



Data analysis for Geoscience

GEO8026

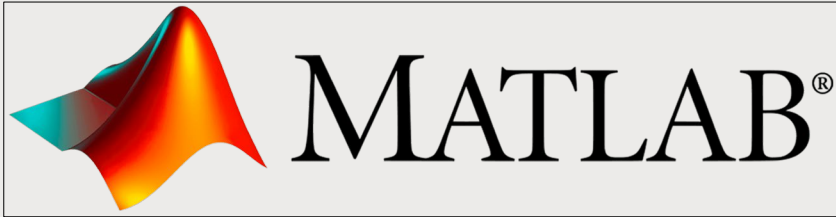
Dr. Matt Perks



MATLAB®

Course Introduction

Part 1: Data processing and analysis with MATLAB



Module delivered over two-weeks

Consisting of five 7-hour blocks:

1-hour lecture, 2-hour practical, 3-hr guided independent study, 1-hour drop-in

Assessment: 1500-word portfolio

Part 2: Statistical methods and graphical techniques with R



Five-week long module consisting of 5hrs/week

Assessment: 1500-word portfolio

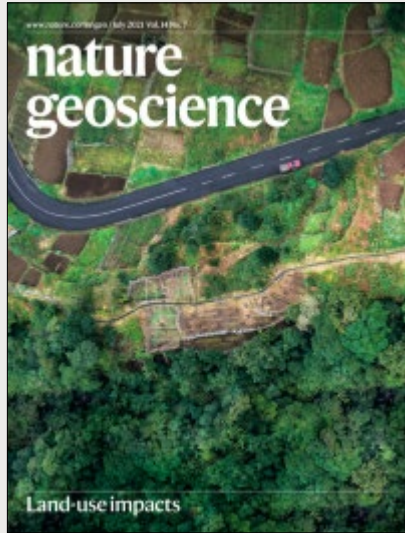
Part 1: Timetable

Timetable week	Date	Time	Location	Activity
Part 1: Data processing and analysis with MATLAB (MP)				
4	Monday 26 th September	09:00 – 10:00	HDB 3.101	Lecture 1: Introduction and MATLAB Primer
		10:00 – 12:00	HDB 3.101	Practical A1: MATLAB Primer
	Tuesday 27 th September	09:00 – 12:00	HDB 3.101	Practical B1: MATLAB challenges
		15:00 – 16:00	HDB 3.101	Drop-in 1
	Wednesday 28 th September	09:00 – 12:00	HDB 3.101	Practical A2: Working with interval data
	Thursday 29 th September	09:00 – 12:00	HDB 3.101	Practical B2: Working with your interval data
		14:00 – 15:00	HDB 3.101	Drop-in 2
	Thursday 30 th September	09:00 – 12:00	HDB 3.101	Practical A3: Quality control (I)

Part 1: Timetable

Timetable week	Date	Time	Location	Activity
Part 1: Data processing and analysis with MATLAB (MP)				
5	Monday 3 rd October	09:00 – 12:00	HDB 3.101	Practical B3: Quality control (II)
		14:00 – 15:00	HDB 3.101	Drop-in 3
	Tuesday 4 th October	09:00 – 12:00	HDB 3.101	Practical A4: Time-series analysis
	Wednesday 5 th October	09:00 – 12:00	HDB 3.101	Practical B4: Univariate and bivariate analysis
		14:00 – 15:00	HDB 3.101	Drop-in 4
	Thursday 6 th October	09:00 – 12:00	HDB 3.101	Practical A5: Image analysis
	Friday 7 th October	09:00 – 12:00	HDB 3.101	Practical B5: Image analysis
		14:00 – 15:00	HDB 3.101	Drop-in 5
12	Wed 23 rd November	12:00	Part 1: Portfolio submission deadline	

Characteristics of Geoscience Data



What are your topic(s) of interest?



What data are you planning on using?



What methods are you planning on using?



