1. **Setup: Import File, Setup Vectors and Examine Time Vector**
This Lab focuses on learning some techniques that could be helpful for your sensor project in studio and on using these techniques to practice and extend your MATLAB skills.

Data File for Exercises: Before starting this section copy an acceleration data set into MATLAB’s current working directory and try these techniques as you go. It will be most helpful if the sensor data used is from your own studio project. If you do not have some sensor data use a previous file or use the provided csv file (on website). It is appropriate to consult with your studio project team for this section. However, everyone must complete the MATLAB exercises and problems themselves.

Background: This lab will draw on topics covered earlier in the term (e.g., Importing text & csv files, Matrix & Vector Addressing, using built-in functions and graphing). If you have problems with any of these topics now is the time to address them. Ask if you have questions. The main text that follows explains how to do things, the gray boxes indicate items that need to go in a script documenting your work. Please use numbered and bolded titles in these boxes as comments to organize your script.

1. Setup: Import Example Data File: Import the data using the *importdata* function. Remember that for a file with text and data *importdata* will create a structure with a structure name and several field names. The data will be in the field name “.data”. Ask as needed, make sure you know how to this.

& Create vectors: Use array addressing to select the time, x, y and z components of sensor data and save them to separate vectors for easy use (ask if you do not remember, also look at Text Variables & Files lab on website – particularly section IV ).

* Start a script: **1) Import Accelerometer Data & convert to t, x, y, & z vectors**
Include commands, comments and results
	+ Import Cell Phone sensor data file
	+ assign the time, x, y and z components to separate vectors
1. Examination of the Time Vector & Sampling Frequency

The nature of the captured time vector will be very important to how well your project works out. This vector needs to be examined and evaluated. In the MATLAB command window:

1. display the first 21 numbers in the time vector (save to show instructor later).
2. Next create and display a vector to show the first 20 intervals between these numbers. (This will require two copies of this portion of the time vector offset by one index. This is similar to what was done in the addressing exercise. Ask for help if stuck). If you get here before the addressing exercise, give it some thought & if you are not sure how to approach, skip for now.

 It is important to notice if the time spacing is constant or if it varies, how much does it vary. The actual performance will depend on your device, the app used and the sampling settings.

Average Sampling Frequency: It is helpful to know the average sampling frequency for the data. The average sampling frequency can most easily be calculated by dividing the *length* of your time vector (using MATLAB’s ***Length*** function) by the last entry in the time vector. Calculate this value in MATLAB.

Evaluate the pattern: Are the values evenly spaced? If not, how are they spaced?

For either case what is the average sampling frequency?

For Script: **2) Time Vector spacing and average sampling frequency**

* + - * display of the first 21 time values,
			* calculate all the time intervals
			* display of the first 20 intervals,
			* calculate and display the average sampling frequency
1. **Vector Tools: Continue development of Sensor Data Practice Script**

Overview: In this section exercises are added to the script started in the last section. These exercises demonstrate various techniques to try on your test series. Use one of the three component vectors for these exercises (i.e., the x-vector or the y-vector or the z-vector). Your script should already include two sections.

In this part the following additional script sections are developed (detailed instructions follow):

1. Analysis of segments of the vector using the reshape function and descriptive statistics (i.e., calculate the mean, standard deviation, minimum, and/or maximum).
2. Using the find function to identify times that are within 10% of the maximum acceleration.
3. Using vector calculations to determine gtotal from the x, y, and z sensor components. Consider other vector calculations appropriate to your particular projects.
4. Frequency Analysis using a provided function, analyze1.m, to determine the frequency components in a sample.

The subsections below describe details for how to conduct these types of analysis in general and how to complete the specific tasks for the script. Include comments in the script clearly identifying the above six sections.

1. Assignment of Segments and Descriptive Statistics

Descriptive statistical commands may be helpful in analyzing a time series segment. Most of these require a consistent logical segment of sensor data to work with. Consider whether any of the following might be useful in summarizing your series.

mean(x) returns the mean of the series x

std(x) returns the sample standard deviation of a vector x

max(x), min(x) returns the maximum value and minimum value respectively of x.

 can also return the location of the maximum (see help or doc)

for all of these, if x is a matrix they return a row vector of the calculated value for each column. (do try these out)

Analysis of multiple vector segments

The reshape function is helpful for looking at sub-sections of the entire vector. To get familiar with this command try the following at the MABLAB command prompt and observe what happens.

>> eg = 1:10

>> x1 = reshape(eg, 2, 5)

>> x2 = reshape(eg, 5, [ ] )

Notice how the reshape command turns the vector into a 2-d matrix with each column an equal length sub-section of the original vector. The descriptive statistics can then be evaluated using the earlier commands.

