined matrix as shown below:

Example:

```
>> A_transpose=A'

A_transpose =

   1   4   7
   2   5   8
   3   6   0
```

Sorting columns and rows follow the syntax: ***B=sort(****A,dim****)***, where *dim* is the dimension of the matrix with the value 1 for column; 2 for row. Matrix A is the variable specified by the user.

Example:

Sorting columns:

```
>> sort(A)

ans =

   1   2   0
   4   5   3
   7   8   6
```

Note that without *dim* being specified, the default value is 1. The default setting is ascending order. The variable name of the sorted matrix can be omitted if no needed.

Sorting column in descending order:

```
>> sort(A,1,'descend')

ans =

   7   8   6
   4   5   3
   1   2   0
```

Sorting row in descending order

```
>> sort(A,2,'descend')

ans =

   3   2   1
   6   5   4
   8   7   0
```

The inverse of matrix **A** can be obtained with the command:

```
>> inv(A)

ans =

  -1.7778   0.8889  -0.1111
   1.5556  -0.7778   0.2222
  -0.1111   0.2222  -0.1111
```

Eigenvalues and Eigenvectors can easily be obtained with the command **[V,E]=eig(***matrix name***)**:

```
>> [V,E]=eig(A)

V =

  -0.2998  -0.7471  -0.2763
  -0.7075   0.6582  -0.3884
  -0.6400  -0.0931   0.8791


E =

  12.1229       0       0
        0  -0.3884       0
        0       0  -5.7345
```

where matrix V consists of Eigenvectors . The corresponding Eigenvalues are shown in matrix E. Eigen values alone can be obtained without the notation "[V,E]":

```
>> E=eig(A)

E =

  12.1229
  -0.3884
  -5.7345
```

where matrix V consists of Eigenvectors . The corresponding Eigenvalues are shown in matrix E. Eigen values alone can be obtained without the notation "[V,E]":

```
>> E=eig(A)

E =

  12.1229
  -0.3884
  -5.7345
```