TOPIC 3
Computer application in Basic mathematics function using MATLAB

Fundamental Expressions/Operations

MATLAB uses conventional decimal notion, builds expressions with the usual arithmetic operators and precedence rules:

```
» x = 3.421
x =
3.4210
» y = x+8.2i
y =
3.4210 + 8.2000i
» z = sqrt(y)
z =
2.4805 + 1.6529i
» p = sin(pi/2)
p =
1
```

Numbers & Formats in MATLAB

Matlab recognizes several different kinds of numbers

Type	Examples
Integer	1362;-217897
Real	1.234;-10.76
Complex	3.21 – 4.3i (i = p√1)
Inf	Infinity (result of dividing by 0)
NaN	Not a Number, 0=0

The “e" notation is used for very large or very small numbers:

-1.3412e+03 = -1.3412×10^{3} = -1341.2

-1.3412e-01 = -1.3412×10^{-1} = -0.13412

All computations in MATLAB are done in double precision, which means about 15 significant figures.

The format -how Matlab prints numbers- is controlled by the “format" command. Type

>>help format

for full list. Should you wish to switch back to the default format then format will suffice.

The command

>>format compact

is also useful in that it suppresses blank lines in the output thus allowing more information to be displayed.

Command	Example of Output
>>format short	31.4162(4–decimal places)
>>format short e	3.1416e+01
>>format long e	3.141592653589793e+01
>>format short	31.4162(4–decimal places)
>>format bank	31.42(2–decimal places)

Keeping and retrieving a data

Issuing the command
>> diary mysession
will cause all subsequent text that appears on the screen to be saved to the file mysession located in the directory in which Matlab was invoked. You may use any legal filename except the names on and off. The record may be terminated by

>> diary off
The file mysession may be edited with your favorite editor (the Matlab editor, or MS Word) to remove any mistakes.

If you wish to quit Matlab midway through a calculation so as to continue at a later stage:
>> save thissession
will save the current values of all variables to a file called **thissession.mat**. This file cannot be edited.

When you next startup Matlab, type
>> load thissession
and the computation can be resumed where you left off. A list of variables used in the current session may be seen with
>> whos

Name	Size	Bytes	Class
p	1x1	8	double array
x	1x1	8	double array
y	1x1	16	double array (complex)
z	1x1	16	double array (complex)

Grand total is 4 elements using 48 bytes

Basic Mathematical Functions

Matlab knows all of the standard functions found on scientific calculators and elementary mathematics. They categorized in Trigonometric, Exponential, Complex, Rounding and Remainder, and Discrete Math functions.

Here's the list of function names that Matlab knows about. You can use online help to find details about how to use them.

Trigonometric Functions

Those known to Matlab are sin, cos, tan and their arguments should be in radians. e.g. to work out the coordinates of a point on a circle of radius 5 centered at the origin and having an elevation 30^o = pi/6 radians:

```
>> x = 5*cos(pi/6), y = 5*sin(pi/6)
x =
4.3301
y =
2.5000
```

In MATLAB, it makes sense to take the sine of an array: the answer is just an array of sine values, e.g.,
sin([pi/4,pi/2,pi])= [0.7071 1.0000 0.0000]

The inverse trig functions are called asin, acos, atan (as opposed to the usual arcsin or sin√1 etc.). The result is in radians.

```
>> acos(x/5), asin(y/5)
ans = 0.5236
ans = 0.5236
>> pi/6
ans = 0.5236
```

The other functions are:

Acos	Inverse cosine	cos	Cosine
Acosd	Inverse cosine, degrees	cosd	Cosine, degrees
Acosh	Inverse hyperbolic cosine	cosh	Hyperbolic cosine
Acot	Inverse cotangent	cot	Cotangent
Acotd	Inverse cotangent, degrees	cotd	Cotangent, degrees
Acoth	Inverse hyperbolic cotangent	coth	Hyperbolic cotangent
Acsc	Inverse cosecant	csc	Cosecant
Acscd	Inverse cosecant, degrees	cscd	Cosecant, degrees
Acsch	Inverse hyperbolic cosecant	csch	Hyperbolic cosecant
Asec	Inverse secant	sec	Secant
Asecd	Inverse secant, degrees	secd	Secant, degrees
Asech	Inverse hyperbolic secant	sech	Hyperbolic secant
Asin	Inverse sine	sin	Sine
Asind	Inverse sine, degrees	sind	Sine, degrees
Asinh	Inverse hyperbolic sine	sinh	Hyperbolic sine
atan	Inverse tangent	tan	Tangent

atand	Inverse tangent, degrees	tand	Tangent, degrees
atanh	Inverse hyperbolic tangent	tanh	Hyperbolic tangent
atan2	Four-quadrant inverse tangent		

Detail information and examples of the functions can be obtained using command :
>>help name_of_function

Exponential

The exp function is an elementary function that operates element-wise on arrays. Its domain includes complex numbers. **Y = exp(x)** returns the exponential for each element of **x,** denotes the exponential function **exp(x) = e^{x}** and the inverse function is **log**.

