Review of MATLAB programming

MATLAB matrices [ ]

- MATLAB was originally written for Linear Algebra (i.e. Matrix Algebra).
- MATLAB is short for MATrix LABoratory.
- A matrix is a 2-Dimensional Array which obeys special arithmetic rules.
- Matrices are written like:

M x N

Rows x Columns

3 x 4

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

- This is why in MATLAB, arrays are initialized with square brackets, e.g. :

```
A = [ 1 2 3 4; 5 6 7 8 ; 9 10 11 12];
```

Arrays

- The fundamental unit of data in MATLAB
- Scalar (0D): 1 x 1 array, e.g. s = [5] or s = 5;
- Vector (1D):
  - Row vector: 1 x N array, e.g. x = [1 2 3 4 5] or x = 1:5;
  - Column vector: M x 1 array, e.g. x = [1; 2; 3; 4; 5] or x = [1 2 3 4 5]' or x = [1:5]';
- Matrix (2D): M x N, e.g. A = [1 2 3; 4 5 6];
- N-dimensional arrays (e.g. 3D: M x N x P)

Empty arrays can exist, too. An example:
A = [ ],
size(A) is 0x0

Initialising arrays with built-in functions

- zeros(m, n)
- ones(m, n)
- eye(m, n)
- nan(m, n)
- rand(m, n)
- randn(m, n)

```
>> b = zeros(2, 3);
>> b = ones(2, 3);
>> b = eye(2, 3);
>> b = nan(2, 3);
```

These functions all create arrays. Which types should be self explanatory... `ones(2)==[1 1;1 1]` *and* `ones(1,2) == [1 1]` (`eye` *is the eyedentity matrix—sound it out to yourself.*)

Array length, size, number of elements

```
>> A = [ 1 2 3 4; 5 6 7 8; 9 10 11 12];
>> length(A)
4
>> size(A)
3 x 4
>> numel(A)
12
```

Data types

Basic data types of MATLAB variables

- **Array of numbers: Includes scalars, vectors, matrices…**

  Each number can be a floating point value (or an integer) with a real and imaginary part.

  >> var = 1 + i ;

  Each number could also be a logical (boolean) value: either a 0 or 1 indicating FALSE or TRUE.

  Can have more dimensions to the array, e.g. (3D) M x N x P.

- **string:**

  >> comment = 'This is a character string' ;

Introducing both **structs** and **cells**

- A struct is a special data type whose data is stored in fields that are accessible by name
  - student.name = ‘joe’
  - student.age = 25;

  equivalent to...
  - student = struct(... ‘name’ , ‘joe’ , ’age’ , 25)

student	(1)	(2)	(3)
.name	‘Jack’	‘Jo’	‘Jake’
.age	21	25	30

- A cell is a container that can hold disparate types of data
  - mycell(1) = {[1 5]}
  - mycell(2,1) = {student}

MyCell	{:,1}	(:,2)	{:,3}
{1,:}	[1 5]	‘Ted’	true
{2,:}	21	student	30

curly braces tell MATLAB to wrap this value inside a cell.

Cells vs String Arrays

Character array

- Each character is an element
- Each string must be the same length, but spaces can be used to pad them to the same length.
- Access each string via (row,:)
- Access columns via (:,col)

R	S	O	_
R	E	F	_
R	D	W	B
R	E	D	_
R	D	N	_

Cell array

- Each entire string is an element
- Each string can be any length
- Access string via {whichword}

‘RSO’
‘REF’
‘RDWB’
‘RED’
‘RDN’

simple output to the screen: disp()

