OBJECTIVES

The main purpose of this in-class exercise is to use modular design in developing your MATLAB codes. By the end of this exercise you should be able to develop scripts that call on user-defined and MATLAB functions to perform specific tasks.

NOTES

MATLAB COMMANDS

Throughout this document, the MATLAB commands that you must use in your scripts will appear in **`bold`** face.

COMMENTING

Comment your codes extensively. You can use the percent symbol `%` to enter comments in your scripts. Comments allow the user to understand and follow code easily; it is therefore highly recommended to develop a habit to extensively provide commentary in your codes.

A nice feature you may want to use in your scripts is code sectioning. Code sections allow you to organize, add comments, and execute portions of your code. Code sections begin with double percent signs (`%%`), e.g.,

```
%% Vector Operations
% You can perform a number of binary operations on vectors.
%%
A = 1:3;
B = 4:6;

%% Dot Product
% A dot product of two vectors yields a scalar.
% MATLAB has a simple command for dot products.
s = dot(A,B);

%% Cross Product
% A cross product of two vectors yields a third
% vector perpendicular to both original vectors.
% Again, MATLAB has a simple command for cross products.
v = cross(A,B);
```

CODE COPY-AND-PASTE

If you decide to copy and paste example command(s) presented in lecture slides and in-class exercises in MATLAB, be wary of single quotation marks—you may need to delete and re-enter single quotation marks after pasting the command(s) in MATLAB.

EXERCISES

EXERCISE 13: MODULAR DESIGN

The purpose of this exercise is to modify a script and simplify it by using modular design. You will develop two functions using snippets of code in the provided script called `matlabExercise13s.m` and replace the designed snippets with these user-defined functions.

Open the provided script `matlabExercise13s.m` and explore its content. Pay close attention to line 8 and lines 38–40. You need to write appropriate functions to replace these lines in the script. Specifically,

- Write a function called `matlabExercise13LoadData.m` that replaces line 8 in the provided script. You simply need to copy and paste the line from the provided script into a new m-file called `matlabExercise13LoadData.m` and convert the m-file into a function that has the same name as the m-file. This function should accept as input the filename and output the loaded data
- Write a function called `matlabExercise13FitPoly.m` that replaces lines 38–40 in the provided script. You simply need to copy and paste these lines from the provided script into a new m-file called `matlabExercise13FitPoly.m` and convert the m-file into a function that has the same name as the m-file. This function should accept as input `x`, `y`, `n`, and `xNew` parameters and output the predicted or estimated values `yHat`

DATA

Use the provided dataset `curveFitData.dat` for this exercise.

EXERCISE 14: CURVE FIT FUNCTION

The purpose of this exercise is to write a function to fit a line to data pairs that have been either derived from an exponential function of the form: $y = ae^{bx}$, or a power function of the form: $y = ax^b$. Specifically, create a function called `matlabExercise14CurveFit.m` that

- Accepts as input the `x` and `y` data pairs, the order (i.e., `n`) of the polynomial, and a variable (we will refer to this variable as a flag) indicating which type the data represents (there are only two data types, so this variable could be either 'e' for an exponential function or 'p' for a power function) and outputs the coefficients derived from fitting a linear line to the data
- Ensures that there are no negative values in the data, i.e., the data is filtered
- Fits a linear line to the data and computes the slope and intercept, i.e., first order polynomial coefficients
- Plots the data (y versus x using markers) and the fitted line (yHat versus x using solid line) on the same graph

In addition to above function, create a script and name it `matlabExercise14.m` to load appropriate data (see below) and extract independent and dependent variables before passing them, along with other arguments, to `matlabExercise14CurveFit.m`.

DATA

Use the provided datasets `expData.dat` and `powData.dat` to test your function. In both files, the first row contains `x` values and the second row contains `y` values.

EXERCISE 15: BASIC STATISTICS

The purpose of this exercise is to write a script that calls a function to compute basic statistical measures. Specifically, create a script and name it `matlabExercise15.m`. In your script,

- Load the provided dataset `grades.dat`
- Pass the data as input argument to a function (see below)
- Display the results outputted by the function to screen

The user-defined function called inside your script should:

- Compute and output the following statistical measures: **mean, median, mode, variance, standard deviation, min, max,** and **range**. *The number of arguments returned from this function is 8; think how you can simplify this and only return a single variable from the function that contains all of the statistical measures requested*

EXTRA EXERCISES

Additional exercises are included for extra practice purposes. I encourage everyone to work on them, but first complete the regular exercises—the ones without an X next to their number—then work on the extra exercises.

EXERCISE 10X

The purpose of this exercise is to write a script that calls a function to compute basic statistical measures. Specifically, create a script and name it `matlabExercise15.m`. In your script,

- Load the provided dataset `grades.dat`
- Pass the data as input argument to a function (see below for details)
- Display the results outputted by the function to screen

The user-defined function called inside your script should:

- Compute and output the following statistical measures: **mean, median, mode, variance, standard deviation, min, max,** and **range**
- Use **`varargout`** instead of listing all output arguments or using a structure array
- Return **desired output arguments** that user has specified in the call to the function. *Hint: You need to devise a simple conditional statement that builds the* ***`varargout`*** *variable, i.e., the variable length output argument list. You also need to use MATLAB's* ***`nargout`****, i.e., number of function output arguments.*