

## MATLAB Tutorial

### Chapter 1. Basic MATLAB commands

#### 1.1 Basic scalar operations

First, let's talk about how we add comments (such as this line) to a program. Comments are lines of text that we want to add to explain what we are doing, so that if we or others read this code later, it will be easier to figure out what the code is doing. In a MATLAB file, if a percentage sign, %, appears in a row of text, all of the text following the sign is a comment that MATLAB does not try to interpret as a command. First, let us write a message to the screen to say that we are beginning to run section 1.1.

The command `disp('string')` displays the text string to the screen.

```
disp('Beginning section 1.1 ...')
```

Next, we set a variable equal to one.

```
x=1
```

This command both allocates a space in memory for the variable `x`, if `x` has not already been declared, and then stores the value of 1 in the memory location associated with this variable. It also writes to the screen "`x = 1`". Usually, we do not want to clutter the screen with output such as this, so we can make the command "invisible" by ending it with a semi-colon. As an example, let us use the following commands to "invisibly" change the value of `x` to 2 and then to write out the results to the screen. `x=2`; this changes the value of `x` but does not write to the screen `disp('We have changed the value of x.')`;

Then, we display the value of `x` by typing "`x`" without a semi-colon.

```
x
```

Now, let's see how to declare other variables.

```
y = 2*x; This initializes the value of y to twice that of x
```

```
x = x + 1; This increases the value of x by 1.
```

```
z = 2*x; This declares another variable z.
```

`z` does not equal `y` because the value of `x` changed between the times when we declared each variable.

```
difference = z - y
```

Next, we want to see the list of variables that are stored in memory. To do this, we use the command "`who`".

```
who;
```

We can get more information by using "`whos`".

```
whos;
```

These commands can be used also to get information about only certain variables.

```
whos z difference;
```

Let us say we want to get rid of the variable "`difference`".

We do this using the command "`clear`".

```
clear difference;
```

```
who;
```

Next, we want to get rid of the variables `x` and `y`.

Again, we use the command "`clear`".

```
clear x y;  
who;
```

It is generally good programming style to write only one command per line; however, MATLAB does let you put multiple commands on a line.

```
x = 5; y = 13; w = 2*x + y; who;
```

More commonly one wishes to continue a single command across multiple lines due to the length of the syntax. This can be accomplished by using three dots.

```
z = 2*x + ...  
y
```

Finally, when using clear we can get rid of all of the variables at once with the command "clear all".

```
clear all;  
who; It does not print out anything because there are no variables.
```

## 1.2. Basic vector operations

The simplest, but NOT RECOMMENDED, way to declare a variable is by entering the components one-by-one.

```
x(1) = 1;  
x(2) = 4;  
x(3) = 6;  
x display contents of x
```

It is generally better to declare a vector all at once, because then MATLAB knows how much memory it needs to allocate from the start. For large vectors, this is much more efficient.

```
y = [1 4 6] does same job as code above
```

Note that this declares a row vector. To get a column vector, we can either use the transpose (adjoint for complex  $x$ ) operator  $\mathbf{xT} = \mathbf{x}'$ ; takes the transpose of the real row vector  $x$  or, we can make it a column vector right from the beginning

```
yT = [1; 4; 6];
```

To see the difference in the dimensions of a row vs. a column vector, use the command "size" that returns the dimensions of a vector or matrix.

```
size(xT)  
size(y)  
size(yT)
```

The command length works on both row and column vectors.

```
length(x), length(xT)
```

Adding or subtracting two vectors is similar to scalars.

```
z = x + y  
w = xT - yT
```

Multiplying a vector by a scalar is equally straight-forward.

```
v = 2*x  
c = 4;  
v2 = c*x
```

We can also use the `.` operator to tell MATLAB to perform a given operation on an element-by-element basis. Let us say we want to set each value of  $y$  such that  $y(i) = 2*x(i) + z(i)^2 + 1$ . We can do this using the code

```
y = 2.*x + z.^2 + 1
```

The dot and cross products of two vectors are calculated by

```
dot(x,y)  
z=cross(x,y)
```

We can define a vector also using the notation [a : d : b]. This produces a vector a, a + d, a + 2\*d, a + 3\*d, ... until we get to an integer n where a + n\*d > b. Look at the two examples.

```
v = [0 : 0.1 : 0.5];  
v2 = [0 : 0.1 : 0.49];
```

