Design and Implementation of a Transduction System for the Measurement of a Weakly Nonlinear Mechanical Oscillator

Group 3: Paul Gennaro, Patrick Koczela, Byron Miller, & David Miller
ME310, Professor Farny

April 27, 2015

1 Introduction

A scientific theory is only good if it can be backed by experimental data. In this project, the theory behind a 2nd Order mechanical system will be compared to experimental data collected via digital data acquisition. This will be accomplished by following a transduction scheme created by our group. The results of this project showed that, within uncertainty, the scientific theory behind 2nd order mechanical systems applies to the mechanical oscillator under consideration.

2 Theory

2.1 2nd Order System Theory

Few concepts in engineering provide the mathematical elegance and complexity that a 2nd Order System does. The concept of a 2nd Order System, which is a system that is modeled by a first and second derivative of the system's characterizing variable, was originally developed in the 18th Century alongside calculus and classical mechanics. The model is now used to describe countless number of phenomena: double pendulums, RLC Circuits, and economic cycles, just to name a few. For this project, the model of a 2nd Order System will be applied to two masses coupled by a spring with damping.

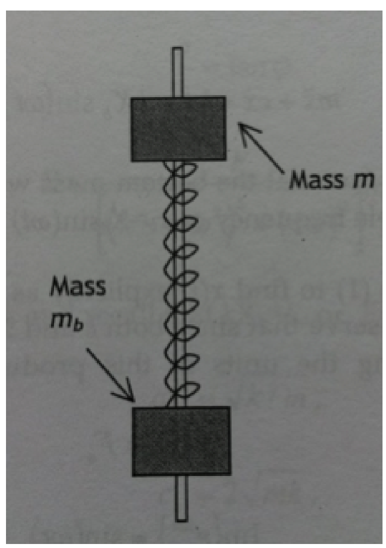


Figure 1: Two masses coupled by a spring. [3]

In this system, a known displacement is applied to the bottom mass mb which is attached to a spring with a mass m on top. The goal of the model is to know the displacement of the top mass as a function of time: $x(t)$. The spring has both a spring constant k and a damping constant c. The forces on the top mass caused by the spring and the damping are linear:

$$F_s = -k\Delta x \tag{1}$$
$$F_d = -c\dot{x} \tag{2}$$

The spring force will always act opposite of the displacement of the top mass and the damping force will always act opposite the velocity of the top mass. By applying Newton's Second Law, which states that the sum of the forces on an object is equal to its mass times its acceleration, to the vertical direction (which is the x-axis) of the spring system, we get:

$$\sum F = m\ddot{x} = -c\dot{x} - k\Delta x \tag{3}$$

By noting that the change in spring length is the difference between the top mass as the bottom mass ($\Delta x = x - x_b$), Equation 3 can be rearranged into the form of a 2nd Order Differential Equation:

$$m\ddot{x} + c\dot{x} + kx = kx_b \tag{4}$$

For this lab, x_b is a known input displacement on the bottom mass. The mass is attached to a motor which moves sinusoidally with some variable frequency ω and some displacement amplitude Xb. Thus the bottom displacement is $x_b = X_b \sin(\omega t)$, so the entire system can be described:

$$m\ddot{x} + c\dot{x} + kx = kX_b \sin(\omega t) \tag{5}$$

Now that the system is described as a 2nd Order Differential Equation, x(t) can be solved for. In order to solve for $x(t)$, we rewrite Equation 5 using complex exponentials, noting that $Im\{e^{i\omega t}\} = \sin(\omega t)$:

$$m\ddot{x} + c\dot{x} + kx = kX_b e^{i\omega t} \tag{6}$$

In using Equation 6 to solve for $x(t)$, we only use the imaginary part of the solution because it is only the imaginary input that we care about. Let us assume linearity so we have a pure harmonic time dependence:

$$x(t) = Xe^{i\omega t} \tag{7}$$

By taking the first and second time derivatives of Equation 7 and plugging them into Equation 8 we get:

$$\dot{x}(t) = i\omega X e^{i\omega t} \tag{8}$$

$$\ddot{x}(t) = -i^2\omega^2 X e^{i\omega t} = -\omega^2 X e^{i\omega t} \tag{9}$$

$$(7, 8, 9) \rightarrow (6)\colon\ (-m\omega^2 + i\omega c + k) X e^{i\omega t} = k X e^{i\omega t} \tag{10}$$

