INTRODUCTION TO PROGRAMMING IN MATLAB

MATLAB NAMING CONVENTIONS
SCRIPTS AND FUNCTIONS

Names of variables must start with an alphabetic character. MATLAB is a case sensitive language, ie x and X are two different variables. No space is permitted. It is a good practice to use descriptive names and avoid abbreviations. One can use capitalisation or an underscore instead of the space, eg polytropicIndex or polytropic_index. A name should be no longer than 32 characters.

The same naming conventions apply to scripts and functions or the so-called M-files. When saving an m-file ensure that the name conforms to the naming conventions.

There are two kinds of M-files:
- Scripts are text files saved with extension *.m that contain a list of valid MATLAB commands. Scripts do not accept input arguments or return output arguments. They operate on data in the workspace.
- Functions are different to scripts. They can accept input arguments and return output arguments. In functions internal variables are local, i.e. they are not visible outside functions. MATLAB commands are functions.

If you plan to create and save M-files ensure that you have changed the working folder to where you want to save them. As you develop more of your own M-files, you will want to organize them into other folders and personal toolboxes that you can add to your MATLAB search path. If you duplicate function names, MATLAB executes first the one that occurs first in the search path.

In order to just view the contents of an M-file, for example, mean.m, use type, ie type mean.

VARIABLES

The default variable in MATLAB is a numerical rectangular matrix (array). A one-dimensional matrix (one column or row) is called a vector. A one-element matrix is called a scalar. The values are stored as double precision floating point values. Each value occupies eight bytes of memory. The following line shows an example of allocating values in matrix a. Note the use of square brackets.

\[
a = \begin{bmatrix} 1, 2, 3; 4, 5, 6 \end{bmatrix}
\]

\[
a = \\
1 \ 2 \ 3 \\
4 \ 5 \ 6
\]

Elements of matrices are addressed by specifying their row and column combinations enclosed in round parentheses. Use the : operator to address all elements, eg all row elements in the 2nd column:

\[
a(:, 2)
\]

\[
ans = \\
2 \\
5
\]

To print the first element in the second row:

\[
a(2,1)
\]

\[
ans = \\
4
\]

When only a single value is assigned to a variable (scalar), it is still a 1x1 matrix.

\[
b = 5;
\]

\[
b(1,1) = 5 \text{ or}
\]

\[
b
\]

\[
ans = \\
5
\]

\[
b =
\]
To assign a text string to a variable, enclose the string between apostrophes

c = 'VAM2011'

c = VAM2011

To extract the second character:

c(2)

ans =
A

To see the ASCII character codes of the characters in c:

double(c)'

ans =
69
78
67
50
56
49
50

MATLAB has functions that can be used to preallocate matrices and fill them with numbers. Study the MATLAB on-line help for additional explanations and instructions how to use them:

x = zeros(2,3)

x =
0 0 0
0 0 0

x = ones(2,3)

x =
1 1 1
1 1 1

x = eye(3)

x =
1 0 0
0 1 0
0 0 1

Uniformly (rand) or normally (randn) distributed random numbers

x = rand(5,2)

x =
Column 1
0.61543234810009
0.79193703742704
0.92181297074480
0.73820724581067
0.17626614449462

Column 2
0.40570621306210
0.93546969910761
0.91690443991341
0.41027020699095
0.89364953091353
The 'double colon' operator to fill a matrix with numbers (eg from 10 to 15 with the step of 0.5). If the step is omitted, the default step value of 1 is applied.

\[
x = [10 : 0.5 : 15]'
\]

\[
x =
\begin{bmatrix}
10.0000 \\
10.5000 \\
11.0000 \\
11.5000 \\
12.0000 \\
12.5000 \\
13.0000 \\
13.5000 \\
14.0000 \\
14.5000 \\
15.0000
\end{bmatrix}
\]

Also see HELP for \texttt{linspace}

An empty matrix:

\[
x = [ ]
\]

\[
x =
\begin{bmatrix}
\end{bmatrix}
\]

\textbf{Arithmetic Operators} \ + \ - \ * \ \ / \ \ \ ^ \ \ '

MATLAB has two different types of arithmetic operations. Matrix arithmetic operations are defined by the rules of linear algebra. Array arithmetic operations are carried out element-by-element, and can be used with multidimensional arrays.