Code and data management



Generates a backup to the cloud which can be private or public (ensure private for this module)

Avoids having to keep track of multiple versions on your computer e.g. v1, ..., v9, etc

All changes to the uploaded files (commits) can be viewed and changes are reversible

Good practice when working within a team environment

Ensure that your data is hosted online to ensure outputs can be reproduced

This should be publicly accessible to ensure the code completes properly

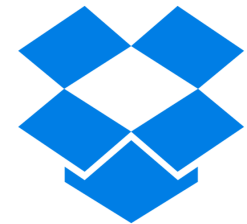
Needs to be shared during submission of portfolio



OneDrive

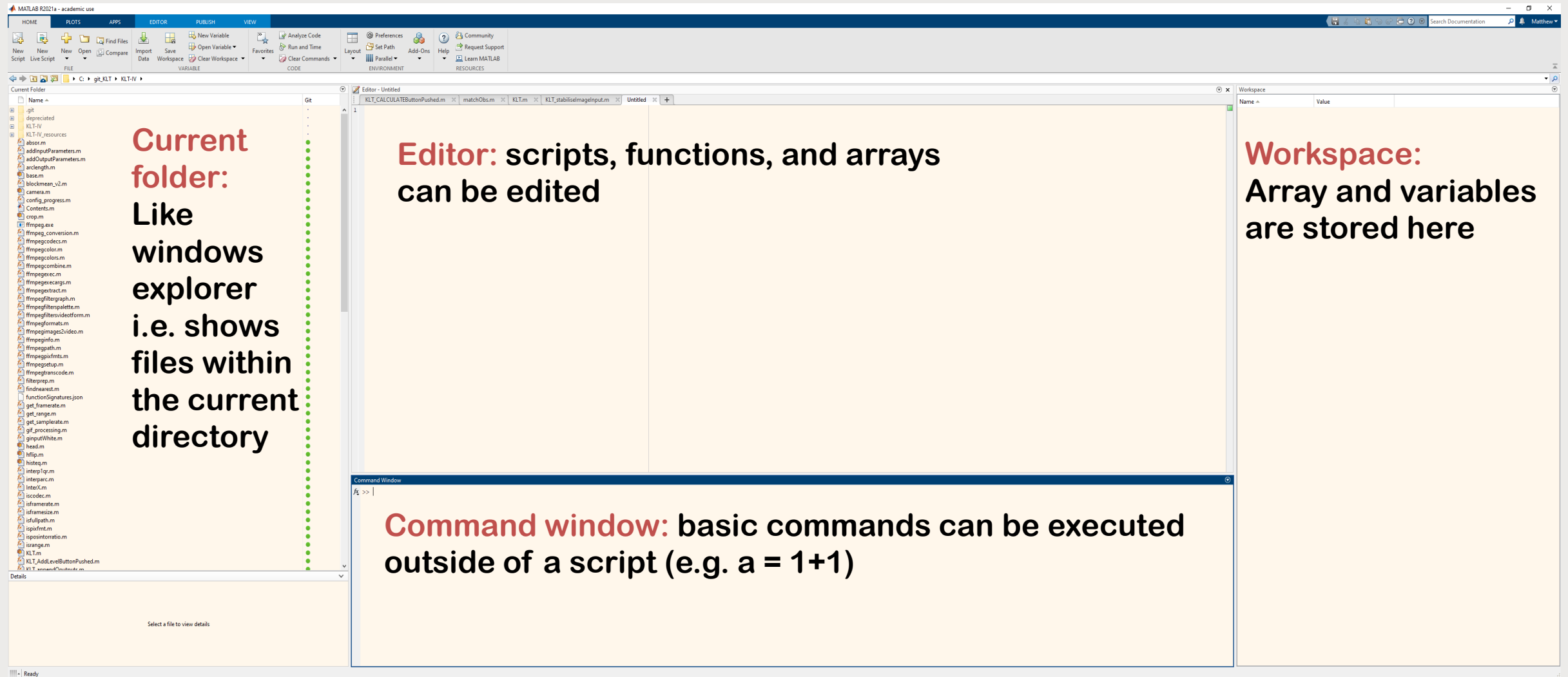


Google Drive

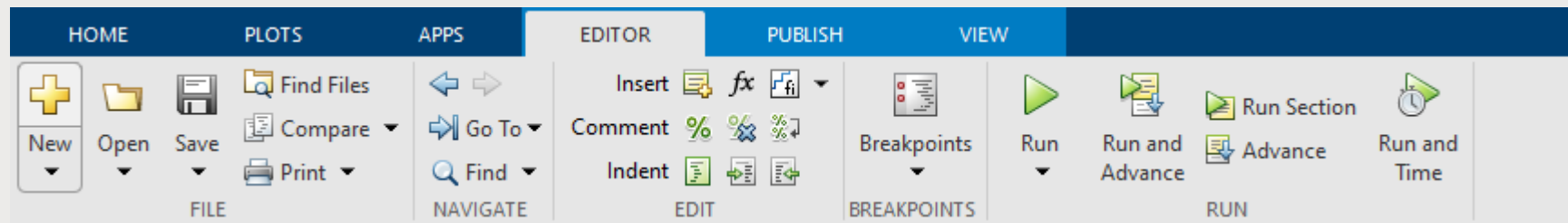
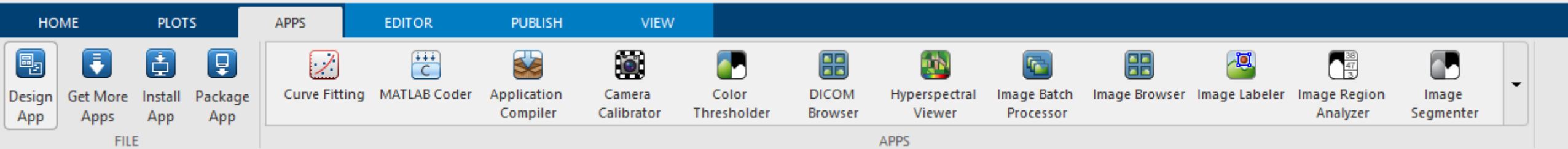
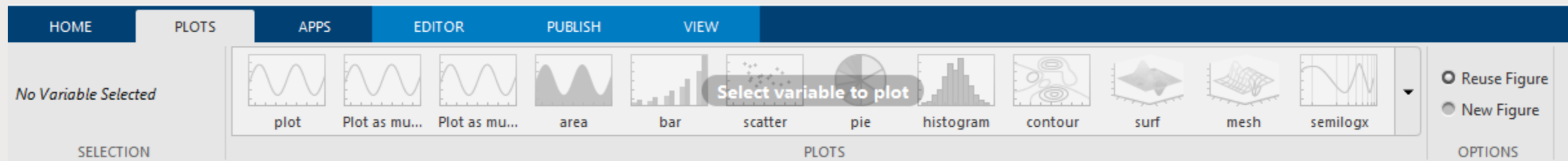
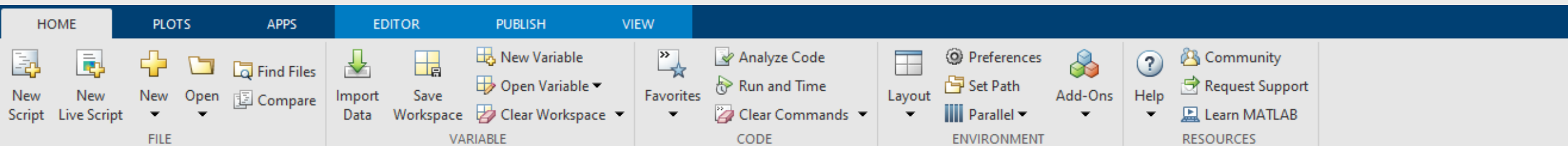