If we want a vector with N evenly spaced points from a to b, we use the command "linspace(a,b,N)".

```
v2 = linspace(0,1,5)
```

Sometimes, we will use a vector later in the program, but want to initialize it at the beginning to zero and by so doing allocate a block of memory to store it. This is done by

```
v = linspace(0,0,100)'; allocate memory for column vectors of zero
```

Finally, we can use integer counting variables to access one or more elements of a matrix.

```
v2 = [0 : 0.01 : 100];  
c=v2(49)  
w = v2(65:70)
```

```
clear all
```

### 1.3. Basic matrix operations

We can declare a matrix and give it a value directly.

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

We can use commas to separate the elements on a line as well.

```
B = [1,2,3; 4,5,6; 7,8,9]
```

We can build a matrix from row vectors

```
row1 = [1 2 3]; row2 = [4 5 6]; row3 = [7 8 9];  
C = [row1; row2; row3]
```

or from column vectors.

```
column1 = [1; 4; 7];  
column2 = [2; 5; 8];  
column3 = [3; 6; 9];  
D = [column1 column2 column3]
```

Several matrices can be joined to create a larger one.

```
M = [A B; C D]
```

We can extract row or column vectors from a matrix.

```
row1 = C(1,:)  
column2 = D(:,2)
```

Or, we make a vector or another matrix by extracting a subset of the elements.

```
v = M(1:4,1)  
w = M(2,2:4)  
C = M(1:4,2:5)
```

The transpose of a real matrix is obtained using the ' operator

```
D = A'  
C, C'
```

For a complex matrix, ' returns the adjoint (transpose and conjugate. The conjugation operation is removed by using the "transpose only" command .'.

```
E = D;  
E(1,2) = E(1,2) + 3*i;  
E(2,1) = E(2,1) - 2*i;  
E', E.'
```

The "who" command lists the matrices in addition to scalar and vector variables.

```
who
```

If in addition we want to see the dimensions of each variable, we use the "whos" command. This tells us the size of each variable and the amount of memory storage that each requires.

```
whos
```

The command "size" tells us the size of a matrix.

```
M = [1 2 3 4; 5 6 7 8; 9 10 11 12];  
size(M)  
num_rows = size(M,1)  
num_columns = size(M,2)
```

Adding, subtracting, and multiplying matrices is straight-forward.

```
D = A + B  
D = A - B  
D = A*B
```

We can declare matrices in a number of ways.

We can create a matrix with m rows and n columns, all containing zeros by

```
m=3; n=4;  
C = zeros(m,n)
```

If we want to make an N by N square matrix, we only need to use one index.

```
C = zeros(n)
```

We create an Identity matrix, where all elements are zero except for those on the principle diagonal, which are one.

```
D = eye(5)
```

Finally, we can use the . operator to perform element-by-element operations just as we did for vectors. The following command creates a matrix C, such that  $C(i,j) = 2*A(i,j) + (B(i,j))^2$ .

```
C = 2.*A + B.^2
```

Matrices are cleared from memory along with all other variables.

```
clear A B  
whos  
clear all  
who
```

## 1.4. Using character strings

In MATLAB, when we print out the results, we often want to explain the output with text. For this, character strings are useful. In MATLAB, a character string is written with single quotation marks on each end.

```
course_name = 'Numerical Methods Applied to Chemical Engineering'
```

To put an apostrophe inside a string, we repeat it twice to avoid confusing it with the ' operator ending the string.

```
phrase2 = 'Course''s name is : ';  
disp(phrase2), disp(course_name)
```

We can also combine strings in a similar manner to working with vectors and matrices of numbers.

```
word1 = 'Numerical'; word2 = 'Methods'; word3='Course';  
phrase3 = [word1, word2, word3]
```

We see that this does not include spaces, so we use instead

```
phrase4 = [word1, ' ', word2, ' ', word3]
```

We can convert an integer to a string using the command "int2str".

```
icount = 1234;  
phrase5 = ['Value of icount = ', int2str(icount)]
```

