SM-2302: Software for Mathematicians

Lecture 1: Introduction to MATLAB

Dr Huda Ramli
Mathematical Sciences, FOS, UBD
huda.ramli@ubd.edu.bn

Semester I, 2025/26

Adapted from 6.057 IAP 2019 (MIT OCW)

What is MATLAB?

- MATLAB stands for **MAT**rix **LAB**oratory
- It can be thought of as a super-powerful graphing calculator
- A high-level language for numerical computation, visualization, and programming
- Widely used in engineering, science, and mathematics
- Commands are executed line-by-line

Getting Started

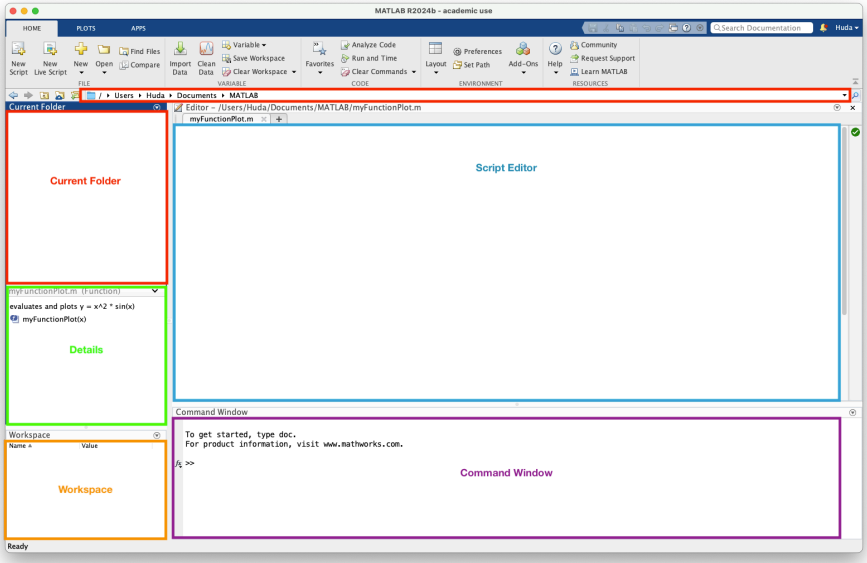
MATLAB is installed in the campus computer labs. However, if you wish to work from home or on your laptop, you can either

1. Use MATLAB Online on your web browser; or
2. Install the MATLAB software on your personal computer.

Mathworks & Campus-wide License (CWL)

- To install or use MATLAB on your web browser, you need to create a Mathworks account using your UBD e-mail.
- You can access the UBD campus-wide suite using your mathworks account.
- Please refer to the MATLAB individual CWL installation guide document.

MATLAB Environment



help & doc

The `help` command is the most important command for learning MATLAB on your own!

- To get info on how to use the sine function: `>> help sin`
- To get a nicer and easy-to-read version of help: `>> doc sin`
- To search for a function by specifying keywords:
  `>> docsearch sin trigonometric`

Scripts: Overview

Scripts are a collection of commands executed in sequence.

- Written in the MATLAB editor
- Saved as m-files (`.m` extension)

To create an m-file script:

- Type `>> edit MyFileName.m` in the command window,
- or click the "New Script" button on the top left

Some notes:

- All variables created or modified in a script retain their values after script execution.
- Add **comments** to your MATLAB scripts
  - Anything following a % sign is interpreted as a comment
  - Comment thoroughly to avoid wasting time later
  - Mark beginning of a code black using %%

Exercise: Scripts

Example 1

1. Make a script with the name `helloWorld.m`
2. When run, the script should show the following text:

```
Hello World!
I am going to learn MATLAB!
```

Hint: Use `disp(...)` to display strings. Strings are written between single quotes, for e.g. `'This is a string'`

Variable types

- MATLAB is a 'weakly typed' language $\Rightarrow$ No need to declare variables.
- MATLAB support various types, some well-known ones are:

  64-bit double (default): `3.84`
  80-bit char: `'hello'`

- Most variables you will deal with are **vectors, matrices, doubles** or **chars**.
- Other types are also supported: complex, symbolic, 16-bit and 8-bit integers (uint16 & uint8), etc.