Dropbox

MATLAB primer



MATLAB primer



MATLAB primer

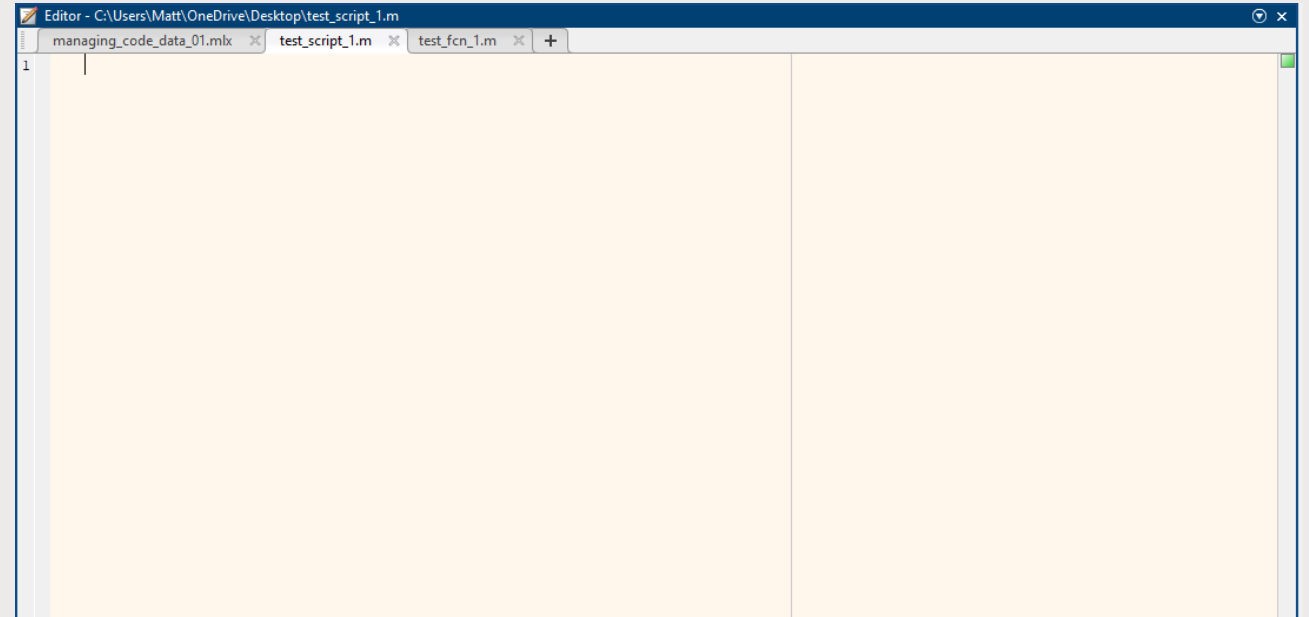
Scripts

Allow you to reuse sequences of commands by storing them in code files

A list of commands that are run from the current workspace.

All variables within the workspace can be called and all variables generated are stored in the workspace upon completion

Can be called/run from the command window, or from other scripts/functions



An example command:

```
run 'test_script_1.m'
```

MATLAB primer

Functions

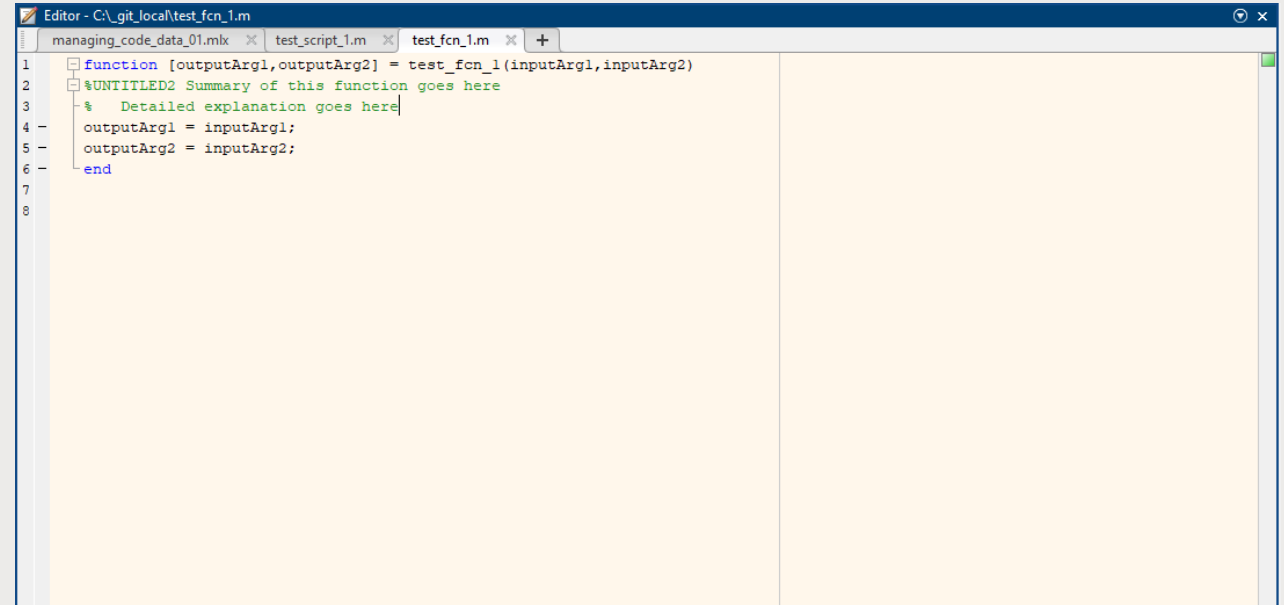
Allow you to reuse sequences of commands by storing them in code files

A list of commands that are run from the current workspace.

