Chapter 01 - The MATLAB Language and Desktop Environment.

Author: Ken Deeley, ken.deeley@mathworks.co.uk

This chapter provides the introductory MATLAB material necessary to work effectively in the MATLAB environment. It serves as a mini "crash-course" covering the fundamental programming techniques and syntax which are required for the remainder of the bootcamp. It's not recommended to enter the bootcamp with no previous MATLAB knowledge, although a very intelligent person should be able to follow the gist of this chapter without getting too much into the low-level programming details. As pre-work for the bootcamp, we strongly recommend the interactive online tutorial available free of charge for academic users at: https://www.mathworks.co.uk/academia/student_center/tutorials/

This contains more than three hours of introductory MATLAB materials and provides suitable prerequisite training prior to attendance at a MATLAB-based bootcamp.

Outline:

- The MATLAB Desktop
- Importing data from one file
- Importing data from multiple files
- Indexing into vectors and matrices to retrieve data
- Concatenating vectors and matrices to construct data
- Removing missing values from data
- Basic plot options
- Annotating plots
- Cell and structure arrays
- Saving data to MAT-files
- Running and publishing scripts
- Code sections

Reference files for this chapter:

- ../MedicalData.txt
- ../HeightWaistData.txt
- ../ArmsLegs/*.txt
- ../Reference/S01_Import.m
- ../Reference/S01_HealthData.mat

Course example: Biomedical data.

We will work with an anonymised biomedical data set from the 2007-2008 US National Health and Nutrition Examination Survey (NHANES). This data is freely available at: http://www.cdc.gov/nchs/nhanes/continuousnhanes/

The idea behind choosing this dataset is that delegates from all cultural and language backgrounds should be able to relate to this data. Measurements such as arm circumference, height, weight, age are applicable to everyone.

The MATLAB Desktop.

It's worth introducing the four main components of the MATLAB Desktop (the screen that appears when MATLAB is started). The four main components are:

- The Current Folder Browser (CFB)
- The Command Window
- The Command History
- The Workspace Browser

The MATLAB Editor.

The MATLAB Editor is used to write, edit, run, debug and publish MATLAB code. We will collect our code into a script in this initial chapter of the course. Either open the Editor, and create a new script, use the 'New Script' or 'New Live Script' buttons or use the blank code spaces in this live script. You might find it easier to work with the Editor window docked into the main MATLAB Desktop, so that it's easy to transfer code from the Command Window and History into the Editor. Modify your Editor preferences via the main MATLAB Preferences dialog:

Rerun Favourite Commands

We can create customised tools for favourite commands which are executed regularly. As an example we will create a tool which clears the screen and the workspace.

1. On the **Home** tab, in the **Code** section, click **Favorites** and then click **New Favorite**. The Favorite Command Editor dialog box opens.
2. In the **Label** field, enter a name for the favourite command. For this example, enter Clear All.
3. In the **Code** field, type the statements you want the favourite command to run. You also can drag and drop statements from the Command Window, the Command History Window, or a file. MATLAB automatically removes any command prompts (>>) from the **Code** field when you save the favourite command.
4. In the **Category** field, type the name of a new category or select an existing category from the drop-down list. If you leave this field blank, the favorite command appears in the default **Favorite Commands** category.
5. In the **Icon** field, select an icon.
6. To add the favourite command to the quick access toolbar, select both the **Add to quick access toolbar** and **Show label on quick access toolbar** options.
7. To run the statements in the **Code** section and ensure that they perform the desired actions, click **Test**.
8. When you are done configuring the favourite command, click **Save**.

Code Sections and Live Editor.

Best practice is to write code in sections when developing scripts. This is strongly recommended, for ease of debugging and publishing. In a .m script use %% to create a new section. A common mistake here is not to type the space after the two percentage signs.

In a .mlx script using the Live Editor use the Section Break (Ctrl+Alt+Enter) or Insert->Section Break to create a new section.

Help and Documentation.

How do we know how to get started, and which functions we need to use? There are two main entry points into the documentation:

- If the function name is known, use the F1 key to access immediate pop-up help.
- If the function name is unknown, search in the box in the top-right.

Function hints can be shown with the shortcut Ctrl+F1 when the cursor is located within the brackets of a function call.

Import data from a single file containing heterogeneous data.

We will use a table to hold the results. Tables are for storing heterogeneous tabular data.

```
% Inspect the data file:
```

Use READTABLE to import the data. If unfamiliar with this function, use the F1/search in the documentation. Search for "Supported File Formats" to bring up a table of available import/export functions in the documentation, and note the READTABLE function.

```
% Use READTABLE to import the data.
```

Why are people so old? They're not really, it's just that Age is recorded in months, so we might want to convert the data.

Table variables are accessed using the dot syntax (similar to structures, if people are familiar with those).

```
% Note: age is recorded in months.
```

Best practice would be to have comments explaining any "magic numbers" in the code. For example, where did the 12 come from?

```
% As an alternative, we could use "self-commenting" variables, e.g.
```

Compute pulse pressure and store this as an additional table variable.

```

```

What about naming conventions? Generally we would use camel case. Table/structure variable names can be capitalised. Try to decide on some convention, and then stick with it.

Plotting vectors and annotating graphs.

How can we visualise the information we have just computed. Note the different types of plots available in the Plots tab.

```
% New figure.
```

Basic discrete plot and annotations.

```

```

Low-Level File I/O.

Suppose we have data from a file containing numeric data. How can we read this in efficiently? Three-step process, using textscan for text files.

