Appendix A - Exercises.

Authors: Ken Deeley, ken.deeley@mathworks.co.uk Juan Martinez, juan.martinez@mathworks.co.uk

This appendix contains several hands-on exercises designed to provide practice with the concepts covered during each chapter of the course. Full worked solutions are available for each exercise. These materials are stored in the "Exercises" folder of the main course directory.

The MATLAB Language and Desktop Environment.

Exercise - Gas Prices Solution - Ex01_GasPrices.m

- Create a new script, and use the readtable function to import the data from gasprices.csv into a table in your Workspace. Hint: You will need to specify the Delimiter and Headerlines options – see the documentation for the `readtable` function.
- Extract the data for Japan from the table into a separate numeric variable, using the dot syntax (.). Compute the mean and standard deviation of the Japanese prices.
- Extract the data for Europe into a numeric array, and compute the mean European price for each year. Hint: check the documentation for the `mean` function.
- Compute the European return series from the prices using the following formula:

```
R(t) = log( P(t+1)/P(t) )
```

Here, P(t) represents a single price series. There are four price series in the European data.

- Compute the correlation coefficients of the four European return series. Hint: See the documentation for the `corrcoef` function. Display the resulting correlation matrix using the `imagesc` function.

The MATLAB Language and Desktop Environment.

Exercise - Australian Marriages Solution - Ex02_AusMarriages

- Create a new script, and use the `readtable` function to import the data from AusMarriages.dat into a table in your Workspace. Hint: You will need to specify the Delimiter option – see the documentation for the `readtable` function. This data file is tab-delimited ('\t').
- Insert an additional variable dn in the table containing the numerical representations of the dates. Hint: MATLAB uses serial date numbers to represent date and time information. Check the documentation for the `datenum` function.
- Plot the number of marriages as a function of time. Format the x-axis tick labels using the `datetick` function.
- Create a 1x25 row vector v of equal values 1/25. Hint: use `ones`.
- Use the `conv` function (with the 'same' option) and the vector from step 4 to smooth the marriage data.
- Overlay the smoothed data on the original plot, using a different colour. Hint: Use hold on.

Algorithm Design in MATLAB.

Exercise - Pacific Sea Surface Temperatures Solution - Ex03_SeaSurfaceTemperatures, SST_grid_sol

- Create a new script, and load the data from Ex03_SST.mat into the Workspace.

- Visualise the first set of temperature data (corresponding to the first column of the variable sst) using: a 2D scatter plot, specifying the colours using the temperature data; a 3D scatter plot, using black points as markers.
- Create equally-spaced vectors of points ranging from min(lon) to max(lon) and min(lat) to max(lat), respectively. Next, create grids lonGrid and latGrid of lattice points using `meshgrid`.
- Create a variable tempGrid by interpolating the first set of temperature data over the lattice of points created in the previous step. Hint: use `griddata`.
- Visualise the resulting interpolated values using the `surf` function.
- Write a function SST_grid taking the longitude, latitude and a set of temperature measurements as its three input arguments. The function should return lonGrid, latGrid and tempGrid as output arguments. Therefore, the first line of your function will be:

```
function [lonGrid, latGrid, tempGrid] = SST_grid(lon, lat, seaTemp)
```

- In your script, invoke your function for each different set of sea surface temperature measurements (i.e. the different columns of the sst variable), and visualise the results in a 4x6 subplot array. Hint: use a for-loop and the `subplot` function.

Algorithm Design in MATLAB.

Exercise - House Prices to Median Earnings Solution - Ex04_PricesToWages, plotPrices_sol, houseUI_complete

- Create a new script, and load the data from Ex04_House.mat into the Workspace.
- Open and view the data table in the Variable Editor. Extract the region names from the table as a cell array of strings, by accessing the Properties metadata of the table. Hint: At the command line, enter >> pricesToWages.Properties
- Use the `strcmp` function and the cell array from the previous step to identify the row of the table containing the data for Sheffield. Extract the data from this row, and plot this data as a function of year (note that the years range from 1997 to 2012).
- Write a function plotPrices with the following specifications:

```
plotPrices accepts a row number as its only input argument.

plotPrices plots the price information from the corresponding row of the     table as a
function of year
```

Note that since functions have their own private workspace, you will need to load the required data inside the function.

