Introductory Worksheet

A. Course introduction

Follow along with Jay's introductory presentation to the workshop and assignment.

B. Introduction to Matlab

1. Complete short introductory presentation
2. Download workshop .zip file and assignment sheet from link in A2L (/Content/Climate Change/Individual Assignments/Introduction to Scientific Computing/
3. Extract to: C:\Users\\Documents\

- Make a folder called MATLAB. Copy the zip file there - Right click > 7-Zip > “Extract Here” - When extracted, you should have the following structure: - C:....\MATLAB\Data\ - C:....\MATLAB\Figs\

B0. Matlab interface

1. Explore the interface layout. Know the name of each panel, and briefly describe what they do.

- Change Matlab's working directory to your newly-created MATLAB folder

B1. Basic command window operation

1. Create a numeric variable using the command window
    - Create a numeric array and assign it a custom name
    - ***hint*** `a = 54` creates a variable named 'a' with a value of 54
2. Create a row vector, a column vectors and a matrix
    - ***hint****: use brackets [ ] to generate vectors and matrices, commas or spaces to separate values in the same row, and semi-colons to indicate new rows.*
    - Create an **5 x 1** column vector; assign it to a named variable. e.g. `cv = [4; 3; 1; 3; 1]`
    - Create a **1 x 5** row vector; assign it to a new named variable
    - Create an **4 x 3** matrix; assign it to a new named variable
3. Create a new vector or matrix that contains a not-a-number (NaN) value to indicate a missing (or null) value.
    - ***note:*** The value NaN is a special character recognized by MATLAB as a null value. Any normal operations that are carried out on variables containing NaNs will result in NaN.
4. Use indexing to create an array that consists of sub-elements of the matrix you created earlier
    - ***example***: `d = matrix1(2,3)` creates a variable *d* containing the value in the second row and third column of *matrix1*
    - ***example2***: `e = matrix1(:,3)` creates a variable *e* containing all values in the third column of *matrix1*
5. Perform basic arithmetic [+ - / *] operations on your arrays (try scalars, vectors and matrices).
    - Do you encounter any issues?
    - try `sqrt(d);` , `log;`

6. Perform basic statistical functions on your arrays
    - e.g. `mean(d);`, `std(d);`, `median(d);`

B2. Learning more about Matlab

- There are two helpful commands for learning more about what a matlab function (like 'mean', for example) does:
    - To view help documentation in the command window type **help** and the name of the function
    - e.g. `help mean;`
    - To view documentation in a separate window, type **doc** and the name of the function
    - e.g. `doc mean;`

B3. Working in a script (instead of the command window)

Open a new script, work through the rest of the examples in there. *-**tip:*** you can send a line (or many lines) of code to the command window by highlighting them and pressing **F9**

B4. Types of MATLAB variables

MATLAB provides for a number of different variable types to be used. Typically, the most appropriate variable type depends on the nature of your data and your desired processes and outcomes. Here are the MATLAB variable types:

- **integer** arrays (of varying length, signed/unsigned)
- **numeric** arrays (real numbers)
- **logical** arrays (values of 1 or 0, representing true and false)
    - e.g. `f = [12 -3 NaN 7];` `list_nans = isnan(f)`
- **character** arrays (strings stored as vectors of characters)
    - e.g. `char1 = 'my first string'` `char2 = 'is not my last'`
- **cell** arrays (array of indexed cells, each capable of storing an array of different dimension and type; think of Excel spreadsheets, if each cell could contain countless more cells inside of it)
    - e.g. `cell1 = {345, 'Mazda', [34 43]; 34, NaN, 'zoom'}`
- **structure** arrays (expandable tree of variables, which can each be of different size and type)
    - e.g. `s.a = 1; s.b = {'A','B','C'};`
- **objects** (user classes and java classes)

Below, you'll learn a bit more about basic array types.

Numeric arrays

You've already had a basic introduction to numeric arrays through your work with scalars, vectors and matrices.

Character arrays

1. Create two character arrays using strings of characters; assign them variable names.
    - e.g `char1 = 'my first string'` `char2 = 'is not my last'`

2. Combine (concatenate) the two character arrays into a new single character array
    - e.g. `char3 = [char1 char2];`
3. Figure out how to insert another string between char1 and char2

Cell arrays

1. Create a 2x2 cell array, where one of the cells contains:
    - a scalar (i.e. a single numeric value) e.g. `cell1{1,1} = 99;`
    - a character array e.g. `cell1{1,2} = 'bottles of beer on the wall';`
    - a matrix
    - a NaN

Structure arrays

1. Enter the following

- `student(1).name = 'John';` - `student(1).age = 23;` - `student(2).name = 'Sabrina';` - `student(2).age = 24;` 2. Create a new type of category for both students named 'grade' and give each student a value.

B5. Running your newly-created script

- In the command window, clear all variables from the workspace with the `clearvars;` command
- Run your script using the 'play' button. Note that you will be required to save it. Save it to the same directory as the rest of your work.

B6. Functions

A function is similar to a script in that it is a list of commands to be executed. Where it differs, however, is that functions run within their own personal Workspace (called a 'stack'). By default, internal variables will not be placed in the Workspace, and functions will not be able to 'see' any existing variables in your Workspace.

You pass data into and out of functions using the function declaration statement at the top of the function. - It looks something like `function [output1 output2] = name_of_function(input1 input2 input3)`

Functions can have any number of input and output variables, and can be named mostly anything (as long as they don't have a special character or a space in them).

Functions are useful when you are interested in only select output variables in a set of commands. You can also run a function inside of a script (or another function), which makes it a very space and time-saving way to run many commands.

Functions are called by using their name and the output/input format specified above (see below example). For example, the operation 'mean' is a function.
- It is called by, e.g. `[average] = mean(values);` where 'average' is the output, 'mean' is the function name, and 'values' are the input.

1. Open \Other Scripts\isleapyear.m.

- What does it do? How can we use it? 2. Figure out how to use isleapyear from the command line.

B7. Control structures and indexing

- Create a new blank script

For loops

- Loops are handy when you need to repeat a process many times, such as when reading through many rows or columns of data, or running a process iteratively.
- The **for** function allows you to do this. Learn more by typing into the command window `doc for;`

1. In your new script, create a for loop that increments an index **yr** from 1000 to 2020 by 1.

- within the loop, call the isleapyear function, using the value of yr. Set the function to be verbose.

If statements

- An 'if' statement evaluates if a condition is met (true) or not met (false), and allows the user to perform different functions based on the outcome.
    - If the statement is true (condition is met) Matlab will execute whatever commands are below it.
    - If it is untrue, it will execute any commands that are below the 'else' statement.
    - Just like 'for' loops, 'if' statements have to be closed with an 'end'

1. Within the existing for loop, add an if statement that checks if two conditions:

- That the value of **yr** is a leap year (***hint*** you'll use the output from isleapyear) - That the value of **yr** is a prime number (***hint*** you'll use the function **isprime**) 2. If both conditions are met, use the **disp** function to send a message to the command window.

Indexing and the 'find' function

1. Use the **randi** function to create a variable **rand_nums** that is a *10000 x 1* list of random numbers between 1 and 1000
2. Use the **find** function to create a list of:

- All rows where rand_nums are greater than (>) 500 - All rows where rand_nums is less than or equal to (<=) 100 - All rows where rand_nums is between 100 and 700 - All rows where rand_nums is exactly 999

Creating your own function

3. Finally, make your own function (call it **leap_and_prime**). This function should:

- take an integer year as input - check if it is a leap year *and* a prime number - display a message if both are true.