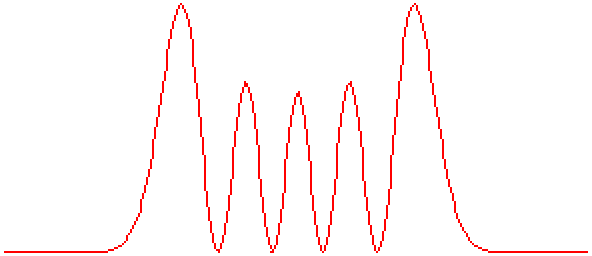
EP547 Computational Methods in QM

Topic 1

Basic MATLAB Tutorial



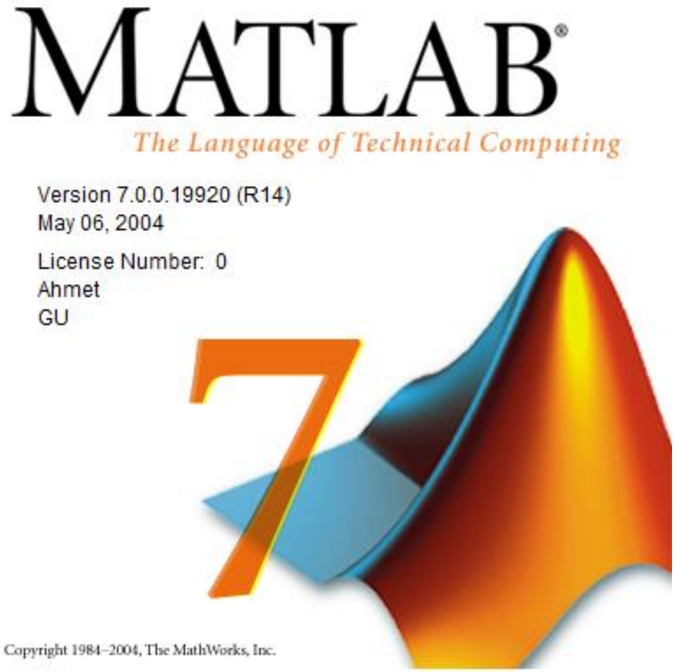
Department of Engineering Physics

University of Gaziantep

Feb 2013

Content

1. Basic Commands
2. Data Types
3. Variables
4. Arrays
5. Cells
6. Strings
7. Some Intrinsic Functions
8. Input / Output
9. Operators
10. Conditional Statements
11. Loops



1. Basic Commands

- **`help`** ***`command`***

  get help for a command

- **`clear all`**

  clears all the memory (workspace)

- **`clear x`**

  removes variable x from the memory

- **`whos`**

  lists all the variables (and details) on the workspace

- Semicolon (;) at the end will suppress the output

- Command history: upper & lower arrows, also command name guess

- If you don't use a variable name, your calculation is labelled and assigned by **`ans`** variable.

2. Data Types

- MATLAB data types are classes.

- Most commonly used types are
  - **`double`**
  - **`char`**
  - **`logical`**

- All of them are considered by MATLAB as arrays

- Numerical objects belong to the class **`double`** (i.e. double precision array).

- A scalar is treated as a 1 1 array.

3. Variables

- Variable names, which must start with an English Letter.

- MATLAB is case sensitive:
  `Food` and **`food`** are different variables

- There are several built-in constants and special variables:

`ans`	Default name for results
`eps`	Smallest number for which 1 + eps > 1
`inf`	Infinity
`NaN`	Not a number
`i` or **`j`**	$\sqrt{-1}$
`pi`	π

```
>> eps
ans = 2.2204e-016
```

```
>> pi
ans = 3.1416
```

```
>> x = 3 + 4i    % complex number
x = 3.0000 + 4.0000i
```

4. Arrays and Matrices

- Arrays can be constructed in several ways.

```
>> A = [5 -3 4 2]
A = 5     -3      4      2
```

```
>> A = [5, -3, 4, 2]
A = 5     -3      4      2
```

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

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

```
>> v = [1 2 3]   % row vector
v = 1   2   3
```

```
>> v = [1; 2; 3]   % column vector
v =
     1
     2
     3
```

```
>> v = [1 2 3]'   % transpose of a row vector
v =
     1
     2
     3
```