For example, try: >> mean(x1)

However, the reshape command only works if the vector can be evenly divided into a series of columns (e.g., try: >> reshape(eg, 3, []). This can be fixed by shortening the length of the vector for even divisibility. The remainder function and vector addressing can be used to carry this out automatically. For example try the following:

>> a = rem(length(eg), 3)

>> x3 = reshape( eg(1:end-a), 3, [ ] )

Can you explain how the above works?

For the project it may be helpful to divide the vector into columns that represent a planned time (e.g, ~ one second of data in each column). Type “help reshape” to see help file for this new command.

For Script: **3) Vector Subsections & Descriptive Statistics**
Add code to reshape one of the sensor vectors to a series of columns and calculate the mean of those columns. Include use of the rem command and vector addressing to insure that the number of elements in the vector can be reshaped. Ideally each column isa planned length of time (e.g. ½ sec.).

1. *MATLAB’s find function*

The find function in MATLAB can be very handy for finding interesting parts of a vector (or matrix). It can provide the index location of items that match a given condition. The general setup is:

find(logical statement) This function returns all index locations that are true for the logical statement provided. E.G.: find(x>0.5) will return the index number for each case in x that is greater than 0.5. Type “*doc find”* at the command line for further information.

*Finding Exercises*
Try the following at the command line in order to become familiar with using the find function:

* Set the following two vectors into the workspace:
 >> a = [-3, 0, 0, 2, 5, 8] and >> b = [-5, -2, 0, 3, 4, 10].
* Try the relational comparison: >> test = b < a
* Try the following variations on the find command
 >> find( b < a ) Find returns the index location of the true cases

>> find( test ) same as previous

>> a(b < a ) This returns the a values for the true cases

>> b(b < a ) This returns the b values for the true cases

* Now try using *find* on your example sensor data.
	+ Store the time data in the variable t
	+ Store the x values in the variable x
	+ Use *find* to pull out the x values for times between 1.5 and 2 seconds using the following commands:

>> t( t >= 1.5 & t <= 2 )

>> x( t >= 1.5 & t <= 2 )

**Show instructor:** Show your instructor or assistant the results of these tests. Do ask questions.

For the script: **4) the Find Function**
Write code that will identify times that are greater than 90% of the maximum acceleration of the sensor segment you are examining (i.e. find points where x > 0.9\*max(x)). Include this code in the script. The script should display the time and value of the points within 10% of the maximum.

1. Vector Calculations
Using vector calculations covered earlier in the term will be very useful in analyzing the sensor data. This particularly includes the element-by-element (i.e. dot operator) calculations. As needed, review earlier labs and/or ask your instructor for help with using these calculations on your particular project problem.

In analyzing the sensor data it is important to consider the directions that might be important to your analysis. The standard data set includes the three dimensions and the time vector. At any given time the x, y and z sensor data recorded from a sensor app are simply components of a three dimensional vector. Equation 1, calculates the total magnitude for a given time (i.e., its Pythagorean length).

 (1)

 **where:** gtotal = the total sensor value (sensor units)
 gx, gy, gz = the x, y, z-direction components of the sensor value (same units)

For script: **5) Vector Calculations**
 Create a vector of total reading for each time in your sample using the three component vectors. This should be done using vector calculations. Display the first 10 values in this vector. This completes the practice script (see assignment requirements on page 6).

1. Frequency analysis

Download, to your current working directory, the function analyze1.m from the course website. This function can be used to identify dominate frequency components in a time series.

Inputs: The function requires two inputs: 1) a vector of amplitudes evenly spaced in time and 2) the sampling frequency for that vector. Samples from your sensor app may not be evenly spaced (having the same time between each sample) but are likely close. Use one of your sensor components and its average sampling frequency found earlier.

Output: This function results in a “Frequency Power Spectrum” graph. The x-axis of this graph is frequency and the y-axis is the relative strength (the “power”) of each frequency. If this graph has distinct peaks that are tall relative to the rest of the frequencies, the x-coordinate of a peak will be a dominant repeated frequency in the sample analyzed. Do semicolon this function as it does output the vector of frequency powers.

For this function to work well the sampling rate needs to be high enough that more than two points are sampled per cycle (preferably much more than 2). It will likely work best if the second highest sampling rate on the device is used (e.g., the “Game” rate for the Android Physics Toolbox suite). The highest rate will tend to have a less uniform time spacing. It is also important that several cycles are captured and that a constant periodic function is available for the entire sample (see your instructor for more details). Generally a vector of at least 1024 samples is needed (do ask about these factors).

For script: **6) Frequency Analysis (analyze1.m)**
Include code to run the analyze1 function on one of the sensor components in the script. Copy the resulting frequency power spectrum graph to a word document. Typical call: analyze1(y, Fs);

1. **Exploratory Data Analysis**

When dealing with new data the best initial analysis approach is usually to graph the data and to do it in many different ways. In statistics there is even a formal approach to this called “Exploratory Data Analysis” which primarily consists of systematic ways to explore data with graphs and other hints at possible relationships.

