S O L U T I O N S

Introduction to University Mathematics 2017

MATLAB Worksheet II

Complete the following tasks and hand in before the end of the lab session.

1. Variables You can create new variables in MATLAB by simply using the equal sign. If you type

   `x=6`

   MATLAB creates a new variable called x, which at the moment has been given value 6 (typically you would then put x through a whole load of mathematical operations). You should see that `x` now appears in the *Workspace* window. The name (and value) of every variable you create will appear in this window.

   You are now free to manipulate x (e.g. try `x^2` or `3*x+5`). No surprises there.

   But try this

   `x=x+1`

   Of course this is a nonsensical *mathematical* statement, but in programming, it *does* make sense, and actually reads more like

   "*new* `x` = *old* `x` +1"

   And you can now see that x is now assigned the value 7.

   In programming, you should always take the sign `=` to mean an *assignment* rather than an equality (I find it useful to think of the statement above as $x \to x + 1$).

   If you understand MATLAB's `=` as the assignment symbol, you can see it's ok to type `x=2`, but not `2=x`, since you can't *assign* 2 any other value (try it and see).

   Here are some exercises with variables. Let's start again with a clean slate by typing `clear`. This wipes out all the variables you created. Careful not to get this confused with `clc`, which just clears the command window.

   (a) Write down the MATLAB command needed in each step:

   i) create x, which can be any integer you like, `x=-1771`

   ii) create y, which is the year you were born multiplied by i,`y=1770i`

   iii) create z, which is the sum of x^2 and y^2, `z=x^2+y^2`

   iv) replace z with the value of $4\sqrt{z}$. `z=4*sqrt(z)`

   Write down the value of $\ln z$ **`log(z)=5.4724`** (4 dec pl.).

   (b) Search the MATLAB documentation (search box on top-right) for an article called "*variable names*". Read the article and experiment. Explore even the "*See also*" suggestions. Then answer the following questions.

i) The maximum number of characters allowed in a variable name is **63**.

ii) Does MATLAB treat `myvar` and `MyVar` as the same variable?

No.

iii) Are `2day` and `_xyz` valid variable names? Explain.

No. Variable names must begin with a letter.

iv) Note that MATLAB allows you create a variable name which clashes with one of MATLAB's stored constants (*e.g.* `i=2`) or function (*e.g.* `sin=3`), although you may suffer some nasty consequences.
If you have accidentally created, say, `sin = 3`, how would you undo this error?

Type `clear sin`. You can also just type `clear` but you'll lose all variables. Beware.

Are the names `cat` and `dog` free from such a clash? How can you tell?

Try `exist cat` to see that it has an 'illegal' level of 5 (meaning it is a MATLAB function). However `exist dog` tells you it is OK.

OR type `cat` to see that MATLAB expects some arguments (i.e. it is a predefined function), but `dog` is simply an undefined variable (so you can go ahead and define it).

OR type `cat` and press TAB. This is called tab-completion, i.e. MATLAB is guessing that you're typing one of its stored words, which include `cat` as you can see in the list that appears.

v) There are some highly reserved "*keywords*" that MATLAB forbids you to use as variable names. List 3 such names.

The full list can be found by typing `iskeyword`. You will be using many of these keywords soon.

2. Types. Every variable has a *type* or *class* associated with it.

   Real numbers with decimal points (also called *floating points* or simply *floats* – keep this word in mind) are probably the most interesting type of numbers for numerical problems. They come in two flavours: *single* or *double* precision, depending on how much memory you are willing to use to store them. By default, MATLAB stores all numbers as *double-precision floats* ("*doubles*" in short).

   i) Computer memory is commonly measured in *bit* or *byte*. Explain what they mean.

   One bit = one binary integer (1 or 0). A byte is 8 bits.

   ii) How much memory is needed to store a variable of type `double`? **8 bytes (or 64 bits)**

   We will explore many other types of variables as we go along.

3. Logical. Another important type of variable in programming is the *logical* type, with only two members: True or False.

   Sometimes 1 and 0 are used to denote true or false, but which is which?

   Ans: **1= True, 0 = False**

   Hint: For example, ask MATLAB if $6 < 7$.

Here are a few MATLAB logical tests that produce a true/false answer, together with their mathematical meaning.

`<`	test for $<$	`<=`	test for $\leq$
`>`	test for $>$	`>=`	test for $\geq$
`==`	test for equality	`~=`	test for $\neq$
`&`	AND	`|`	OR
$\sim$	NOT		

In programming, inequality signs are *logical tests* (`6<7` reads "Is 6 less than 7?"). They are not there to convey facts (mathematically, $6 < 7$ reads "6 *is* less than 7").

Hence, it also makes sense to type `7<6` (try it and see).

Here AND, OR, NOT are the *logical* (or *Boolean*) operators which work exactly as you think they should. For instance,

`true&false = false` `true|false = true` $\sim$`(false) = true`

Try typing the LHS of these statements into MATLAB (also try the 0,1 notation).

(a) Complete the following truth tables for the AND and OR operators.

`&`	1	0
1	**1**	**0**
0	**0**	**0**

`|`	1	0
1	**1**	**1**
0	**1**	**0**

(b) What is the difference between the following commands?

`pi=3` and `pi==3`

`pi=3` assigns a variable 'pi' with value 3, whereas `pi==3` tests whether `pi` equals 3. Incidentally the latter will be true if you had done the command `pi=3` first.