Only the variables defined as inputs are available within the function

Only the output variables are added to the Workspace upon completion

Can be called/run from the command window, or from other scripts/functions

A screenshot of the MATLAB Editor window. The title bar shows the file path 'C:_git_local\test_fcn_1.m'. The editor has several tabs open: 'managing_code_data_01.mlx', 'test_script_1.m', and 'test_fcn_1.m'. The active tab 'test_fcn_1.m' displays a function definition. The code is as follows:

```
1 function [outputArg1,outputArg2] = test_fcn_1(inputArg1,inputArg2)
2 %UNTITLED2 Summary of this function goes here
3 % Detailed explanation goes here
4 outputArg1 = inputArg1;
5 outputArg2 = inputArg2;
6 end
7
8
```

Input an output argument naming conventions

An example command:

```
[varOutA, varOutB] = test_fcn_1(varInA, varInB)
```


Interfacing with folders and directories

Over the course of this module, you will generate several scripts, functions, and use a range of datasets.

Organization of your files and folders is critical.

For MATLAB to call a script, function, or access a file, it needs to be visible on the 'search path'.

To add files and folders to the search path, you can use the `addpath` function. E.g. :

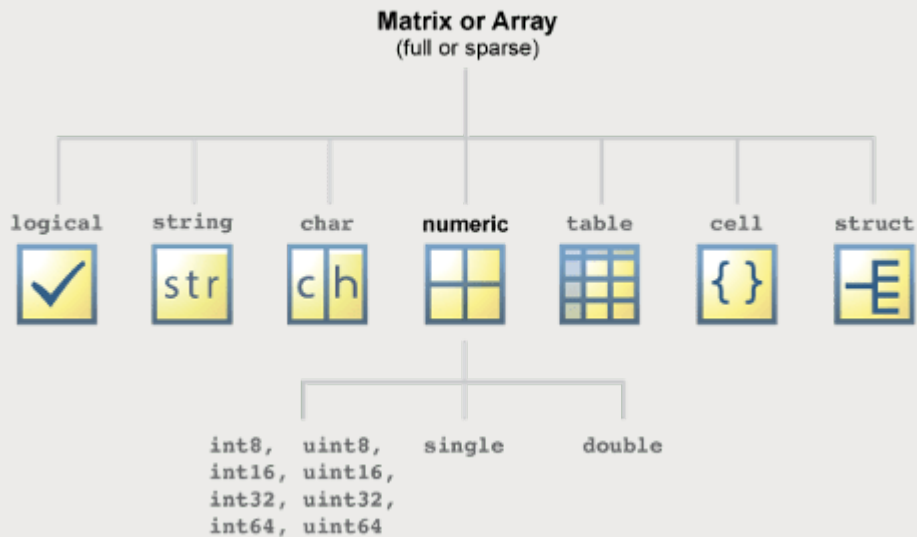
```
addpath(genpath('C:\Users\Matt\OneDrive - Newcastle University\GE08026'));
```

This will enable MATLAB to call and/or load files within this folder and all subfolders.

Your University OneDrive account is useful for storing your files (5 TB personal storage space)



Data types



Useful command:

whos

List the variables within the workspace and identifies the data type

Logical = Boolean = 0 or 1, false or true

String = collection of characters (letters)

Character = each letter is treated separately

```
s = "apple"; f = s(1) % f = "apple"
```

```
c = 'apple'; f = c(1) % f = 'a'
```

Cell = Treated as non-numeric (but can be numeric)

```
c2 = {'apple', 0};
```

Numeric = only numbers (and NaN).

Double precision (default, 64 bits) most widely used

Single precision (32 bits)

int8, int16, ... = Variables with ... bit signed integers

uint8, ... = Variables with ... bit non-negative integers

Table = mixed input (numeric and string)

