CT248 - Introduction to Modelling

Introduction to Modelling

Andrew Hayes

Student ID: 21321503

1BCT1

NUI Galway

February 2022

Contents

1 Lecture 01 - MATLAB Overview

MATLAB (MAtrix LABoratory) is a powerful technical computing system for handling scientific & engineering calculations, designed to make **matrix computations** particularly easy. It is a **high-level language** with an interactive development environment. MATLAB is **loosely typed**.

MATLAB is used for:

- Numerical computation.
- Data analysis & visualisation.
- Algorithm development & programming.
- Application development & deployment.

1.1 Variables

A variable name in MATLAB may consist only of letters a-z, digits 0-9, and the underscore (_) and they must start with a letter. A variable name may not exceed 63 characters. MATLAB is case-sensitive, including command & function names.

The use of a semi-colon ; at the end of a variable initialisation prevents the value from being displayed.

```
carSpeed = 20;
```

The MATLAB Workspace All variables created during a session remain in the workspace until they are **cleared**.

```
clear;
mph = input("Please enter the speed in mph: ");
kph = mph * 1.6;
fprintf("%d mph = %d kph\n",mph,kph);
```

The command `who` lists all the names in the current workspace. The command `whos` lists the size of each variable.

Comments can be made in MATLAB using the `%` sign.

1.2 Expressions

An **expression** is a formula consisting of variables, numbers, operators, & function names. Expressions are evaluated when you enter them into the MATLAB prompt. Note that MATLAB uses the function `ans` to return the last expression that was evaluated but not assigned to a variable.

1.2.1 Statements

MATLAB **statements** are frequently of the form `variable = expression`.
E.g.: `s=u*t-g/2*t.^2`.
This is an example of an **assignment statement**, with the value of the expression assigned to the variable on the left-hand side. A semicolon is at the end to suppress the output.

Statements that are too long for a line can be extended to several lines with an ellipsis of at least 3 dots `...`
Statements on the same line can be separated by commas or semicolons.

1.3 Input & Output

The `input()` statement is used to read in input data.

```
name = input ("Enter your name: ");
```

The `disp()` statement is used to **display** data. The general form of the `disp` statement is `disp(variable)`.

```
>> disp("Hello World!")
>> x = 2;
>> disp(["The answer is ", num2str(x)])
The answer is 2.
>> disp([2 3 x])
2.00 3.00 2.00
```

To display more than one variable, embed the variables into an **array**. All components of a MATLAB array must be the same type.

```
% Script file for converting temperatures from F to C.
% Step 1: Get the input
F = input ("Enter the temperature in degrees F: ");

% Step 2: Convert to C
 C = (F -32) * 5/9;

% Step 3: Display the result
fprintf("The temperature of %f (F) is %f (C)\n",F,C);
```

1.4 Arithmetic Operators

The evaluation of an expression is achieved by means of arithmetic operators. The arithmetic operations on two scalar constants / variables is shown. Left division may seem curious, however *matrix* left division has an entirely different meaning.

Operation	**Algebraic Form**	**MATLAB**
Addition	$a + b$	`a + b`
Subtraction	$a - b$	`a - b`
Multiplication	$a \times b$	`a*b`
Right Division	a/b	`a/b`
Left Division	$b\backslash a$	`b\a`
Exponent	a^b	`a^b`

MATLAB uses PEMDAS as to determine **precedence**.

1.5 Repetition & Loops - The For Loop

The `for` statement repeats some statement a specific number of times. The two basic `for` loop structures are as follows:

```
for index = j:k
    statements;
end

for index = j:m:k
    statements;
end
```

`j:k` is a *vector* with elements $j, j+1, j+2, \cdots, k$. `j:m:k` is a vector with elements $j, j+m, j+2m, \cdots$ such that the final element does not exceed k. `index` must be a variable. Loop statements should be indented.

```
for i = 1:5
    disp(i);
end
```

This for loop repeats the `disp()` statement 5 times, starting with `i = 1` and ending with `i = 5`, incrementing by 1 each iteration of the loop.

For a for loop of the form `for i = a:b`, the number of iterations will be $b - a + 1$.