5. Cells

- A cell array is a sequence of arbitrary objects

```
>> c = {[1 2 3], 'Hello Everybody', 2 + 7i}
c =
  [1x3 double]  'Hello Everybody`   [2.0000+ 7.0000i]

>> c{1}       % first cell
ans =  1     2     3

>> c{1}(3) % third element of first cell
ans = 3
```

6. Strings (Character Arrays)

- String is a sequence of characters.

```
>> s1 = 'University ';
>> s2 = 'of Gaziantep';
>> s3 = strcat(s1,s2);
>> s3
s3 = University of Gaziantep

>> s1(1:5)
ans = Unive

>> s2(8:12)
ans = antep
```

7. Some Intrinsic Functions

Function	Description	
`abs(x)`	$\|x\|$	
`sin(x)`	sine of x	(*x is in radian*)
`cos(x)`	cosine of x	
`tan(x)`	tangent of x	
`sind(x)`	sine of x	(*x is in degrees*)
`cosd(x)`	cosine of x	
`tand(x)`	tangent of x	
`asin(x)`	angle in radian from $\sin^{-1}(x)$	
`acos(x)`	angle in radian from $\cos^{-1}(x)$	
`atan(x)`	angle in radian from $\tan^{-1}(x)$	
`log(x)`	ln(x)	
`log10(x)`	$\log_{10}(x)$	
`exp(x)`	e^x	
`mod(x,y)`	x modulo y	(mod(12,5) = 2)

8. Input / Output

User input

```
>> a = input('birinci sayi: ');
birinci sayi: 2
>> b = input('ikinci sayi: ');
ikinci sayi: 3
>> c = a + b
c = 5
```

```
>> p = input('input an array: ');
input an array: [1 2 3]
>> p'

ans =

     1
     2
     3
```

User output

- Normally MATLAB displays numerical results with about five digits, but this can be changed with the format command:

`format long`	16-digit display
`format short`	5-digit display

```
>> pi              % default format
ans = 3.1416

>> format long
>> pi
ans = 3.14159265358979

>> format short
>> pi
ans = 3.1416
```

- A simple way of printing values is to use **disp** function:

disp(*value*)

```
>> x = 10;
>> disp(x);
    10

>> y = [1 3 5 7];
>> disp(y)
     1     3     5     7

>> disp(y')
     1
     3
     5
     7

>> disp('University of Gaziantep')
University of Gaziantep
```

- To print formatted output, one can use **`fprintf`** function:

```
fprintf(format, list)
```

format contains formatting specifications

list is the list of items to be printed

Typical format specifiers are:

%*w***`d`**	Integer notation
%*w.d***`f`**	Floating point notation
%*w.d***`e`**	Exponential notation
`\n`	Newline character (line break)

where *w* is the width of the field and

d is the number of digits after the decimal point.

```
>> x = 123.456;
>> i = 1453;

>> fprintf('i = %d ve x = %f\n',i,x)
i = 1453 ve x = 123.456000

>> fprintf('i = %7d ve x = %10.2f\n',i,x)
i =    1453 ve x =     123.46
```

9. Arithmetic Operators

+	Addition	2 +4 = 6
–	Subtraction	2 – 4 = –2
*	Multiplication	2 * 4 = 8
/	Right division	2 / 4 = 0.5
\	Left division (*we'll see later*)	2 \ 4 = 2
^	Exponention (x^y)	2 ^ 4 = 16
.*	Element-wise multiplication	
./	Element-wise division	
.^	Element-wise exponention	

```
>> 3 + 5      % scalar addition

ans = 8
```

```
>> 3 ^ 5

ans = 243
```

```
>> 3 / 5

ans = 0.6000
```

```
>> 3 \ 5

ans = 1.6667
```

```
>> a = [1 2 3];
>> b = [-2 3 4];
>> a + b          % vector addition

ans =
    -1     5     7
```

```
>> a * b'      % dot product

ans = 16
```

```
>> a.*b        % element-wise product

ans =

    -2     6    12
```

```
>> A = [1 2 3; 4 5 6];
>> B = [7 8 9; 0 1 2];
>> A + B     % matrix addition

ans =
     8    10    12
     4     6     8
```

```
>> A * B'     % matrix product

ans =
     50     8
    122    17
```

```
>> A.*B      % element-wise product

ans =

     7    16    27
     0     5    12
```

- Comparison Operators

  - `<` Less than
  - `<=` Less than or equal to
  - `>` Greater than
  - `>=` Greater than or equa to
  - `==` Equal to
  - `~=` Not equal to

- Logical Operators

  - `&` AND
  - `|` OR
  - `~` NOT

10. Conditional Statements

```
if condition
   block
end
```

Executes the block of statements if the condition is true.

```
if condition_1
  block_1
elseif condition_2
  block_2
...
else
  block_n
end
```

Executes the block_i of if the condition_i is true.