Structure = groups related data using data containers called fields

Logic & relation operations

Logic

Used to assess whether a command meets a requirement

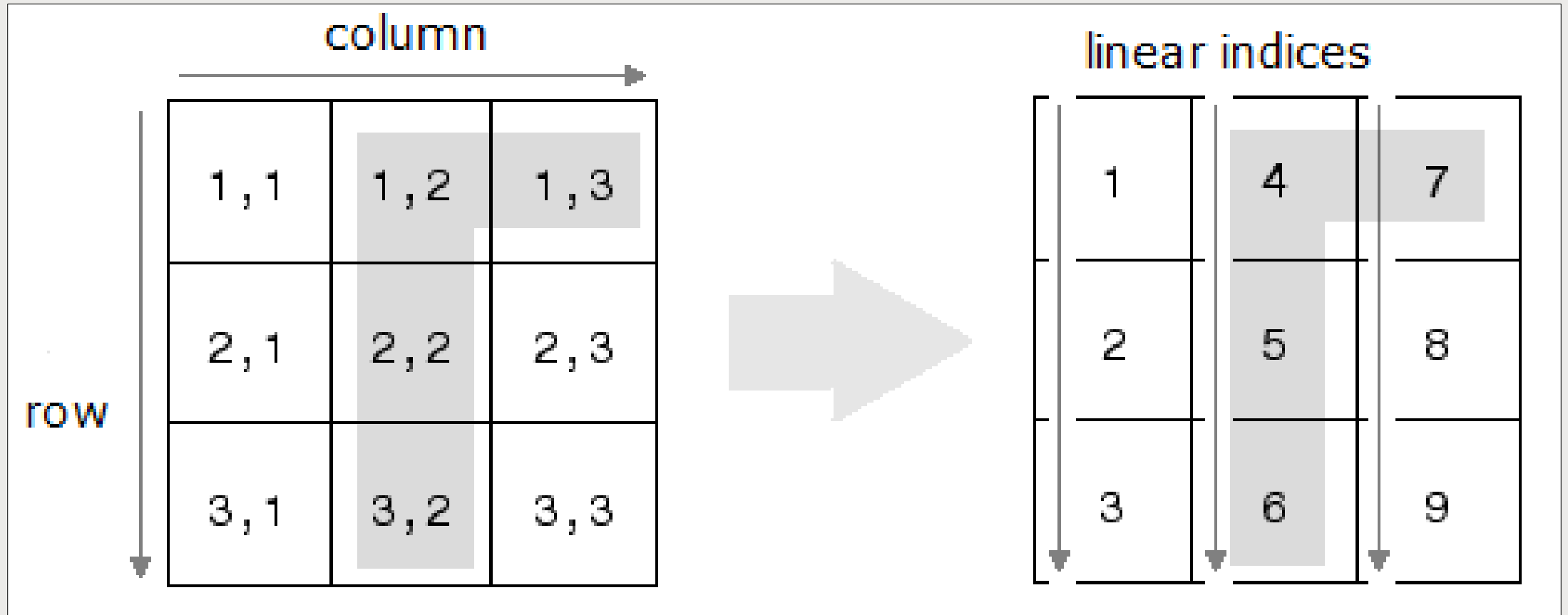
~	NOT: Does not
&	AND: used to link expressions
	OR: used to link expressions
&&	Short-circuit &: Used on two scalar expressions
	Short-circuit OR: Used on two scalar expressions

Relation

Used to assess whether a command meets a requirement

==	Equal to
~=	Not equal to
<	Less than
<=	Less than or equal to
>	More than
>=	More than or equal to

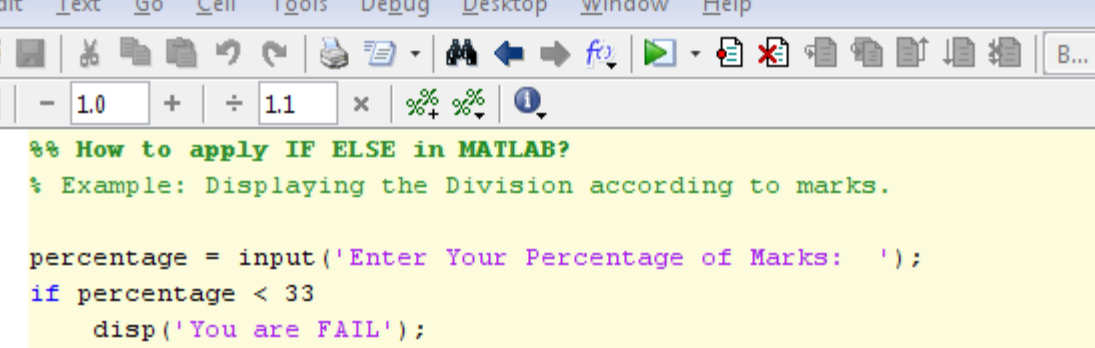
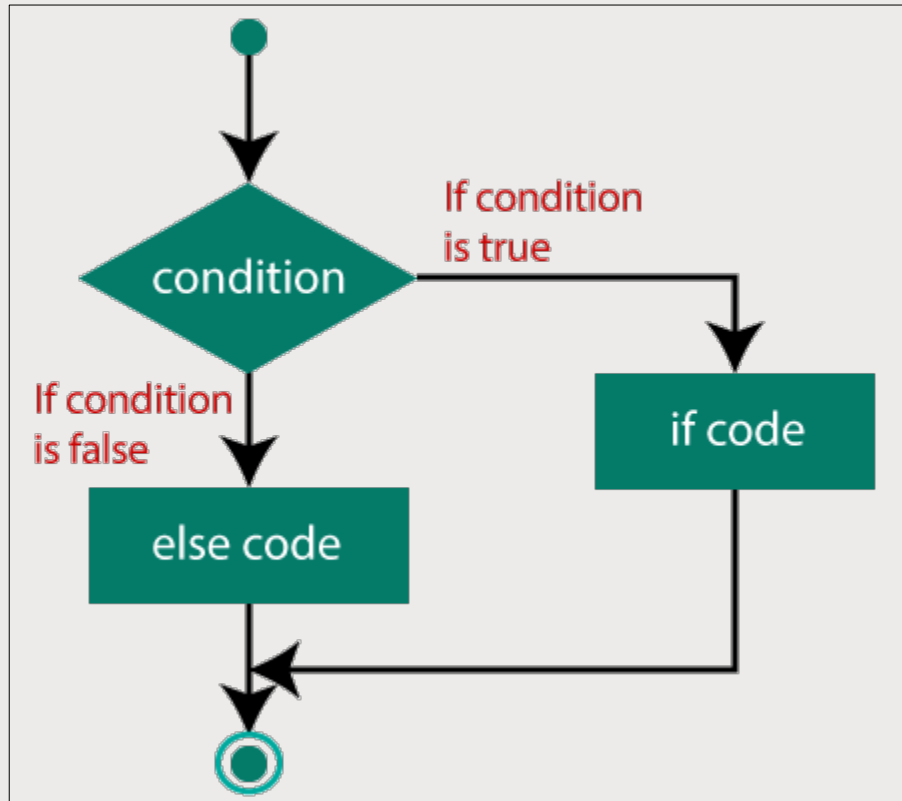
Array indexing