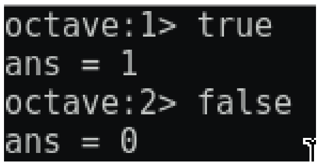
The following example program takes in n numbers and displays their average.

```
clear;
sum = 0;

n = input("How many numbers: ");

for i=1:n
    num = input("Please enter a number: ");
    sum += num;
end

avg = sum/n;

fprintf("The average is %f\n.", avg);
```

1.6 Relational Operators

Relational Operators form a logical expression that is either *true* or *false*. Relational operators form the basis for decision logic.

In MATLAB, "true" & "false" can be represented both lexically (`true`, `false`) and logically (`1`, `0`). When returning the value of a Boolean expression or variable, MATLAB will use logical true (`1`) & false (`0`).



```
octave:1> true
ans = 1
octave:2> false
ans = 0
```

Relational Operator	**Meaning**
<	Less than
<=	Less than or equal to
==	Equal to
~=	Not equal to
>	Greater than
>=	Greater than or equal to

1.6.1 The "If" Statement

In MATLAB, `if` statements take the general form(s):

```
if condition; statements; end

if condition
    statements
else
    statements
end

if condition
    statements
```

```
elseif condition
    statements
else
    statements
end
```

```
num = input("Enter a number: ");
if num < 0; disp("Negative number"); end
```

2 Matrices

In linear algebra, a **matrix** is a rectangular grid of numbers arranged into *rows* & *columns*. An $r \times c$ matrix has r rows & c columns. Uppercase letters (e.g., A) are usually used to denote variables that are matrices. a_{ij} denotes the element in A at row i & column j.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

2.1 Multiplying a Matrix by a Scalar

A matrix A can be multiplied by a scalar k, resulting in a matric of the same dimensions as A. The multiplication takes place in a straightforward fashion, with each element in the new matrix being the product of k times the corresponding element in A.

$$kA = k \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} ka_{11} & ka_{12} & ka_{13} \\ ka_{21} & ka_{22} & ka_{23} \\ ka_{31} & ka_{32} & ka_{33} \end{pmatrix}$$

Let matrix C be the $r \times c$ product AB of the $r \times n$ matrix A with the $n \times c$ matrix B. Each element of C, C_{ij}, is equal to the vector dot product of row i of A with column j of B.

$$AB = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$AB = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

In MATLAB, a matrix is a rectangular object consisting of rows & columns. Matrices have comprehensive support with functions & operators. Rows can be separated when initialising a matrix using the semicolon `;`.

```
>>a = [1 2 3; 4 5 6]
a =
    1   2   3
    4   5   6

>>a'
ans=
        1   4
        2   5
        3   6
```

Individual elements are usually referenced with two subscripts in the form `matrixname(row, column)`. However, elements can also be referenced using *one* subscript. In this case, MATLAB will treat the entire vector as a single column.

```
A=[1 2 3;4 5 6;7 8 9]
A(1,1)
A(1)
A(9)
```

Using the colon `:` operator in place of a subscript denotes all the elements in the corresponding row or column. `A(3,:)` means "all the elements in the third row". `A(:,3)` means "all the elements in the third column". You can extract subsets of a matrix using something like `A(1:2, 2:3)`.

2.1.1 Matrix Functions

Function	**Description**
`eye()`	Identity Matrix
`linspace()`	Vector with linearly spaced elements
`ones()`	Matrix of 1s
`rand()`	Uniformly distributed random numbers & arrays
`randn()`	Normally distributed random numbers & arrays
`zeros()`	Matrix of 0s.
`det()`	Determinant
`eig()`	Eigenvalues & eigenvectors.
`expm()`	Matrix exponential
`inv()`	Matrix inverse
`trace()`	Sum of diagonal elements
`{}\` and `/`	Linear equation solutions.

2.2 Vectors

A **vector** is a special type of matrix, having only one row or one column. MATLAB handles matrices & vectors in the same way.

Rules:

- Elements in the list must be enclose in square brackets.
- Elements in the list must be separated by either spaces or by commas.

```
x = [1 2 3 4 5 6]
y = [1,2,3,4,5,6]
```

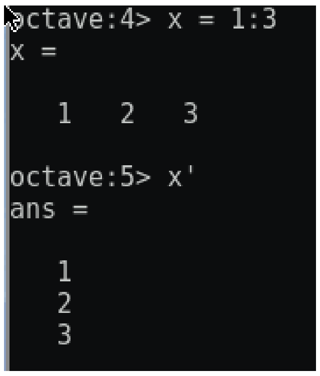
Vectors can also be initialised using the colon `:` operator.

```
x = 1:10
y = 1:5:10
```

Operations can be performed on vectors, e.g `sum(x)`, `mean(x)`, etc. Elements in a vector can be accessed via `x(1)` or `x(1:3)` to get a range of elements. Note: vectors in MATLAB are indexed from 1, not 0.