>> format long e, exp(log(9)), log(exp(9))
ans = 9.000000000000002e+00
ans = 9

>> format short
and we see a tiny rounding error in the first calculation. log10 gives logs to the base 10.
A more complete list of elementary functions is given below.

exp	Exponential
expm1	Exponential of x minus 1
log	Natural logarithm
log1p	Logarithm of 1+x
log2	Base 2 logarithm and dissect floating-point numbers into exponent and mantissa
log10	Common (base 10) logarithm
nextpow2	Next higher power of 2
pow2	Base 2 power and scale floating-point number
reallog	Natural logarithm for nonnegative real arrays
realpow	Array power for real-only output
realsqrt	Square root for nonnegative real arrays
sqrt	Square root
nthroot	Real nth root

Complex

Complex numbers are formed in MATLAB in several ways. Examples of complex numbers include:

```
>> c=1-2i  % the appended I signifies the imaginary part
c =
   1.0000 - 2.0000i

>> c1=1-2j   %j also works
```

c1 =
   1.0000 - 2.0000i

The function abs computes the magnitude of complex numbers or the absolute value of real numbers. **abs(X)** returns an array Y such that each element of Y is the absolute value of the corresponding element of X.

If X is complex, abs(X) returns the complex modulus (magnitude), which is the same as **sqrt(real(X).^2 + imag(X).^2)**

```
>>abs(-5)
ans =
     5

>>abs(3+4i)
ans =
     5
```

The other functions to handle complex numbers in MATLAB are given below.

abs	Absolute value
angle	Phase angle
complex	Construct complex data from real and imaginary parts
conj	Complex conjugate
cplxpair	Sort numbers into complex conjugate pairs
i	Imaginary unit
imag	Complex imaginary part
isreal	True for real array
j	Imaginary unit
real	Complex real part
sign	Signum
unwrap	Unwrap phase angle

Rounding and Remainder

There are a variety of ways of rounding and chopping real numbers to give integers. Use the following definitions in order to understand the output given below:

fix	Round towards zero
floor	Round towards minus infinity
ceil	Round towards plus infinity
round	Round towards nearest integer
mod	Modulus after division
rem	Remainder after division

```
>> x = pi*(-1:3), round(x)
x =
-3.1416   0   3.1416   6.2832   9.4248
```

ans =
-3 0 3 6 9
>> fix(x)
ans =
-3 0 3 6 9
>> floor(x)
ans =
-4 0 3 6 9
>> ceil(x)
ans =
-3 0 4 7 10
>> sign(x), rem(x,3)
ans =
-1 0 1 1 1
ans =
-0.1416 0 0.1416 0.2832 0.4248

Type **>>help round** in command window for further information.

Discrete Math (e.g., Prime Factors)

The function **factor(n)** returns a row vector containing the prime factors of n.

>>f = factor(123)
f =
3 41

MATLAB provides some useful functions to handle operation in discrete mathematics, which are:

factor	Prime factors
factorial	Factorial function
gcd	Greatest common divisor
isprime	True for prime numbers
lcm	Least common multiple
nchoosek	All combinations of N elements taken K at a time
perms	All possible permutations
primes	Generate list of prime numbers rat,
rats	Rational fraction approximation

Try to find lcm and gcd of three numbers: 42,72 and 144.

Tabulating Functions

Example: Tabulate the functions y = 4 sin 3x and u = 3 sin 4x for x = 0; 0:1; 0:2; …… ; 0:5.

```
>> x = 0:0.1:0.5;
>> y = 4*sin(3*x); u = 3*sin(4*x);
>> [ x' y' u']
ans =
         0         0         0
    0.1000    1.1821    1.1683
    0.2000    2.2586    2.1521
    0.3000    3.1333    2.7961
    0.4000    3.7282    2.9987
    0.5000    3.9900    2.7279
```

Note the use of transpose (') to get column vectors.
(we could replace the last command by [x; y; u;]')
We could also have done this more directly:

```
>> x = (0:0.1:0.5)';
>> [x 4*sin(3*x) 3*sin(4*x)]
```

Random Numbers

The function **rand(m,n)** produces an m x n matrix of random numbers, each of which is in the range 0 to 1. **rand** on its own produces a single random number.

```
>> y = rand, Y = rand(2,3)
y =
0.9191
Y =
0.6262 0.1575 0.2520
0.7446 0.7764 0.6121
```

Repeating these commands will lead to different answers.