(c) Now create variables `u = 3` and `v = -1` (let's also `clear pi` just in case).
In each of the following cases, explain what the command means, and give the answer you expect (confirming your answers with MATLAB).
The first one has been done for you.

i. `u>=v+4`
Ans: This tests whether $u \geq v + 4$, which is TRUE since $3 \geq 3$.

ii. $\sim$`(u==u)`
"NOT(is $u = u$?)". Clearly `u` equals `u`, so `u==u` gives 1. The tilde says NOT(1), so it gives the opposite answer, which is `0`.

iii. `(u<10)&(pi<1)`
"(is $u < 10$?)AND(is $\pi < 1$?)" This reduces to (1)AND(0). From the truth table, this is `0`.

iv. `(v~=1)|(u==1)`
"(is $v \neq 1$?)OR(is $u = 1$?)" This reduces to (1)OR(0). From the truth table, this is `1`.

(d) (A conundrum) This is an extract from my MATLAB screen:

```
>>x==x
ans =
     0
```

What value did I assign to `x`? (or has my MATLAB gone mad?)

x=NaN.

This unique property of NaN is often used to test whether a code gives sensible answer: just test whether the answer your code produces equals itself. If not, it is a NaN, and something clearly went wrong with your code.

(Before we move on, try typing `whos`)

4. Arrays.

MATLAB stands for **Matrix Laboratory**. Indeed, Vectors and matrices are at the heart of MATLAB, and much of MATLAB's unique capabilities are due to its vector/matrix based computation. Even problems that don't seem to involve vectors are often more *efficiently* solved in MATLAB when they are rephrased in terms of vectors.

In computing, a vector (*i.e.* a list of numbers) is also called an *array*.

(a) Creating arrays. Let's first look at how to create arrays. Try typing

```
A=[1 2 4 9]
```

This creates a horizontal array (row vector) called `A`. Note that spaces are used to separate entries in the array (you can also use commas). The most important thing to note here is the use of **square brackets**, which tells MATLAB that you're creating an array. Here are other quick ways to create arrays. Experiment and fill in the blanks. The first row has been done for you. Try to make sense of the syntax in each case.

Commands	Array created
`[1:5]`	[1 2 3 4 5]
`[3:6]`	**[3 4 5 6]**
`[1:2:9]`	**[1 3 5 7 9]**
`[1:2:10.3]`	**[1 3 5 7 9]**
`[4:-1:0]`	**[4 3 2 1 0]**
`[3:2]`	**empty array**

(b) Use the colon notation to generate the array `myvec=[-1 -1.5 -2 -2.5 -3]`

Ans: `myvec= [-1:-0.5:-3]`

How much memory does it take to store `myvec`? Ans: **40 bytes (5 doubles = 5*8 bytes), try `whos`, which list your variables, their types and sizes.**

(c) Square VS Round. Square brackets are used to create an entire array, whereas round brackets are used to denote specific *elements* of the array. For example, if we create

```
M = [0 sqrt(3) -48 pi]
```

Then `M(1)=0` (its first element), and `M(3) = -48`.

Warning: Many programming errors occur when you mix up these two types of brackets.

Round brackets can also be used to modify specific elements of the array. Explain what each of these commands does to `M`.

a) `M(3) = 0` b) `M(4) = 2*M(4)` c) `M(2) = M(1)`

a) The 3rd element is replaced by 0.
b) The fourth element is doubled.
c) The second element is given the same value as the first element.

(d) Expand and contract. Continuing with the array `M`. Explain carefully what happens when you type each of the following commands

`M(5)=1`

This creates 1 as the fifth element of `M`.

`M(16)=9`

This creates 9 as the 16th element of `M`, whilst the 6th to the 15th element are all 0 by default.

`M(4:8)`

This displays the 4th to the 8th element of `M`.

Note that the last command does not change `M`. At this point your array `M` should contain 16 elements.

What single command can we use to shrink `M` by discarding all the even entries? (i.e. replace `M` by a smaller version of itself, keeping only `M(1)`, `M(3)`, `M(5)` *etc.*)

`M=M(1:2:16)` OR `M=M(1:2:15)` OR `M=M(1:2:end)` .
If you write just `M(1:2:16)`, that doesn't do anything to `M` (it just picks bits of `M` to display). To change `M` you need to reassign it to something else.
Alternatively, you can 'blank out' all the even entries by typing `M(2:2:16)=[]`

(e) Column vectors. There are two ways to create a column vector.

i) Separate array entries using a semicolon ; (rather than spaces). Try creating

```
B=[1;2;3;4;5;6]
```

ii) Use the apostrophe ' to *transpose* a row vector.

Write down the command which will create `B` using method ii)

B=[1 2 3 4 5 6]', or, better still, B=[1:6]'

5. Matrices. We can think matrices as layers of row vectors, each layer separated by ; as in the previous activity. For example, try

```
mat = [3 4 5; 0 9 8]
```

We say that `mat` is a matrix of dimension 2×3. Again note the square brackets. Round brackets can be used to pick out or modify elements as we did with vectors, but now elements are located using 2 numbers, `(row,column)`.

(a) Explain what the command `mat(1,3)` shows you.

It displays $\texttt{mat}_{13}$ (element in first row, 3rd column), which is 5.

(b) Explain what the command `mat(:,2)` shows you.

It displays the 2nd column of `mat`.

(c) How can you add `[0 3 6]` to `mat` as its third row ? (preferably without typing out `mat` all over again.)

`mat(3,:)=[0 3 6]` **or** `mat = [mat; 0 3 6]`

(d) How can you swap the rows and columns of `mat` (keeping the result as `mat`)?

`mat=mat'`

(e) What commands can be used to quickly generate these matrices? (use a single command in each case.)

$$X = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad Y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad Z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

`X=zeros(3), Y=ones(3,5), Z=eye(4).`

Read the documentation under *Array Creation and Concatenation.*

Next week: matrix functions and operations. I will assume you know basic matrix and vector operations.