Naming variables

To create a variable, simply assign a value to a name:

```
myNumberVariable = 3.14
myStringVariable = 'hello world!'
```

Variable name rules:

- First character must be a **letter**
- After that, any combination of numbers, letters and _
- Names are **case-sensitive** (e.g. `var1` is different to `Var1`)

Built-in variables

Build-in variables cannot be used for anything else!

`i, j`	complex numbers
`pi`	has value 3.1415...
`ans`	stores the result of the last unassigned value
`Inf, -Inf`	infinities
`NaN`	Not a Number

Scalars

- A variables can be given a value explicitly: `>> a = 10`
  which shows up in the workspace.
- Or a function of explicit values or existing variables: `>> b = 1.3 * 45 - 2 * a`
- To suppress the output, end the line with a **semicolon**: `>> c = 19/9;`

Arrays

Arrays are an important feature of MATLAB. There are 2 types of arrays:

- Matrix of numbers (double or complex)
- Cell array of objects (more advanced data structure)

	Row vector	**Column vector**
Command	`>> row = [1 2 5.4 -6.6]`	`>> col = [4; 2; 7]`
Output	$\begin{bmatrix} 1 & 2 & 5.4 & -6.6 \end{bmatrix}$	$\begin{bmatrix} 4 \\ 2 \\ 7 \end{bmatrix}$
Size	1×4	3×1

Matrices

Create **matrices** like vectors:

Command	Output
`>> A = [1 2; 3 4];`	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

Concatenate vectors or matrices (dimension matters):

```
a = [1, 2];                          % 1x2 row vector
b = [3 4];                           % 1x2 row vector
c = [5; 6];                          % 2x1 column vector
d = [a; b];                          % 2x2 matrix
e = [d c];                           % 2x3 matrix
f = [[e e]; [a b a]];                % 3x6 matrix
str = ['Hello' 'I am' 'John'];       % strings are character vectors
```

Save, clear & load

- Use `save` to save variables to a file:
  - `>> save myFile a b` saves variables `a` and `b` to the file `myFile.mat` in the current directory
  - Make sure you are in the correct working directory.
- Use `clear` to remove variables from the workspace:
  `>> clear a b` clears variables `a` and `b` from the workspace.
- Use `load` to load variables into the workspace:
  `>> load myFile` puts variables `a` and `b` back in the workspace.

Exercise: Variables

Example 2

1. Create a variable `start` using the function `clock`
2. What is the size of `start`? Is it a row or column?
3. What does `start` contain? See `help clock`
4. Convert the vector `start` to a string. Use the function `datestr` and name the new variable `startString`
5. Save `start` and `startString` into a mat file named `startTime`

Exercise: Variables II

Example 3

1. In `helloWorld.m`, read in variables you saved using `load`
2. Display the following:

```
I started learning MATLAB on [date, time]
```

Hint: Use the `disp` command again. Remember that strings are just vector of characters, so you can join two strings by making a row vector with the two strings as sub-vectors.

Basic operations

- Arithmetic operations using `(+, -,*, /)`:

```
>> 7/45
>> (1+1i)*(1+2i)
>> 1/0
>> 0/0
```

- Exponentiation:

```
>> 4^3
>> (3+4*1j)^2
```

- Complicated expressions (use parentheses):

```
>> ((2+3)*3)^0.1
```

Built-in functions

- MATLAB has an extensive library of built-in functions.
- Call a function using parentheses and passing parameters.

```
>> sqrt(2)      | >> log(2)       | >> log10(0.23)
>> cos(1.2)     | >> atan(-0.8)   | >> exp(2+4*1i))
>> round(1.4)   | >> floor(3.3)   | >> ceil(4.23)
>> angle(1i);   | >> abs(1+1i)    | >> sin(pi/4)
```

Example 4 (Scalars)

Goal: Complete the `helloWorld` script using scalar operations in MATLAB.