2.2.1 Transposing Vectors

The vectors generated thus far have been *row vectors*. *Column vectors* may be needed for matrix operations. The single quote `'` can be used to transpose a vector.



```
octave:4> x = 1:3
x =

   1   2   3

octave:5> x'
ans =

   1
   2
   3
```

3 Lecture 02 - Logical Vectors & Arrays

3.1 Element-Wise Operations

4 MATLAB Functions

MATLAB allows you to create your own function **M-files**. A function file differs from a script file in that the function M-file communicates with the MATLAB workspace through specially designated input & output arguments.

A function M-file `name.m` has the following general form:

```
function [outarg1, outarg2, ...] = name(inarg1, inarg2, ...)
% help text
outarg1 = ...;
outarg2 = ...;
end
```

- **Function Keyword -** The function file must start with the keyword `function`.
- **Input & Output Arguments -** The input & output arguments define the function's means of communication with the workspace.
  - If there is more than one output argument, the output arguments must be separated by commas & enclosed in square brackets.
  - Input & output variables can be vectors.
- **Help Text -** When you type `help function_name`, MATLAB displays the comment lines that appear between the function definition & the first non-comment line.
- Function names should follow the MATLAB rules for variable names.

If a function changes the value of any of its input arguments, the change is **not** reflected in the actual argument. An input argument is only passed by value if a function modifies it. If a function does not modify an input argument, it is passed by reference.

A variable number of arguments can be passed to a function. A function may be called with all, some, or none of its arguments. The same applies to output arguments. `nargin` displays the number of arguments passed.

4.1 Scope

Any **local variables** defined inside a function are not accessible outside the function. These local variables exist only inside the function, which has its own workspace, separate from the base workspace.

Variables which are defined in the base workspace and are not normally accessible inside functions. If functions (& possibly the base workspace) declare variables as **global**. then they will all share a single copy of those variables.

Persistent variables, unlike local variables, remain persistent between function calls. A persistent variable is initially an empty array. Persistent variables can be declared with the keyword `persistent <variablename>`.

4.2 Subfunctions

The function M-file may contain code for more than one function. The first function in the M-file is called the **primary function**. Additional functions are known as **subfunctions**, and are visible only to the primary functions & to the other subfunctions. Subfunctions follow each other ina ny order *after* the primary function.

5 Processing Images in MATLAB

5.1 Examples of Image Types

- **Binary Images:** 2D arrays that assign one numerical value from the set (0,1) to each pixel in the image. Often called **logical images**. `0` corresponds to black, while `1` corresponds to white. Binary images can be represented as a simple bit stream.
- **Intensity/Grayscale Images:** 2D arrays that assign one numerical value to each pixel, representing the intensity at said pixel. The range is bounded by the bit resolution of the image.
- **RGB Images:** 3D arrays that assign three numerical values to each pixel, with each value corresponding to the red, green, & blue image channel. Pixels are accessed by `I(Row,Column,Channel)` in MATLAB.

5.2 Key MATLAB Functions

MATLAB's image processing toolbox contains an extensive image processing library, including:

- `imread(filename)` - Reads an image from a graphical file and converts it to a MATLAB array object.
- `imshow(object)` - Displays an image.
- `linspace(X1,X2,N)` - Generates a row vector of `N` linearly spaced points between `X1` & `X2`. If `N` is excluded, 100 points will be generated.
- `subplot(m,n,p)` - Creates axes in tiled positions by dividing the current figure into an `m`-by-`n` grid and creating axes in the position specified by `p`. MATLAB numbers subplot positions by row. The first subplot is the first column of the first row, the second subplot is the second column of the second row, etc.

5.3 Image Colour

An image contains one or more colour channels that define the intensity or colour at a particular pixel location `I(m.n)`. In the simplest case, each pixel location contains only a single numerical value representing the signal level at that point in the image. The most common colour map is **greyscale**. The maximum numerical value representing the signal level is $2^8 - 2 = 255$.