Rearranging Equation 10:

$$X = \frac{kX_b}{k - m\omega^2 + ic\omega} \tag{11}$$

Multiplying the top and bottom by the complex conjugate we get:

$$X = \frac{k - m\omega^2}{(k - m\omega^2)^2 + (c\omega)^2} + i\frac{c\omega}{(k - m\omega^2)^2 + (c\omega)^2} \tag{12}$$

Noting the properties of complex numbers, Equation 12 fits the type of expression $a + ib = Ae^{i\omega}$, where $A = \sqrt{a^2 + b^2}$ and $\Phi = \arctan(a/b)$, so:

$$X = \frac{kX_b}{\sqrt{(k - m\omega^2)^2 + (c\omega)^2}} e_{i\Phi} \tag{13}$$

If we divide the top and bottom by k and defining the natural frequency ω_n and damping ratio ζ:

$$\omega_n = \sqrt{\frac{k}{m}} \tag{14}$$

$$\zeta = \frac{c}{2\sqrt{mk}} \tag{15}$$

We can write:

$$X = \frac{X_b}{\left[\left(1 - \left(\frac{\omega}{\omega_n}\right)^2\right)^2 + \left(2\zeta\frac{\omega}{\omega_n}\right)^2\right]^{1/2}} e^{i\Phi} \tag{16}$$

For convenience, we can write the magnitude ratio which is the ratio of input and output magnitudes:

$$\frac{X}{X_b} = M = \frac{1}{\left[\left(1-\left(\frac{\omega}{\omega_n}\right)^2\right)^2 + \left(2\zeta\frac{\omega}{\omega_n}\right)^2\right]^{1/2}} e^{i\Phi} \tag{17}$$

From the definition of Φ, we can write:

$$\Phi = \arctan\left(\frac{a}{b}\right) = \arctan\left(\frac{2\zeta\frac{\omega}{\omega_n}}{1-\left(\frac{\omega}{\omega_n}\right)^2}\right) \tag{18}$$

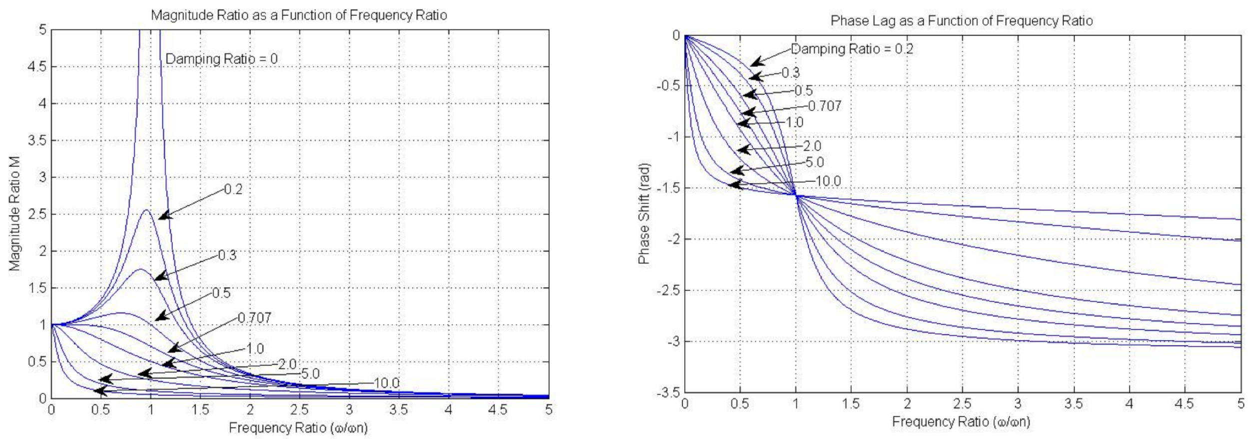
Now we have all the information required to write a complete steady-state solution from our original assumptions:

$$x(t) = Im\{Xe^{i\omega t}\} = Im\{MX_b e^{i\omega t+\Phi}\} = MX_b \sin(\omega t + \Phi) \tag{19}$$