The disp(*array*) function

```
>> disp( 'Hello world' )
Hello world
>> disp(5)
     5
>> name = ‘Joe Sixpack'; age = 55;
>> disp( [ 'Hello ‘,  name, ‘ you are ‘,num2str(age),
   ‘years old.’ ] )
Hello Joe Sixpack, you are 55 years old.
```

formatted output to the screen (or to a string variable): sprintf()

sprintf(*format, list_of_variables…*)

```
>> name = 'Joe Sixpack';
>> age = 55;
>> s = sprintf('Hello %s you are %d years old.', name,
   age);
>> disp(s);
Hello Joe Sixpack, you are 55 years old.
```

formatted output

- %d integer
- %f floating point format
- %e exponential format
- %g either floating point or exponential format, whichever is shorter
- \n new line character
- \t tab character

- %5.2f floating point format with 5 characters where 2 are after the decimal point.
- %3d an integer that uses 3 characters max

```
disp(sprintf('%5.2f', pi));          %  3.14
disp(sprintf('%5.3f', pi));          % 3.142
disp(sprintf('%d', pi));             % 3
disp(sprintf('%05.2f', pi));         % 003.14
```

Mathematical Operator Precedence

Hierarchy of operations

- x = 3 * 2 + 6 / 2 – 3^2;
- Processing order of operations is important
  - parentheses (starting from the innermost)
  - exponentials (from left to right, note 3^2 = 3 squared = 9)
  - multiplications and divisions (from left to right)
  - additions and subtractions (from left to right)

```
>> x = 3 * 2 + 6 / 2 – 3^2;
x =
    0
```

Best practice: Clarify with parentheses

```
>> x = (3 * 2) + (6 / 2) – (3^2);
```

Now the meaning is obvious.

Special Values

- pi: π value up to 15 significant digits
- i, j: sqrt(-1)
- Inf: infinity (such as division by 0)
- NaN: Not-a-Number (division of zero by zero)
- clock: current date and time in the form of a 6-element row vector containing the year, month, day, hour, minute, and second
- date: current date as a string such as *16-Feb-2004*
- eps: epsilon is the smallest difference between two numbers
- ans: stores the result of an expression
- now: current date and time as a datenum

Basic Mathematical Functions

- abs, sign
- log, log10, log2
- exp, power
- sqrt
- sin, cos, tan
- asin, acos, atan
- max, min
- round, floor, ceil, fix
- mod, rem
- mean, median, mode, std, sum, cumsum
- nanmean, nanmedian, nanstd, nansum (if your data contains NaN values).

Warning: `NaN` *will screw up some calculations like* "`mean`". *So use* "`nanmean`".

Remember these

- help *command (or* helpwin *command)* → Online help
- doc *command* → Online help
- lookfor *keyword* → Lists related commands
- which → Version and location info
- path → Shows all directories on your path.
- clear → Clears the workspace
- clc → Clears the command window
- diary *filename* → Sends output to file
- diary on/off → Turns diary on/off
- who, whos → Lists content of the workspace
- more on/off → Enables/disables paged output
- Ctrl+C → Aborts operation
- … → Line continuation
- % → Comments
- %% → Sections (in scripts)

Functions must be in one of the directories stored in path for help, lookfor and which to be able to find them, or to be able to call them.

Includes current directory.

For loop

```
for myVar = someArray
      %  put some statements here to do something based on myVar
end
```

Example 1: A simple countdown

```
for count = [10: -1 : 0];
      disp(sprintf('%d…', count));
      pause(1);   % pause for 1 second
end
disp('we have lift off!');
```

10…9…8…7…6…5…4…3…2…1…0…we have lift off!

Example 2: Loop over a cell array of stations

```
station = {'RDWH'; 'RSO'; 'REF'; RDN'};
for stationNum = 1 : length(stations)
      [lat, lon] = loadStationCoordinates( station{stationNum} );    % my own function
      plotStation(lat, lon, station{stationNum});   % my own function
end
```

While loop

```
while (someExpressionIsTrue)
    %  put some statements here to do something
end
```

Example 1: A simple countdown

```
count = 10;
while (count > 0)
    disp(sprintf('%d…', count));
    pause(1);   % pause for 1 second
     count = count – 1; % subtract 1 from count, else loop will never end!
end
disp('we have lift off!');
```

10…9…8…7…6…5…4…3…2…1…0…we have lift off!