The period character (\texttt{.}) distinguishes the array operations from the matrix operations. However, since the matrix and array operations are the same for addition and subtraction, the character pairs \texttt{.+} and \texttt{.-} are not used.

\texttt{+} Addition or unary plus.
\ A+B adds A and B. A and B must have the same size, unless one is a scalar. A scalar can be added to a matrix of any size.

\texttt{-} Subtraction or unary minus.
\ A-B subtracts B from A. A and B must have the same size, unless one is a scalar. A scalar can be subtracted from a matrix of any size.

\texttt{*} Matrix multiplication.
\ C = A*B is the linear algebraic product of the matrices A and B. More precisely, For nonscalar A and B, the number of columns of A must equal the number of rows of B. A scalar can multiply a matrix of any size.

\texttt{.*} Array multiplication.
\ A.*B is the element-by-element product of the arrays A and B. A and B must have the same size, unless one of them is a scalar.

\texttt{/} Slash or matrix right division.
\ B/A is roughly the same as B*inv(A). More precisely, B/A = (A\texttt{.'}B\texttt{.'})\texttt{.'}. See \texttt{\}.

\texttt{/} Array right division.
\ A./B is the matrix with elements A(i,j)/B(i,j). A and B must have the same size, unless one of them is a scalar.

\texttt{\} Matrixslash or matrix left division.
\ More on this at later stage.

\texttt{^} Matrix power.
\ X^p is X to the power p, if p is a scalar. If p is an integer, the power is computed by repeated squaring. If the integer is negative, X is inverted first.

\texttt{.^} Array power.
\ A.^B is the matrix with elements A(i,j) to the B(i,j) power. A and B must have the same size, unless one of them is a scalar.

\texttt{'} Matrix transpose.
A' is the linear algebraic transpose of A. For complex matrices, this is the complex conjugate transpose.

.' Array transpose.
    A.' is the array transpose of A. For complex matrices, this does not involve conjugation.

**Relational Operators** <, >, <=, >=, ==, ~=

Relational operations syntax:
A < B
A > B
A <= B
A >= B
A == B
A ~= B  (not equal)
The relational operations return 1 if true or 0 if false.

```matlab
a=5; b=5;
a == b
ans =
    1    %true

a > b
ans =
    0    %false

a ~= b
ans =
    0    %false
```

**Logical operators**
Element-wise &, | and ~

The examples shown in the following table use vector inputs A and B, where
A = [0 1 1 0 1];
B = [1 1 0 0 1];

& (and) returns 1 for every element location that is true (nonzero) in both arrays, and 0 for all other elements.

A & B = 01001

| (or) returns 1 for every element location that is true (nonzero) in either one or the other, or both, arrays and 0 for all other elements.
A | B = 11101

~ (not) complements each element of input array,
~A = 10010

xor returns 1 for every element location that is true (nonzero) in only one array, and 0 for all other elements.
xor(A,B)=10100

**Short-curcuit && and ||**
So called short-curcuit operators operate on scalar values.

&& (and) returns true (1) if both inputs evaluate to true, and false (0) if they do not.

|| (or) returns true (1) if either input, or both, evaluate to true, and false (0) if they do not.

Syntax
A && B
A || B
FLOW CONTROL AND BASIC PROGRAMMING

Keywords
The objective of this section is to practice writing programming scripts that use mainly keywords of Matlab and refrain from using the functions provided in MATLAB toolboxes as much as possible. In the following exercises MATLAB built-in functions should be used only to generate data for manipulations.

MATLAB has only 17 keywords that can be found in the on-line help or listed by executing:

```
iskeyword
```

```
ans =
    'break'
    'case'
    'catch'
    'continue'
    'else'
    'elseif'
    'end'
    'for'
    'function'
    'global'
    'if'
    'otherwise'
    'persistent'
    'return'
    'switch'
    'try'
    'while'
```

All other hundred and hundred of "commands" that are used in MATLAB (such as sum, cumsum, mean, prod, diff, trapz etc.) are in fact functions that have been written and provided in Matlab toolboxes for convenience.

