1 INTRODUCTION

Contents

Initialization

Things to do before an interactive section. (Emacs users also set C-x f fill column to 57.)

```
% reduce needless whitespace
format compact
% reduce irritations
more off
% start a diary
%diary lecture1.txt
```

Basic computations

```
2+3
2-3
2*3
2/3
2^3
1.5/.5
disp('1.5/.5')
1.5/.5
```

```
ans =  5
ans = -1
ans =  6
ans =  0.66667
ans =  8
ans =  3
1.5/.5
ans =  3
```

Exponential notation

```
% exponential notation for Planck's constant
1.0546e-34
% or not
0.00000000000000000000000000000000010546
```

```
% square of Planck's constant
1.0546e-34^2
```

```
ans =      1.0546e-34
ans =      1.0546e-34
ans =      1.1122e-68
```

Comments

How to document your code.

```
% Lines starting with a single % are explanatory comments
% that Matlab ignores.
disp('Comments start with %')

% Lines starting with %% are also ignored, and act as
% section headers in the "published" output.
disp('Section headers start with %%')
```

```
Comments start with %
Section headers start with %%
```

Basic functions

```
% getting the square root of a number
sqrt(9)

% matlab (and all science) uses radians by default
sin(30)
sin(pi/180*30)

% avoid using degrees if not needed
sind(30)
```

```
ans =  3
ans = -0.98803
ans =  0.50000
ans =  0.50000
```

Variables

Variables are named storage locations.

```
Note:
        In a
                  VARIABLENAME=VALUE
        command, VALUE is evaluated *first*.
        Then that VALUE is stored in the variable
        with name VARIABLENAME.  If no variable
        with that name exists as yet, it is
        created.
```

```
Note:
                  VARIABLENAME=VALUE
             is ***not*** an equality
```

It is an assignment statement. It stores VALUE in VARIABLENAME. To test whether the number in VARIABLENAME is the same as VALUE, use instead VARIABLENAME==VALUE.

```
% there is no variable named 'x' yet (no response)
who x
% see also the workspace window

% The next statement (command) is *not* a question.  It
% tells matlab to create a variable named 'x', if it does
% not yet exist, (like now), and then put the value 3 in
% that storage location.

% create x and store 3 in it
x=3

% now we have a variable x
who x
whos x
% see also the workspace window

% we can print out its value by invoking its name
x

% with a final ;, the value is *not* printed
x;
x=3;
```

```
x =  3
Variables in the current scope:
x
Variables in the current scope:
```

```
  Attr Name          Size                          Bytes
       Class
  ==== ====          ====                          =====
       =====
         x           1x1                              8
              double
Total is 1 element using 8 bytes
x =   3
```

Computing with variables

```
% we can compute with x
x=x+7

% In the above statement, the right hand side is
% evaluated *first*.  Then the result, 10, is put in the
% storage location named "x".  The old value, 3, is
% *lost*.

% we can double x
x=x+x
% try using the Up-Arrow key a few times
```

```
x =   10
x =   20
```

Creating your own functions

Matlab provides a function sqrt(x) that 'returns' the square root of x. But suppose you would like a function sqr(x) that returns the square instead of square root of x.
The *general* way to do it is to write a function file. For our sqr function, the function file must be called sqr.m. A *minimal* example of the contents of that file is shown below:
Contents of sqr.m:

```
function x2 = sqr(x)

x2 = x*x;

end
```

```
% Let's test out our new function!
sqr(2)
```

```
sqr(3)
x=4
sqr(x)
y=sqr(x);
y
% seems to work OK
```

```
ans =   4
ans =   9
x =   4
ans =   16
y =   16
```

HOW TO DO HOMEWORKS

Demonstrated for hw0:

1. Open program "Secure Shell Client"
2. Select "Quick Connect"
3. Enter 'wolf' for Host (without the quotes)
4. Enter your COE username and then your password.
5. Enter '~dommelen/gethwN' where N is the hw number.
6. On success, enter 'exit' and close the Client.
7. Select the created hwN folder in Matlab.
8. Put the solution of question 1 into q1.m.
9. Try executing 'clear' and 'q1' in the command window.
10. If OK , enter 'publish q1.m pdf'
11. Print out the created q1.pdf in the html folder.
12. Same for the other questions.
13. Staple all printouts together.
14. Hand in at the start of class.

MORE ON BASIC COMPUTATIONS

Here are some additional points about simple computations.

Bad numbers

```
% Inf(inity)
1/0

% N(ot)aN(umber)
0/0

% "underflow" can be dangerous
1.0546e-34^10

% "overflow" is at least as bad
(1/1.0546e-34)^10
```

```
warning: division by zero
ans = Inf
warning: division by zero
ans = NaN
ans = 0
ans = Inf
```

Accuracy

Normally Matlab numbers have a "relative" error of about 10^{-16}. That means that there are about 16 correct digits, starting from the first nonzero digit.

```
% try something
(1/3)+(1/3)+(1/3)-1

% oops, not intended to be that accurate, try again
(1/3)+(1/3)+(1/3)+(1/3)+(1/3)+(1/3)-2

% try it with bigger numbers
(1000/3)+(1000/3)+(1000/3)+(1000/3)+(1000/3)+(1000/3)
    -2000

% print out the "absolute" error in 1
eps(1)
% print out the "absolute" error in 1000
eps(1000)

% watch very large values of the argument of trig
    functions
sin1=sin(10*pi)
sin2=sin(10000000000000000*pi)
```

```
cos1=cos(10*pi)
error=cos1-1
cos2=cos(100000000000000000*pi)
error=cos2-1
```

```
ans = 0
ans =    -2.2204e-16
ans =    -2.2737e-13
ans =     2.2204e-16
ans =     1.1369e-13
sin1 =    -1.2246e-15
sin2 = -0.37521
cos1 =  1
error = 0
cos2 = -0.53004
error = -1.5300
```