1. Let your learning time constant be 1.5 days, and convert this to seconds and assign it to the variable `tau`.
2. The course duration is 8 days. Convert this to seconds and assign it to the variable `endOfClass`.
3. Use the following model for your knowledge over time: $k(t) = 1 - e^{-t/\tau}$
   - Compute your knowledge at `endOfClass` and store it in `knowledgeAtEnd`.
   - Use the `exp` function.
4. Display the sentence:

```
At the end of SM-2302, I will know X% of MATLAB
```

   - Replace `X` with the percentage value of `knowledgeAtEnd`.
   - Use `num2str` to convert numbers to strings.

Hint: There are 86,400 seconds in a day.

Transpose

- **Transpose** operator turns a column vector into a row vector, and vice-versa:
  ```
  >> a = [1 2 3 4+i]
  >> transpose(a)
  ```
- **Hermitian-transpose**: `>> a'` transposes and conjugates all complex numbers.
- For vectors of real numbers, `a.'` and `a'` give the same results.

Addition & Subtraction

- Addition and subtraction are **element-wise**.
- Sizes must match (unless one is a scalar).
- Use the transpose to make sizes compatible.

For example, input the following matrices in your workspace:

$$\mathbf{A} = \begin{pmatrix} -3 & -1 \\ 3 & 2 \\ 3 & 0 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 7 & 3 \\ -2 & 2 \\ 3 & -1 \end{pmatrix}, \quad \mathbf{C} = \begin{pmatrix} -3 & -2 & -2 \\ 5 & 8 & -2 \end{pmatrix}$$

Command	`>> A+B`	`>> B-A`	`>> A+C.'`
Output	$\begin{bmatrix} 4 & 2 \\ 1 & 4 \\ 6 & -1 \end{bmatrix}$	$\begin{bmatrix} 10 & 4 \\ -5 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -6 & 4 \\ 1 & 10 \\ 1 & -2 \end{bmatrix}$

Element-wise functions

- All functions that work on scalar also work on vectors:

```
>> t=[1 3 5];
>> f=exp(t);
```

is the same as

```
>> f = [exp(1) exp(3) exp(5)];
```

- To do element-wise operations use the dot (i.e. `.*`, `./`, `.^`)
- Dimensions must match (unless one is scalar).

For example, input the following vectors in your workspace:

$$\mathbf{a} = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 4 \\ 2 \\ 1 \end{pmatrix}$$

Errors	Valid
`>> a.*b`	`>> a.*b.'`
`>> a./b`	`>> a./b.'`
`>> a.^b`	`>> a.^(b.')`

Operations

- Multiplication can be done standard or element-wise.
- Standard multiplication (`*`) is the matrix multiplication $\Rightarrow$ inner dimensions must match.
- Standard exponentiation (`^`) can only be done on square matrices or scalars.
- Left and right division (`/ \`) is the same as multiplying by inverse.

Example 5

Try to understand the output of these commands:

```
>> a*b
>> A*a
>> A*B
>> A.*B
>> A.^3
>> A'
>> inv(A)
>> A./B
>> A/B
>> A\B
```

Example 6 (Vector Operations: Elapsed Time)

Goal: Calculate how many seconds have elapsed since the start of class.

- In `helloWorld.m`, define variables for time conversions:
  - `secPerMin`, `secPerHour`, `secPerDay`
  - `secPerMonth` (assume 30.5 days/month), `secPerYear` (12 months/year)
- Create a row vector `secondConversion` in the order:

```
[secPerYear, secPerMonth, secPerDay, secPerHour, secPerMin, 1]
```

- Use `clock` to get the current time: `currentTime = clock;`
- Compute elapsed time: `elapsedTime = currentTime - start;`
- Compute scalar `t` as: `t = secondConversion * elapsedTime';`
  (Transpose as needed for dimension compatibility)

Example 7 (Vector Operations: Current Knowledge)

Goal: Display your current state of MATLAB knowledge.