While loop

```
while (someExpressionIsTrue)
    %  put some statements here to do something
end
```

Example 2: Do something every ten minutes until a particular time

```
waitTime = 600;  % seconds between iterations
endTime = datenum(2009, 4, 29, 12, 30, 0);
while (now < endTime)
    disp(sprintf('Time now is %s. Remember the lecture at  %s', datestr(now, 31), datestr(endTime,
    31));
    pause(waitTime);   % pause for 10 minutes
end
disp('Too late!');
```

```
Time now is 2009/04/29 12:18:05. Remember the lecture at 2009/04/29 12:30:00.

Time now is 2009/04/29 12:28:05. Remember the lecture at 2009/04/29 12:30:00.

Too late!
```

If…else…end

```
if (someExpressionIsTrue)
        doThis;
elseif (someOtherExpressionIsTrue)
        doThisInstead;
else
        doSomethingElseInstead;
end
```

Example 1:

```
myBirthday = [1971 07 18];
dv = clock; % returns current date/time like [yyyy mm dd HH MM SS];
if (dv(2) == myBirthday(2)  &&  dv(3) == myBirthday(3))
        disp('Happy Birthday!');
end
```

Example 2:

```
name = input('Enter your name', 's');
theAList = {'Bill'; 'George'; 'Barack'};
theBList = {'Al'; 'Dick'; 'Joe'};
if (sum(ismember(theAList, name)>0)
        fprintf('Welcome %s you are on the A List', name);
elseif (sum(ismember(theBList, name)>0)
        fprintf('Welcome %s you are on the B List', name);
else
        fprintf('You''re not on the list, you can''t get in');
end
```

switch...case...otherwise...end

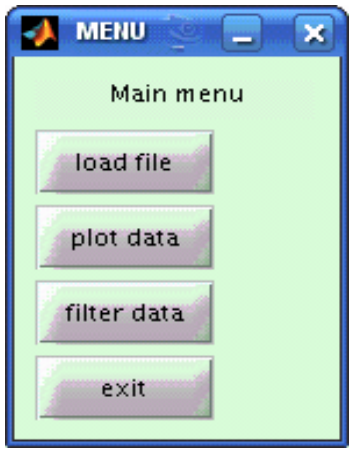
Example: Simple interactive menu

```
choice = 0;
while (choice ~= 4)
    choice = menu('Main menu', 'load data', 'plot data', 'filter data', 'exit');

    switch choice
        case 1, data = loadData();
        case 2, plotData(data);
        case 3, filterData(data, filename);
        otherwise, disp('exiting...');
    end

end

% *** define your loadData, plotData and filterData functions ***
```



Creating functions

```
function outputStuff = function_name(inputStuff)
% FUNCTION_NAME here is the one line summary of the function, used by LOOKFOR
% This is the body of the function where it is explained exactly how to
% call it, what it does to the data, and shows an example of how it should be used.
% All of this shows up when someone types help function_name at the prompt

% Because this line isn't contiguous with the previous comments, it doesn't appear
% on the help.  Instead, it is merely a comment internal to the program

outputStuff = inputStuff;  %this is where the actual operations start
```

- A function only knows about variables that are created within it, so there is no need to worry about pre-existing values.
- The comments immediately below the function declaration are displayed when the user asks for **HELP** for a function
- The MATLAB command **lookfor** searches the first comment line

Vectorizing your code

```
% log of numbers from .01 to 10
x = .01;
for k = 1:1001
  y(k) = log10(x);
  x = x + .01;
end
```

```
% log of numbers from .01 to 10
x = .01:.01:10
y = log10(x);
```

```
% append ".new" to all files in direct
files = dir;
for n = 1:numel (files)
  newfiles(n)=...
    {strcat(files(n).name, '.new')}
end
```

```
% append ".new" to all files in direct
files = dir;
newfiles = strcat({files.name},'.new')
```