Likewise, we can convert a floating point number of a string of k digits using "num2str(number,k)".

```
Temp = 29.34372820092983674;  
phrase6 = ['Temperature = ', num2str(Temp,5)]  
phrase7 = ['Temperature = ', num2str(Temp,10)]
```

```
clear all
```

## 1.5. Basic mathematical operations

### EXPONENTIATION COMMANDS

We have already seen how to add, subtract, and multiply numbers. We have also used on occasion the ^ operator where  $x^y$  raises x to the power y.

```
2^3, 2^3.3, 2.3^3.3, 2.3^(1/3.3), 2.3^(-1/3.3)
```

The square root operation is given its own name.

```
sqrt(27), sqrt(37.4)
```

Operators for use in analyzing the signs of numbers include

```
abs(2.3), abs(-2.3) returns absolute value of a number  
sign(2.3), sign(-2.3), sign(0) returns sign of a number
```

The commands for taking exponents and logs are

```
a=exp(2.3) computes  $e^x$   
log(a) computer the natural log  
log10(a) computes the base 10 log
```

### TRIGONOMETRY COMMANDS

The numerical value of pi can be invoked directly

```
pi, 2*pi
```

NOTE THAT MATLAB CALCULATES ANGLES IN RADIANS

The standard trigonometric functions are  
**sin(0), sin(pi/2), sin(pi), sin(3\*pi/2)**  
**cos(0), cos(pi/2), cos(pi), cos(3\*pi/2)**  
**tan(pi/4), cot(pi/4), sec(pi/4), csc(pi/4)**

Their inverses are  
**asin(1), acos(1), atan(1), acot(1), asec(1), acsc(1)**

The hyperbolic functions are  
**sinh(pi/4), cosh(pi/4), tanh(pi/4), coth(pi/4)**  
**sech(pi/4), csch(pi/4)**  
with inverses  
**asinh(0.5), acosh(0.5), atanh(0.5), acoth(0.5)**  
**asech(0.5), acsch(0.5)**

These operators can be used with vectors in the following manner.  
**x=linspace(0,pi,6)** create vector of x values between 0 and pi  
**y=sin(x)** each y(i) = sin(x(i))

#### ROUNDING OPERATIONS

round(x) : returns integer closest to real number x  
**round(1.1), round(1.8)**

fix(x) : returns integer closest to x in direction towards 0  
**fix(-3.1), fix(-2.9), fix(2.9), fix(3.1)**

floor(x) : returns closest integer less than or equal to x  
**floor(-3.1), floor(-2.9), floor(2.9), floor(3.1)**

ceil(x) : returns closest integer greater than or equal to x  
**ceil(-3.1), ceil(-2.9), ceil(2.9), ceil(3.1)**

rem(x,y) : returns the remainder of the integer division x/y  
**rem(3,2), rem(898,37), rem(27,3)**

mod(x,y) : calculates the modulus, the remainder from real division  
**mod(28.36,2.3)**

#### COMPLEX NUMBERS

A complex number is declared using i (or j) for the square root of -1.

**z = 3.1-4.3\*i**

**conj(z)** returns conjugate, conj(a+ib) = a - ib

**real(z)** returns real part of z, real(a+ib) = a

**imag(z)** returns imaginary part of z, imag(a+ib) = b

**abs(z)** returns absolute value (modulus),  $a^2+b^2$

**angle(z)** returns phase angle theta with  $z = r*\exp(i*theta)$

**abs(z)\*exp(i\*angle(z))** returns z

For complex matrices, the operator ' calculates the adjoint matrix, i.e. it transposes the matrix and takes the conjugate of each element

**A = [1+i, 2+2\*i; 3+3\*i, 4+4\*i]**

**A'** takes conjugate transpose (adjoint operation)

**A.'** takes transpose without conjugating elements

## COORDINATE TRANSFORMATIONS

2-D polar coordinates (theta,r) are related to Cartesian coordinates by

**x=1; y=1;**

**[theta,r] = cart2pol(x,y)**

**[x,y] = pol2cart(theta,r)**

3-D spherical coordinates (alpha,theta,r) are obtained from Cartesian coordinates by

**x=1; y=1; z=1;**

**[alpha,theta,r] = cart2sph(x,y,z)**

**[x,y,z] = sph2cart(alpha,theta,r)**

**clear all**