- Bonus: open the houseUI function file and modify the popup menu callback so that the plot is updated when the user selects a region of interest.

Test and Verification of MATLAB Code.

Exercise - String Subset Replacement Solution - test_repblank_sol

- Open the script app_repblank and run the code. The objective of this exercise is to write unit tests for the repblank function. Open the repblank function and read the help to understand the function's specifications and requirements.
- Open the main test function test_repblank. This main test function currently has four local test function stubs.
- Write code in the test_OneBlank local test function to verify that repblank exhibits correct behaviour when the input string contains one blank character (i.e. one space). Hint: use `verifyEqual`.
- Write code in the test_ManyBlank local test function to verify that repblank exhibits correct behaviour when the input string contains multiple consecutive blank values.
- Write code in the test_FirstBlank local test function to verify that repblank exhibits correct behaviour when the input string contains leading spaces.
- Write code in the test_LastBlank local test function to verify that repblank exhibits correct behaviour when the input string contains trailing spaces.
- Run all tests using the runtests command and ensure that all tests pass.
- Bonus: write another local test function to verify that repblank throws an error when called with a string consisting entirely of blanks. Hint: use verifyError with an appropriate error identifier. Ensure that all tests still pass as before.

Test and Verification of MATLAB Code.

Exercise - Levenshtein String Distances Solution - test_strdist_sol

- Open the script app_strdist and run the code. The objective of this exercise is to write unit tests for the strdist function. Open the strdist function and read the help to understand the function's specifications and requirements.
- Open the main test function test_strdist. This main test function currently has a setupOnce method, a teardownOnce method and three local test function stubs.
- Write code in the setupOnce method which adds the test data folder STRDIST_Test_Data to the path, and then initialises the TestData structure contained in the testCase object by loading the MAT-file words.mat.
- Write code in the teardownOnce method which removes the test data folder STRDIST_Test_Data from the path.
- Write code in the test_OneWord local test function to verify that strdist exhibits correct behaviour when called with only the first word from the words.mat MAT-file. (In this case, the expected answer is 0.)
- Write code in the test_ThreeWords local test function to verify that strdist exhibits correct behaviour when called with the first three words from the words.mat MAT-file. (In this case, the expected answer is the row vector [7, 5, 7].)
- Write code in the test_DimensionsOutput local test function to verify that strdist returns a solution of the correct dimensions when called with the first four words from the words.mat MAT-file. (In this case, the expected answer is the row vector [1, 6].)

Test and Verification of MATLAB Code.

Exercise - Counting Word Frequencies Solution - test_getwords_sol

- Open the script app_getwords and run the code. The objective of this exercise is to write unit tests for the getwords function. Open the getwords function and read the help to understand the function's specifications and requirements.
- Open the main test function test_getwords. This main test function currently has a setupOnce method, a teardownOnce method and three local test function stubs.
- Write code in the setupOnce method which adds the test data folder Literature to the path.
- Write code in the teardownOnce method which removes the test data folder Literature from the path.
- Write code in the test_IncorrectFilename local test function to verify that getwords exhibits correct behaviour when called with a non-existent file. Hint: use `verifyError`. In this case, the expected error ID is getwords:noFile.
- Write code in the test_AllWords local test function to verify that the first output of getwords has the correct size when the function is called on the file 'sherlock_holmes.txt'. In this case, the expected size of W is [207, 1]. Hint: use `verifySize`.
- Write code in the test_MaxFreq local test function to verify that getwords correctly returns the frequency of the most common word in 'sherlock_holmes.txt'. The most common word in 'sherlock_holmes.txt' occurs with frequency 10. Hint: use `verifyEqual`.

Debugging and Improving Performance.

Exercise - Debugging and Improving Existing Code Solution - analyzerEx_sol, analyzerEx_sol_vectorised

- In this exercise, you will debug an existing code file. Open the function analyzerEx in the Editor and inspect the Code Analyzer warning messages.
- Fix each of the Code Analyzer warnings, and call the function to ensure that the results are as expected.
- Bonus: Improve the code by vectorising the double for-loop. Hint: look up the repmat function, or use matrix multiplication or the cumsum function.