Otherwise block_n is executed.

Note that the use of else statement is optional.

```
>> edit tekmi.m
```

```
a = input('input a: ');
if mod(a,2)==1
     fprintf('%d is odd number\n',a)
else
     fprintf('%d is even number\n',a)
end
```

```
>> tekmi
input a: 6
6 is even number
>>
```

```
>> edit quadratic.m
```

```
% MATLAB script to calculate the
% roots of f(x) = a x^2 + b x + c
a = input('Input a ');
b = input('Input b ');
c = input('Input c ');
Delta = b*b - 4*a*c;

x1 = ( -b - sqrt(Delta) )/(2*a);
x2 = ( -b + sqrt(Delta) )/(2*a);

if Delta < 0
  disp('Two imaginary roots:')
  x1, x2
elseif Delta > 0
  disp('Two real roots:')
  x1, x2
else
  disp('One real root:')
  x1
end
```

```
>> quadratic
Input a 1
Input b -2
Input c -8
Two real roots:
x1 = -2
x2 = 4

>> quadratic
Input a 1
Input b -4
Input c 4
One real root:
x1 = 2

>> quadratic
Input a 1
Input b 2
Input c 3
Two imaginary roots:
x1 = -1.0000 - 1.4142i
x2 = -1.0000 + 1.4142i
```

11. Loops

```
while condition

   block

end
```

Executes the block of statements if the condition is true.

After execution of the block, condition is evaluated again. If it is still true, the block is executed again. The loop is repeated until the condition becomes false.

```
>> edit loop1.m
```

```
k = 1;
while k<=6
  fprintf('%d  %d\n',k,k*k);
  k=k+1;
end
```

```
>> loop1
1   1
2   4
3   9
4   16
5   25
6   36
```

```
for target = sequence
    block
end
```

```
>> edit loop2.m
```

```
for n = 1:5
  y(n) = cos(n*pi/10);
end
disp(y')
```

```
>> loop2
0.9511
0.8090
0.5878
0.3090
0.0000
```

This is equivalent to:

```
>> n = 1:5;
>> y = cos(n*pi/10);
>> y
y =
0.9511   0.8090   0.5878   0.3090   0.0000
```

```
>> x = 0:0.4:2;
>> for i = 1:length(x)
     fprintf('%4.1f %11.6e\n',x(i),log(x(i)))
   end

Warning: Log of zero.
 0.0  -Inf
 0.4  -9.162907e-001
 0.8  -2.231436e-001
 1.2   1.823216e-001
 1.6   4.700036e-001
 2.0   6.931472e-001
```

Example: Write a program to find and list all integer (x,y) pairs satisfying the condition |x|+|y|<=3.

```
>> edit pairs.m
```

```
for x = -3:3
for y = -3:3
   if abs(x)+abs(y) <= 3
     fprintf('(%d  %d)\n',x,y)
   end
end
end
```

```
>> pairs
(-3  0)
(-2  -1)
(-2  0)
(-2  1)
(-1  -2)
...
```

break and **continue** Statements

Any loop can be exited by a **break** statement.

continue statement allows you to jump to the next iteration.

kes.m

```
for x = -5:5
  if x == 0
    break
  end
  fprintf('%f  %f\n',x,1/x)
end
```

```
>> kes
-5.000000  -0.200000
-4.000000  -0.250000
-3.000000  -0.333333
-2.000000  -0.500000
-1.000000  -1.000000
```

devam.m

```
for x = -5:5
  if x == 0
    continue
  end
   fprintf('%f  %f\n',x,1/x)
end
```

```
>> devam
-5.000000  -0.200000
-4.000000  -0.250000
-3.000000  -0.333333
-2.000000  -0.500000
-1.000000  -1.000000
1.000000  1.000000
2.000000  0.500000
3.000000  0.333333
...
```

References

[1]. http://www.mathworks.com/products/matlab

[2]. Numerical Methods in Engineering with MATLAB,
J. Kiusalaas, Cambridge University Press (2005)

[3]. Numerical Methods for Engineers, 6th Ed.
S.C. Chapra, Mc Graw Hill (2010)