**Interactive Graph Properties** (new technique!):

Graphs in MATLAB can be edited interactively. This can be very helpful in exploring a longer sensor recording. Interactive properties are accessed from the figure window created by the plot command. Try the following steps with some of your group’s sensor data (or use provided csv file):

* Plot the one of the sensor data series vs. time. Then
* To start interactive plotting, select ***Axis Properties*** from the ***Edit***menu in the figure window. This should open a property editor dialog.
* Notice the created property editor includes the x variable limits (in the rulers section, XLim). These can be changed to zoom in on a specific area of the graph. Make x limits a narrower range and see what happens to the graph.
* In the labels section (push the arrow to open it up) notice XLabel box in the property dialogue. Click on the “Text” to open a sub-panel that allows you to change the label in a variety of ways. Experiment with these options
* There are similar options for the y-axis label.
* Double click on the plotted line to edit its format (line type, marker type, line thickness, color …). Change several properties of the graphs line.
* Click on other parts of the plot to see corresponding formatting options. Adjust several items using these dialogs. Have some fun with this.

Remember you can use the data curser to determine specific values. Hover over any point on the graph. Notice the information that is shown (the x and y coordinates. Move the mouse to see successive points. Click on a point to set a label of the coordinates on that point.

1. **Integration using a Right Riemann Sum** (example to help with assignment problem b)

One view of integration is it is simply the area under a curve. The simplest numerical approximation of the integral is series of rectangles approximating this area. For a Right Riemann Sum these are rectangles that are the width of the change in the x variable, times the height at the right of the interval. Equation 2 represents this approximation for a single rectangle between x = a and x = b.

 (2)

To get the total integral the area of all the bars must be added together.

In MATLAB for a y-vector (representing f(x) in Equation 2) and a t-vector, (the x in Equation 2) the code could be:

Note if the time interval is constant then delta can simply be the constant interval (i.e. delta = 1/sampling frequency). Try this on one of your component vectors.

 >> delta = t(2:end)-t(1:end-1);

 >> A = sum(y(2:end).\*delta)

Where A is the calculated estimate of the integral.

1. **Assignment** (due at the beginning of the next lab period)
	1. Sensor Data Practice Script as outlined above. To turn in: A copy of a fully commented script with the following components (use the part numbers below in the comments). These steps are outlined in the shaded boxes throughout this handout.
2. Setup: Import Accelerometer Data & convert to t, x, y, & z vectors
3. Time Vector: Time Vector spacing and average sampling frequency
4. Statistics: Vector Subsections & Descriptive Statistics
5. Find Function Analysis: Points within 10% of vector maximum
6. Vector calculations: total sensor magnitude and display of first 10 values
7. Frequency Analysis: A frequency power spectrum graph

The execution of the above script including both command window results and the graph figure window with the frequency analysis (script publishing may also be used).

* 1. Develop a function to integrate a set of vectors using the trapezoidal rule (or other more precise rule, e.g. Simpson’s Rule). Trapezoidal rule uses trapezoids instead of rectangles in order to approximate the area (see posted slides). Use your own vector calculation (i.e., do not use a built-in MATLAB integration function). The inputs are a time vector and a component vector.

To turn in: A completed program development worksheet including:

* Complete setup (i.e., the first page of the program development worksheet) with
* hand calculations for a made-up time vector and a made-up magnitude vector of at least four elements each
* the fully commented code, and
* validation tests on two cases: 1) your made-up vectors & 2) a vector from phone sensor data

Creating Subplots (a helpful aside): Creating several subplots in one MATLAB figure window is a handy way to look at the three sensor components on separate plots. The subplot function is used to create these multiple graph axes in one figure window. In its basic form it takes three inputs:

N = the number of rows of separate plot axes

M = the number of columns of separate plot axes

P = the number of the current plot being worked on.
Plots are numbered row wise from the top left corner

Function syntax: >> subplot(N, M, P)

After the subplot function is executed the normal plotting commands can be used to create the identified plot. The subplot is then used again generally with the same N and M but with the next plot number (i.e., the next P). Standard plot commands can then be used to create the next plot. For example try these commands:

 >> x = 1:20; y1 = x; y2 = sin(x/3); y3 = x.^2;

>> subplot(3, 1, 1)

>> plot(x, y1, ‘p’)

>> xlabel(‘x’), ylabel(‘y1’), title(‘just some arbitrary plot’)

>> subplot(3, 1, 2)

These executables will: 1) create some example data, 2) create a figure ready for a three rows x one column array of plots (i.e., three plots in a single column), 3) plot the y1 vs x (a linear plot) in the top position of the figure and label that plot, 3) with the second subplot function call, prepare for the second plot to be added to the figure in a similar fashion. Continue by plotting y2 vs x.