The open/close statements are essential.

```
% Open file HeightWaistData.txt
```

The next step is critical. This is where you can control the precise format MATLAB will use to import the data. See the documentation for textscan for more details.

```
% Two columns of numeric (double, floating point) data.


% Close file
```

Do we really need to use DOUBLE to store the data? How many significant figures are there? What else could we use to save memory? (SINGLE, specified via %f32 in TEXTSCAN).

Cell Arrays.

The data from textscan is imported as a cell array in the MATLAB workspace. The data is imported into a 1x2 cell array (each column of data is stored in a separate cell). A useful visualisation function for inspecting cell arrays is CELLPLOT. This can help to alleviate misunderstanding of how cell arrays work.

```
% Display cell data
```

Converting Data Types.

All the elements of a MATLAB array must be the same type. This rule is particularly important when attempting to concatenate data of different types (e.g. strings and numbers). In the documentation, you can see the reference page at: MATLAB -> Language Fundamentals -> Data Types -> Data Type Conversion which has a comprehensive list of conversion functions.

```
% Convert the data to a useful format.
```

Alternatively, you could use heightWaist = [heightWaistData{1}, heightWaistData{2}] or even heightWaist = [heightWaistData{:}].

```
% Try computing the mean of each variable.
```

Why does this not give us "proper" values? NaNs - missing data. We would like to remove them to obtain proper results. However, we need to know how to index into arrays before attempting this.

Accessing Data in Arrays.

Data in MATLAB can be accessed using several indexing methods. Generally, row/column, or subscript, indexing is the natural method to introduce first. Useful tips here include the use of the "end" keyword, as well as the use of the colon as an index as a shortcut for 1:end. Row/column indices may have vector as well as scalar values.

```
% Extract height and waist data separately.
```

Dealing with Missing Data.

This requires use of logical indexing (passing logical arrays as indices to other arrays). The key function here is ISNAN. The documentation page 'Use is* Functions to Detect State' has a list of all useful functions for detecting certain conditions or states (enter "is* functions" in the search bar to find this). Also, the reference page >> doc ops has a list of all MATLAB's operators, including logical operators which are required here.

```
% Remove any observations containing NaNs.
```

Importing Data from Multiple Files.

We have seen how to import homogeneous and heterogeneous data from single files. Often, researchers have multiple data files with similar formats, so it is useful to know how to batch process a list of files. The documentation contains a list of functions for performing file and directory management, such as changing directory, moving files, and creating directories: MATLAB -> Programming -> Files and Folders -> File Operations.

The first step is to form a list of files to import data from.

Warning: be careful here to avoid platform-dependent issues. Use of FILESEP is generally recommended to avoid hard-coding slashes or backslashes. This should recover a list of .txt files in the "ArmsLegs" directory.

```
% Form a list of files to import data from.
```

Programming Constructs.

Here, we need to operate on each data file. We know how many data files there are, so a for-loop is appropriate. Use ISKEYWORD to display a list of MATLAB's programming keywords.

Structure Arrays.

We would like to store the name of each file (a string) as well as the numerical data it contains (a matrix with three columns) together in the same variable. Tables are not suitable for this, because in a table all variables must have the same number of observations. We can use a structure array instead. This allows access to the data using the same convenient dot syntax we used for tables, but also allows storage of mixed-size data. (However, structures have far less available functionality associated with them than tables. For example, SORTROWS comes with tables but not with structures.)

```

```

Concatenation.

After reading all the data, we would like to merge it into one variable so that we have all of the body measurements together. Data can be concatenated using square brackets [ ].

```

```

For comma-separated lists, VERTCAT, HORZCAT and CAT can be used.

```
% Concatenate all of the numerical data into one variable
```

Logical Indexing.

We have seen logical indexing above. We used it to detect and remove missing values. It is an important and widely applicable MATLAB language feature. For example, we might want to visualise the split/separation between two different groups in our population, e.g. males and females.

```
% Create logical array showing the positions of 'F' in medData.Sex
```

```
% Create 3-dimensional scatter plot of ArmLeg data for all females
```

```
% Add data for males to the same graph and label axes

```

Saving and Loading MAT-Files.

We have spent a lot of time and effort into importing our data effectively. Do we really want to repeat this effort if we need the data again? Probably not. You can save all the data to disk using a MAT-file (designed to hold MATLAB data and variables). Loading from MAT-files is also faster in general terms than running file import code. The idea is to import the data only once, and then save it to a MAT-file for future use or distribution to others.

```
% Save data to a MAT-file.

```

Publishing Code.

At this point, we have a nice script summarising what we've done. To share results with others and for the purposes of presentation, we can export the MATLAB code to a supported external format (HTML, XML, TEX, PDF, DOC, PPT). Publishing code is a key reason to use sections and the live editor when writing MATLAB scripts, as sections will automatically be interpreted as boundaries in the published documents. There are many text mark-up options available from the "Live Editor" and "Insert" menus, including insertion of LaTeX code, bullet lists, enumerated lists, images and hyperlinks.

New MATLAB Data Types

Several new data types and data functionality have been added in recent MATLAB releases. There is good documentation and some very informative live scripts which you may like to explore to find out more about these.

Timetables

These are a type of table which associate a time with each row. They may be useful for processing time-stamped experimental data.

```
doc timetable
```

The examples tab in the documentation gives some live scripts which demonstrate the use of timetables.

Dictionaries

This is a data type which uses a map which stores data as values which can be accessed using unique keys. Each key/value pair is an entry.

```
doc dictionary
```

Again, the examples tab in the documentation gives a live script which demonstrates the use of dictionaries. This data type may be useful where fast access of large datasets is required.

Pivot

This is a new function which has been added to allow summarisation of tabular data in a pivoted table.

```
doc pivot
```