Logical variables and functions

Logical variables: just two values zero and one (false and true). Ordinary real variable is considered as logical one if it is not equal to zero. Function logical converts real variables into logical variables. Complex numbers cannot be converted to logical.

```
» a=linspace(0,3,5)
a =
        0 0.7500 1.5000 2.2500 3.0000
» logical(a)
        Warning: Values other than 0 or 1 converted to logical 1
        (Type "warning off MATLAB:conversionToLogical" to suppress this warning.)
ans =
        0 1 1 1 1
» warning off MATLAB:conversionToLogical
» b=logical(a)
b =
        0 1 1 1 1
```

Functions that return logical scalars or vectors or operate on logical variables:

any	True if any element of vector is true.
all	True if all elements of vector are true.
find	Find indices of non-zero elements.
isnan	True for Not-A-Number.
isinf	True for infinite elements.
finite	True for finite elements.
Isempty	True for empty matrix.
isstr	True for text string.
isglobal	True for global variables.
isglobal	True for global variables.
isreal	Returns logical 0 if any element has an imaginary component, even if the value of that component is 0.
isa	Detect an object of a given MATLAB class

Examples:

```
» b
b =
0 1 1 1 1
» any(b)
ans =
1
» all(b)
ans =
0
» find(b)
ans =
2 3 4 5
» isempty(b)
ans =
0
» isstr(b)
ans =
0
» isnan(b)
ans =
0 0 0 0 0
» isstr('some string l,kksdKJGajsdHGF*&%^')
ans =
1
» isreal(b)
ans =
1
```

» isreal(b*i)
ans =
0
There are many functions that start with is. The description of these functions can be obtained by searching Matlab help index for is*. Learning about these functions is highly recommended.

Relational operators:
<, <=, >, >=, ==, ~= perform element by element comparison of two scalars/arrays/matrices (or array/matrix and scalar).
<, <=, >, >= operators test only real part for comparison, == and ~= test both.
For string comparison strcmp is used. (usage strcmp(st1,st2))
The result of these operations can be assigned to a variable. Examples:
» a=[1,2,3,4,5], b=fliplr(a)
a =
1 2 3 4 5
b =
5 4 3 2 1
» c=a<=b
c =
1 1 1 0 0
» d=a~=b
d =
1 1 0 1 1
» e=a==b
e =
0 0 1 0 0
» find(e)
ans =
3

Logical Operators:
&, |, ~ (and, or, not) work element by element on matrixes. ‘Exclusive OR’ is not an operator but function xor.
Examples below use logical variables created above:
» c&d
ans =
1 1 0 0 0
» c&e
ans =
0 0 1 0 0
» c|e
ans =
1 1 1 0 0

Combinatorics

Probability calculations often involve counting combinations of objects and the study of combinations of objects is the realm of combinatorics. Many formulas in combinatorics are derived simply by counting the number of objects in two different ways and setting the results equal to each other. Often the resulting proof is obtained by a "proof by words." Formulas are not necessarily derived from mathematical manipulations of factorial functions as some students might think. Some of MATLAB's combinatorial functions are illustrated in this section.

Permutations

Permutations arise when we choose r objects in succession from a population of n distinct objects (referred to as sampling without replacement). In this case, the number of possibilities is:

$$n(n\text{-}1)(n\text{-}2)\times\ \times\times(n\text{-}r\text{+}1)$$

If we sample with replacement of the objects, the number of possibilities is nr .
Permutations may also be thought of as the re-arrangement (i.e., permutation) of a set of objects. If there are n objects, then there are

$$n(n\text{-}1)(n\text{-}2)\times\times\times(1)=n!$$

different permutations of these n objects. The above formulas are likely familiar to you. MATLAB provides a **factorial** function:

```
>> factorial(20).
```

It is generally preferable (because it is quicker), however, to use the gamma function to calculate factorials. We apply the fact that G(n+1)=n! . For example,

```
>> factorial(5)
ans =
120
>> gamma(6)
ans =
120
```

Consider now the number of possible ways in which k objects can be chosen from a total of n objects, which is written $\binom{n}{k}$

We can figure out the formula for $\binom{n}{k}$ just by counting. We know that there are a total of n! permutations of the n objects. Now let us count the number of permutations of n objects in a different way. We first consider all combinations of k objects chosen from a total of n objects. In other words, one can divide the n objects into k and (n-k) objects. But we also know that there is a total of k! permutations of the k objects and a total of (n-k)! permutations of the (n-k) objects. Therefore, the total number of permutations of the n objects is

$$n! = \binom{n}{k} * k! * (n-k)!$$

Again, this formula is derived simply by counting, not by expanding factorial functions. The above formula can be re-expressed as

$$\binom{n}{k} = \frac{n!}{k!*(n-k)!}$$

MATLAB provides a function for calculating combinations (n=5, k=3):

>> nchoosek(5,3).

****

http://www.chem.duke.edu/~boris/matlab/Lesson_2.pdf

http://www.eelab.usyd.edu.au/ELEC2103/UserFiles/File/tute6.pdf