Matrix indexing
 $A(1,2)$

Linear indexing
 $A(4)$

Condition statements



```
1 %% How to apply IF ELSE in MATLAB?
2 % Example: Displaying the Division according to marks.
3
4 percentage = input('Enter Your Percentage of Marks: ');
5 if percentage < 33
6     disp('You are FAIL');
7 elseif percentage >= 33 && percentage < 45
8     disp('You are passed with THIRD Division')
9 elseif percentage >= 45 && percentage < 60
10    disp('You are passed with SECOND Division')
11 elseif percentage >= 60 && percentage < 75
12    disp('You are passed with FIRST Division')
13 elseif percentage >= 75
14    disp('You are passed with HONOURS')
15 end
16
17 %% MATLAB | MODULE-01 | POST-17 | AVAILABLE ON: INFO4EEE,"info4eee.com"
```


Loops (for and while)

For loop

```
% Example 1
X = randn(100,1) % create a 100x1 array of random numbers
for a = 1:length(X) % a increases in value by +1 each time the loop is executed
    if X(a,1) < 1 % we call the particular cell for each loop
        X(a,1) = NaN; % replace all negative values with NaN
    end
end
```

```
% Example 2
grades = [88; 98; 87; 76; 89; 93; 95];
for index = 1 : length(grades)
    grades(index) = grades(index) + 3; % add 3 to each grade
    if grades(index,1) > 100 % if the mark is > 100
        grades(index,1) = 100; % assign it as 100
    end
end
```

While loop

```
% Example 1
X = randn(100,1) % create a 100x1 array of random numbers
timer1 = 0;
tic % start a stopwatch
while timer1 < 10 % while time is less than 10 seconds
    X = X + 1; % add one to X
    timer1 = toc % find out how long stopwatch has been running and save result to timer
end
```

```
% Example 2
n = 0;
x = abs(randn(1,1)*10000000); % choose a very large random number
while abs(x) > 1 % while x > 1
    x = x/2; % divide x by 2
    n = n + 1; % increase n by 1
    if n > 5 % if over 5 loops have been executed
        break % exit out of the loop
    end
end
clear all
```

Vectorization

For loop

```
% Example 1
X = randn(100,1) % create a 100x1 array of random numbers
for a = 1:length(X) % a increases in value by +1 each time the loop is executed
    if X(a,1) < 1 % we call the particular cell for each loop
        X(a,1) = NaN; % replace all negative values with NaN
    end
end

% Example 2
grades = [88; 98; 87; 76; 89; 93; 95];
for index = 1 : length(grades)
    grades(index) = grades(index) + 3; % add 3 to each grade
    if grades(index,1) > 100 % if the mark is > 100
        grades(index,1) = 100; % assign it as 100
    end
end
```