- Use the same model as before:

$$k(t) = 1 - e^{-t/\tau}$$

- Compute `currentKnowledge` using the value of `t` and previously defined `tau`.
- Display the message:

```
At this time, I know X% of MATLAB.
```

Replace `X` with the percentage value (use `num2str(currentKnowledge * 100)`)

Automatic initialisation

- Initialise a vector of ones, zeros, or random numbers:

`>> ones(1,10)`	Row vector with 10 elements, all 1s
`>> zeros(20,1)`	Column vector with 20 elements, all 0s
`>> rand(1,45)`	Row vector with 45 random elements of uniform(0,1)
`>> nan(1,50)`	Row vector of NaNs (uninitialised variables)

- Initialise a linear vector of values:

`a=linspace(0,10,5)`	5 value vector that starts at 0 and ends at 10

- The colon (`:`) operator does the same task:

`b=0:2:10`	Starts at 0, increments by 2, and ends at or before 10 Increment can be decimal or negative
`c=1:5`	If increment is not specified, default is 1

Example 8 (Vector functions)

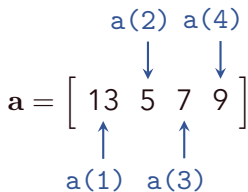
Goal: Calculate your learning trajectory.

- In `helloWorld.m`, make a linear time vector `tVec` that has 10,000 samples between 0 and `endOfClass`
- Calculate the value of your knowledge (`knowledgeVec`) at ech of these time points using the same equation as before:

$$k = 1 - e^{-t/\tau}$$

Vector indexing

- **MATLAB indexing starts with 1 (not 0)**
- `a(n)` returns the n^{th} element:



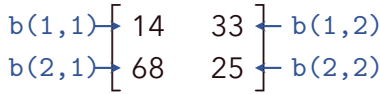
- The index argument can be a vector, where each element is looked up individually and returned as a vector of the same size as the index vector:

```
>> x = 0:2:100;
>> k = [12 13 5 8];
>> x(k)a
```

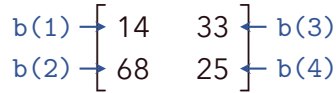
Matrix indexing

Matrix indexing can be done in two ways:

- using **subscripts (row, column)**:



- using **linear indices** (as in vectors):



Picking submatrices:

```
>> A=rand(5)          % a shorthand for 5x5 matrix
```

Advanced indexing

- Suppose we have $\mathbf{c} = \begin{bmatrix} 12 & 5 \\ -2 & 13 \end{bmatrix}$, then

```
>> d=c(1,:)          % selects all elements in 1st row
>> e=c(:,2)          % selects all elements in 2nd column
>> d=c(2,:)=[3 6]    % replaces second row of c
```

- MATLAB contains functions to help find desired values:

```
>> vec = [5 3 1 9 7]
>> [minVal,minInd] = min(vec);      % gets the minimum value and its index
>> [maxVal,maxInd] = max(vec);      % gets the maximum value and its index
```

Advanced indexing

- To find indices of specific values or ranges:

```
>> ind = find(vec==9); vec(ind) = 8;
>> ind = find(vec>2 & vec<6);
```

  In most cases, **logical indexing** is faster than `find`! For example,

```
>> vec(vec==9) = 8;
```

Example 9 (Indexing)

Goal: When will you know 50% of MATLAB?

- Find the index where `knowledgeVec` is closest to 0.5. Mathematically, you want the index where the value of $|\texttt{knowledgeVec} - 0.5|$ is at a minimum (use `abs` and `min`)
- Then use that index to look up the corresponding time in `tVec` and name this time `halfTime`.
- Convert `halfTime` to days by using `secPerDay` and finally, display the string:

```
I will know half of MATLAB after X days.
```

Plotting

- A simple example:

```
x = linspace(0, 4*pi, 10);
y = sin(x);
plot(y); % plot values against their index
plot(x,y); % usually we want to plot y against x
```

- `plot` generates dots at each (x,y) pair and then connects the dots with a line.
- To make the plot of a function look smoother, evaluate at more points:

```
x = linspace(0, 4*pi, 1000);
plot(x,sin(x)); % x and y must be same size, or else you'll get an error
```

Example 10 (Plotting)

Goal: Plot the learning trajectory.

- In `helloWorld.m`, open a new figure (use `figure`)
- Plot the knowledge trajectory using `tVec` and `knowledgeVec`
- When plotting, convert `tVec` to days by using `secPerDay`
- Zoom in on the plot to verify that `halfTime` was calculated correctly