5.4 Operations on Pixels

The most basic type of image processing is a **point transform** which maps the value at individual points (pixels) in the input image to corresponding points in an output image. In a mathematical sense, it's a one-to=one functional mapping from input to output.

Types of operations include:

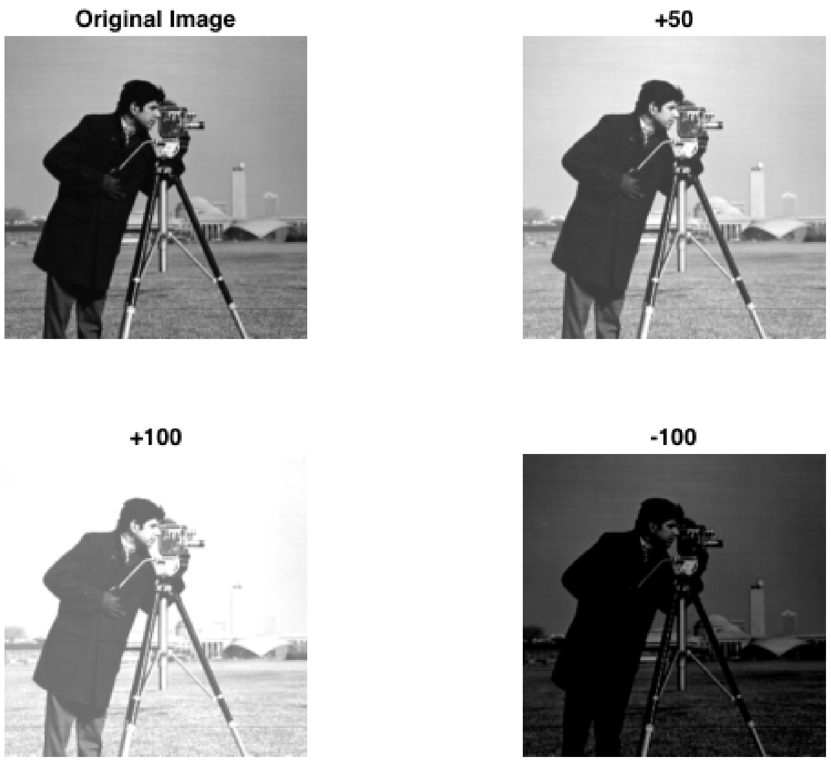
- Pixel addition & subtraction.

- Thresholding.
- RGB to Greyscale.
- Rotation (90º).
- Simple cropping.

5.4.1 Arithmetic Operations on Images

Arithmetic operations can be performed quickly and easily on images. The example below shows contrast adjustment.

```
I = imread("cameraman.tif");

O = I + 50;
O1 = I + 100;
O2 = I - 100;

subplot(2,2,1),imshow(I),title("Original Image");
subplot(2,2,2),imshow(O),title("+50");
subplot(2,2,3),imshow(O1),title("+100");
subplot(2,2,4),imshow(O2),title("-100");
```



5.4.2 Thresholding

Thresholding produces a binary image from a greyscale or colour image by setting pixels to 1 or 0 depending on whether they are above or below the threshold value. This is useful to help separate the image foreground from the background. Logical operators are very useful for this.

```
I = imread('rice.png');
T1 = I > 100;
T2 = I > 105;
```

```
T3 = I > 110;
T4 = I > 115;
T5 = I > 120;
subplot(2,3,1),imshow(I),title('Original Image');
subplot(2,3,2),imshow(T1),title('Threshold @100');
subplot(2,3,3),imshow(T2),title('Threshold @105');
subplot(2,3,4),imshow(T3),title('Threshold @110');
subplot(2,3,5),imshow(T4),title('Threshold @115');
subplot(2,3,6),imshow(T5),title('Threshold @120');
```

6 Data Science

6.1 Introducing the Table in MATLAB

Tables are used to collect heterogeneous data & metadata in a single container. Tables are suitable for storing column-oriented or tabular data that is often stored as columns in a text file or in a spreadsheet. Tables can accomodate variables of different types, sizes, units, etc. Tables are often used to store experimental data, with rows representing the different observations and columns representing different measured variables.

Tables can be subscripted using parentheses much like ordinary numeric arrays, but in addition to numeric and logical indices, one can use a table's variable and row names (if defined) as **indices**. One can access individual variables in a table much like fields in a structure, using **dot subscripting**. One can access the contents of one or more variables using **brace subscripting**.