Precedence

If no parentheses are used, the following order of precedence applies to basic computations:
highest: ^
lower: *, /
lowest +, -

```
% without parentheses
2+3*4
% since * takes precedence over +, this is the same as
2+(3*4)
% and not the same as
(2+3)*4

% without parentheses
12/2*3
% since / and * have equal precedence, this is
(12/2)*3
% and not
12/(2*3)
```

```
ans =  14
ans =  14
ans =  20
ans =  18
ans =  18
ans =  2
```

Manipulating variables

```
% always keep track of *what* is stored in a variable
x=1
y=2

% let's try to swap the values naively
y=x; x=y;

% Note in the above that the trailing semi-colons prevent
% the new values of x and y to be printed.  We were
% keeping them secret.  But now look at the results;
% we did not correctly swap the values; the 2 got lost.

% lets try again
x=1
y=2

% This time we prevent the value of y from becoming lost
% by storing it in a temporary variable called 'temp'

% save the original value of y
ySaved=y
% now give y the value of x
y=x
% and give x the *saved* value of y
x=ySaved
```

```
x =   1
y =   2
x =   1
y =   2
ySaved =   2
y =   1
x =   2
```

Pi to a trillion digits is not enough?

```
% show pi (also note workspace)
pi

% the Indiana pi bill would redefine pi as 3.2
pi=3.2
% see workspace
pi
```

```
% maybe not a good idea?
clear pi

% we have the old value back
pi
```

```
ans =   3.1416
pi =   3.2000
pi =   3.2000
ans =   3.1416
```

MORE ON FUNCTIONS

Many students are confused by functions. Let's see whether we can figure out exactly what Matlab does when a simple function like `sqr` is used.
Contents of sqr.m:

```
function x2 = sqr(x)

x2 = x*x;

end
```

Contents of trysqr.m:

```
disp('Start of trysqr.m')

sqr(3)

y=4

sqr(y)
```

To run:

1. Observe the workspace.
2. Set a break point before the first use of sqr.
3. Invoke trysqr.m.
4. Observe the workspace.
5. Use Step-into.
6. Observe the workspace. (Matlab uses Pass-by-Value)
7. Use Step

8. Observe the workspace.
9. Etcetera.

ARRAYS

Arrays are tables of numbers. They are usually created using square brackets.

Some examples

```
% create a row of numbers
list=[1 2 4 9 16]

% matlab functions can handle entire lists!
sqrt(list)

% another example
list=[0 30 45 60 90]
sind(list)
cosd(list)
tand(list)

% there are two ways to create columns of numbers
list=[1; 2; 4; 9; 16]
sqrt(list)

% the other way is to put a quote on a row
list=[1 2 4 9 16]'

% another quote turns it back into a row
list'
```

```
list =
    1    2    4    9    16
ans =
   1.0000    1.4142    2.0000    3.0000    4.0000
list =
    0    30    45    60    90
ans =
   0.00000    0.50000    0.70711    0.86603    1.00000
ans =
   1.00000    0.86603    0.70711    0.50000    0.00000
ans =
   0.00000    0.57735    1.00000    1.73205        Inf
list =
```

```
     1
     2
     4
     9
    16
ans =
    1.0000
    1.4142
    2.0000
    3.0000
    4.0000
list =
     1
     2
     4
     9
    16
ans =
     1     2     4     9    16
```

A trick

You can create some types of arrays more easily using [START:END] notation. More generally, you can use [START:STEP:END] notation.

```
% the straightforward way
list=[1 2 3 4 5 6 7 8 9 10]

% the quickest way
list=[1:10]

% the more general way
list=[1:1:10]

% try another
list=[-4:2:12]
```

```
list =
     1     2     3     4     5     6     7     8     9    10
list =
     1     2     3     4     5     6     7     8     9    10
list =
     1     2     3     4     5     6     7     8     9    10
list =
    -4    -2     0     2     4     6     8    10    12
```

Fixing our sqr function

```
WARNING:
    Matlab has a nasty habit of starting to take
  dot products when multiplying arrays of functions.
  To avoid trouble, precede *, /, and ^ with a point.
```

Contents of sqrFixed.m and sqrDot.m:

```
function x2 = sqrFixed(x)

x2 = x.*x;

end
```

```
function x2 = sqrDot(x)

x2 = x'*x;

end
```

```
list =[1:10]
goodSqrt=sqrt(list)
% badSqr=sqr(list)
goodSqr=sqrFixed(list)
goodDot=sqrDot(list')
```

```
 list =
     1     2     3     4     5     6     7     8     9    10
 goodSqrt =
  Columns 1 through 8:
    1.0000    1.4142    1.7321    2.0000    2.2361    2.4495
          2.6458    2.8284
  Columns 9 and 10:
    3.0000    3.1623
 goodSqr =
      1      4      9     16     25     36     49     64     81
           100
 goodDot =  385
```

End lesson 1