Cellfun

Advanced way of calling operations on each cell using anonymous functions

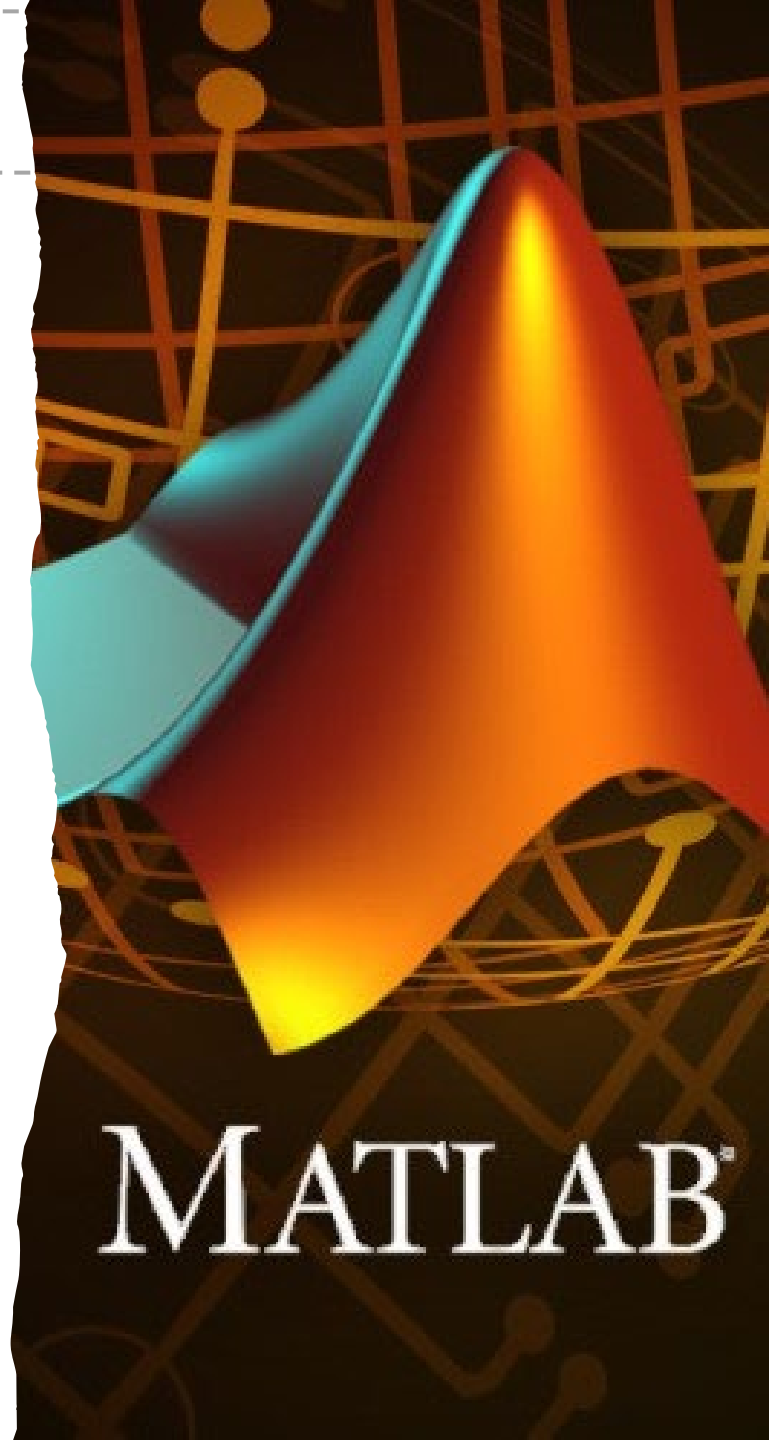
Vectorised version

```
% Example 1
X = randn(100,1) % create a 100x1 array of random numbers
idx = X < 1; % find all values < 1
X(idx) = NaN;

% Example 2
grades = grades + 3; % add three to all grades
idx = grades > 100; % find those above 100
grades(idx) = 100; % limit to 100
```

Summary

- Overview of the importance of source control
- Introduction to the GUI of MATLAB
- Interfacing with files and folders
- Data types
- Logic and relation operations
- Array indexing
- Condition statements
- Loops and vectorisation



Next Session

Practical A1:

Themes:

- 01: Managing code and data
- 02: MATLAB primer
- 03: MATLAB challenges

Room: HDB 3.101

Time: 10:00 – 12:00