USE OF for LOOP

Exercise 1 - The Factorial

Assign an integer to a variable, say N. Write a script that calculates the factorial N! of N. The factorial of N is the product of all the integers from 1 to N, and by definition 0! = 1. Call the script fact.m to prevent a conflict with built-in functions.

Note: Factorial of 0 is by definition 1. Since double precision numbers only have about 15 digits, the answer is only accurate for N <= 21. For larger N, the answer will have the right magnitude, and is accurate for the first 15 digits.

```
%Lines starting with % are interpreted as comments
%fact.m
%my factorial script
%0 <= N <=21
N = input('Enter an integer to calculate the factorial of: ');
fact = 1;

%the loop will decrement the counter k from N down to 2
%no point going down to one-> multiplication by 1 doesn't change anything
%if N < 2 the loop won't be executed. That's why 1 was assigned to fact at the start
for k = N:-1:2
    %k    %uncomment (remove % in this line to see how the loop works
    fact = fact .* k;    %remove ; to see how the loop works
end
fact    %this will display the answer
```
**Exercise 2**

A magic matrix is a square matrix (NxN) filled with integer elements from 1 to N² with equal row, column and diagonal sum of its elements. MATLAB has a function called magic that returns a magic matrix of the specified size

Eg

```matlab
m = magic(3)
m =
8 1 6
3 5 7
4 9 2
```

As can be seen, the sums of rows, columns and diagonal elements are 15.

Using other MATLAB functions such as `sum` and `diag`, this feature of the magic matrix can be verified by executing (see help on `sum` function by executing `help sum`)

```matlab
sum(m,1)  %row-wise, ie for every column
ans =
15 15 15

sum(m,2)  %column-wise, ie for every row
ans =
15
15
15

sum(diag(m))  %along the diagonal
ans =
15
```

Now, using only the keywords (except `magic`) and various variables, write a script (call it, say `magicCheck`) that verifies that the sum of elements of a magic matrix along rows, columns and diagonal is the same.

The start of the script to calculate the sum for each column has been done for you. Expand the script to calculate the sum in each row and diagonal.

```matlab
%This script magicCheck.m checks if a magic matrix is really magic

magicSize = 4;  %decide the size of magic matrix.

%You can use input to make it easier by uncommenting the next line
%magicSize = input('Enter the size of the magic matrix:  ');

m = magic(magicSize)  %generate the magic matrix. No semicolon to print it

%SUM IN EACH COLUMN (equivalent of sum(m) or sum(m,1)

colSum = [ ];  %allocate an empty variable to grow as the calculations progress

%start the outer 'for' loop to step through every column

for col = 1:magicSize
    acc = 0;  %allocate and set to 0 a storage variable(accumulator) to keep the sum
    %start the inner for loop to sum the elements in the given column,
    %progressing through the rows
    for row = 1:magicSize
        acc = acc + m(row,col);
    end
    colSum = [colSum, acc];
end
```

colSum = [colSum, acc]; % (,) will concatenate along the row
end

%The output was suppressed by putting a semicolon at the end of the lines.
%You may remove some of them to see how the loops execute and the colSum expands
%Now print the results
colSum

m =
16  2  3 13
 5 11 10  8
 9  7  6 12
 4 14 15  1
colSum =
 34  34  34  34

LOOP CONTROL. LOGICAL AND RELATIONAL OPERATORS

Exercise 3

Preventing the input to be outside envisaged limits is important for proper functioning of programs, eg preventing the input of temperature below 0K.

Using the keywords while, break or continue, write a loop that persistently requests the entry if data is outside the limits. The while loop is aborted only when the input of variable x from the keyboard is within specified limits, say xmin and xmax.

