OBJECTIVES

The main purpose of this in-class exercise is to write simple MATLAB scripts to compute basic statistical measures (e.g., mean, median, variance, etc.) and generate Histogram plots.

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.

EXERCISE 4: BASIC STATISTICS

PART A

Create a script and name it `matlabExercise4a.m`. In your script,

- Load (**`load`**) the provided data file (`grades.dat`). The data contains a list of test scores for 25 students
- Compute the mean (**`mean`**), median (**`median`**), mode (**`mode`**), standard deviation (**`std`**), variance (**`var`**), minimum (**`min`**), maximum (**`max`**), and range (**`range`**) of the dataset
- Display each measure to the screen (**`disp`**)

PART B

Create a script and name it `matlabExercise4b.m`. In your script, compute the mean, maximum, and minimum values of the dataset **without** using the **`mean`** (or the **`sum`**), **`max`**, and **`min`** commands. Specifically, for each measure, figure out how to construct simple **`for`** loops to compute the mean, maximum, and minimum values. Compare your results with values obtained in PART A.

For example, to add the elements of an array, you can construct a loop that looks like:

```
a=[1 2 3];
sumData=0;%initialize variable
for ii=1:length(a)
    sumData=sumData+a(ii);
end
```

To find the smallest element in an array, you can construct a loop that looks like:

```
b=[2 1 3];
minVal=b(1);%initialize variable
for ii=2:length(b)
    if b(ii) < minVal
        minVal=b(ii);
    end
end
```

PART C

Create a script and name it `matlabExercise4c.m`. In your script, generate a histogram plot of the data. Specifically,

- Use MATLAB's **`histcounts`** command to partition the values into bins, and get the count in each bin as well as the bin edges
- Generate a histogram plot of the data using MATLAB's **`histogram`** command. Use the bin edges determined by **`histcounts`** to generate the plot
- Provide appropriate figure title, x-axis label (e.g., Bins), and y-axis label (e.g., Count)

PART D

Create a script and name it `matlabExercise4d.m`. In your script, generate a histogram plot of the data. Specifically,

- Compute range of data (**`range`**)
- Define the number of intervals or bins for the histogram. For example, define a variable named `nBins` and set it to be equal to `10`
- Compute left end-points of each interval or bin (**Note: this part of the code is the most challenging part; hence, a little contemplation is recommended**). To do this, first set the lower bound value for the first interval (*hint: the lower bound value for the first interval is the same as the minimum value of the data*); then compute the increment (i.e., `delta`) that you would like to add to this value in order to find the next end-point (*hint: the increment is the range of data divided by the number of bins, i.e., `nBins`*); finally construct a simple **`for`** loop to compute the rest of the interval end-points (*hint: the loop variable should start at `2` (why? because you have already determined that the left end-point for the first interval is the minimum value of the data) and go until `nBins`; furthermore, each new end-point should equal to the previous interval's end-point plus the increment)*
- Count number of data values in each bin; use MATLAB's **`histogram`** command. Specifically,

```
N=histogram(data,10);
count=N.Values;%count
```

  Note that when calling **`N=histogram(data,10)`**, the command will automatically generate a histogram plot. We will use this as a reference figure to compare it to the figure that will be generated using MATLAB's **`bar`** command (see below)

- In a new figure window, generate another histogram plot using MATLAB's **`bar`** command. Specifically,

```
%using MATLAB's bar command to generate a histogram plot
bar(x,count,'hist');
```

  where `x` represents the vector that contains the left end-points of each interval computed above

- Provide appropriate figure title, x-axis label (e.g., Bins), and y-axis label (e.g., Count)
- Compare the two histogram plots (the one generated using the **`histogram`** command with the one using the **`bar`** command). Can you spot the difference between the two?

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 4X: BASIC STATISTICS

Perform the same statistical measures described in EXERCISE 4, PART A but instead of `grades.dat` use `grades.mat` data file. Note that this data file is a structure array that contains multiple grade

types (Exams, Quizzes, and Homework). In order to compute the basic statistical measures for each grade type, you need to construct a **`for`** loop to extract scores for a specific grade type, compute the measures, then display the results to screen. Hint. To access each grade type, use: `grades(i)`, where `i=1,2,3,...`, and to access the scores per grade type, use:
`grades(i).scores`