Two approaches are demonstrated:

The endless while 1 loop is aborted with the keyword break when an entry is within the limits
xmin = 0;
xmax = 10;

while 1  %1 ia always true
    x = input('Enter the value:    ');
    if isempty(x)  %when just <ENTER> is pressed x=[ ] ie empty
        continue  %back to the start of the loop
    end
    if x >= xmin && x <= xmax
        break  %correct entry->abort the loop
    end
end
x

The while flag loop is aborted depending on the result of the logical relational operation which returns flag =1 (true) when an entry is outside the limits
xmin = 0;
xmax = 10;

flag = 1; %initially the flag is true to enter the while loop
while flag
    x = input('Enter the value:    ');
    if isempty(x)  %when just <ENTER> is pressed x=[ ] ie empty
        continue  %back to the start of the loop
    end
    flag = x < xmin || x > xmax;  %logical result- true if outside the limits
end
x
Now:

1. Incorporate a similar code in the factorial script FACT.M to prevent the input of non-integers and integers outside $0 \leq N \leq 21$
2. Knowing that \texttt{round(X)} rounds the elements of X to the nearest integer, use it to prevent the input of non-integers.

**Exercise 4**

Plot six sinusoids $\sin(i \cdot x)$ for $i=1$ to $6$ and $x=0$ to $360^\circ$ ($2\pi$ radians). Make the six graphs as subplots that are arranged in 2 rows by 3 columns.

```matlab
%make six subplots, 2 rows by 3 columns
%and plot sin(i*x) for i= from 1 to 6 and x from 0 to 2pi
%generate the values of x from 0 to 2pi with step of, say 0.05 and transpose to a
%column vector
x = [0:0.05:2*pi]';
rows = 2;
cols = 3;
for i = 1: rows * cols
    subplot(rows, cols, i)
    plot(x/pi*180, sin(i * x))  %plot x in degrees
    grid on
    xlabel('Angle ^o')
end
```

![Plot of six sinusoids](image)

Now change the value of \texttt{rows} to, say, 3 and run it again.

**Exercise 5**

Using only the keywords (except \texttt{input} and \texttt{size} – see the on-line HELP) write scripts that are able to return similar results as the following built-in MATLAB functions. Test your programs on various matrices. See the on-line HELP for more information on what these functions do. Ignore N-D (multidimensional) arrays.

1. \texttt{sum}
2. \texttt{cumsum}
3. \texttt{mean}
4. \texttt{prod}
5. \texttt{diff}
For example

help sum

SUM Sum of elements.
   For vectors, SUM(X) is the sum of the elements of X. For
   matrices, SUM(X) is a row vector with the sum over each
column. For N-D arrays, SUM(X) operates along the first
non-singleton dimension.

SUM(X,DIM) sums along the dimension DIM.

Example: If X = [0 1 2
         3 4 5]

   then sum(X,1) is [3 5 7] and sum(X,2) is [3
12];

See also PROD, CUMSUM, DIFF.

help cumsum

CUMSUM Cumulative sum of elements.
   For vectors, CUMSUM(X) is a vector containing the cumulative sum of
the elements of X. For matrices, CUMSUM(X) is a matrix the same size
as X containing the cumulative sums over each column. For N-D
arrays, CUMSUM(X) operates along the first non-singleton dimension.

CUMSUM(X,DIM) works along the dimension DIM.

Example: If X = [0 1 2
         3 4 5]

   then cumsum(X,1) is [0 1 2 and cumsum(X,2) is [0 1 3
3 5 7] 3 7 12]

See also CUMPROD, SUM, PROD.

help mean

MEAN Average or mean value.
   For vectors, MEAN(X) is the mean value of the elements in X. For
matrices, MEAN(X) is a row vector containing the mean value of
each column. For N-D arrays, MEAN(X) is the mean value of the
elements along the first non-singleton dimension of X.

MEAN(X,DIM) takes the mean along the dimension DIM of X.

Example: If X = [0 1 2
         3 4 5]

   then mean(X,1) is [1.5 2.5 3.5] and mean(X,2) is [1
4]

See also MEDIAN, STD, MIN, MAX, COV.