In summary, the 2nd Order System in the project can be modeled with the following equations from the derivation above:

$$\begin{gathered}
m\ddot{x} + c\dot{x} + kx = kX_b \sin(\omega t) \\
x(t) = MX_b \sin(\omega t + \Phi) \\
M = \frac{1}{\left[\left(1-\left(\frac{\omega}{\omega_n}\right)^2\right)^2 + \left(2\zeta\frac{\omega}{\omega_n}\right)^2\right]^{1/2}} \\
\Phi = \arctan\left(\frac{2\zeta\frac{\omega}{\omega_n}}{1-\left(\frac{\omega}{\omega_n}\right)^2}\right) \\
\omega_n = \sqrt{\frac{k}{m}} \\
\zeta = \frac{c}{2\sqrt{mk}}
\end{gathered}$$

Upon observation, the system response is dependent on two variables, the magnitude ratio M and the phase lag Φ. Those two variables are not constant for all inputs or all second order systems, but rather they are dependent on two system characteristics and one input characteristic: the natural frequency ω_n the damping ratio ζ, and the input frequency ω. For engineers, it is helpful to look at a system's response as a function of input frequency and damping ratio.



(a) Magnitude Ratio

(b) Phase Shift

Figure 2: Magnitude ratio and phase shift vs. normalized frequency; multiple damping ratios plotted.

2.2 Transducer Theory

In order to measure the behavior of the spring-mass system, a transducer needed to be attached to the top mass to measure its displacement. For this project, a capacitive micromachined accelerometer was used. If the acceleration of the top mass is known as a function of time, then we can integrate the acceleration twice to determine position. (This process will be discussed more thoroughly in the Analysis and Results section.)

There are several ways to design a digital accelerometer, but all serve the same general purpose: output a change in voltage given a change in acceleration. The accelerometer used in this project is a capacitive micromachined accelerometer designed by Silicon Designs Inc., Model 2210. A capacitive micromachined accelerometer works by measuring the voltage across a small capacitor. By attaching one side of the capacitor to a base and leaving the other side free to move, the mass that is free to move will change position with acceleration. The particular model being used in the project is nitrogen damped, meaning the capacitor is incased in a housing filled with nitrogen gas, causing the damping. By changing position, the capacitance and thus the voltage will change. Through signal conditioning, the accelerometer outputs a voltage for a given acceleration.

A capacitive micromachined accelerometer generally has a high sensitivity compared to other accelerometers, such as piezoelectric accelerometers. Multiple capacitors can be placed in a sensor to measure acceleration in more than one axis. The particular accelerometer used in the project designed by Silicon Designs Inc. is a single axis accelerometer, so the acceleration is only measured in one direction.

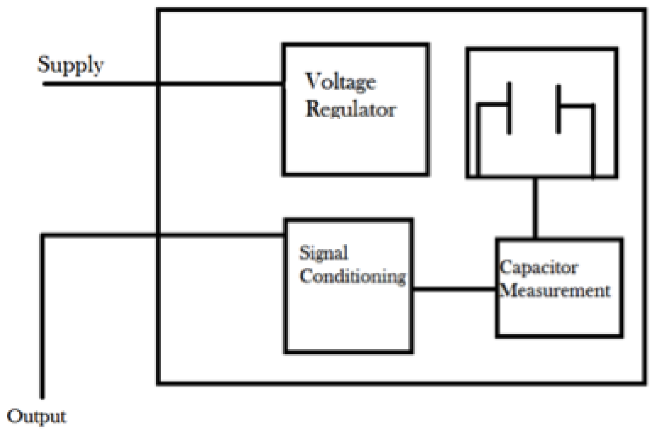


Figure 3: Block diagram of the accelerometer used in the project.[2]

Figure 4: Picture of Silicon Designs Inc., Model 2210 Capacitive Micromachined Accelerometer.

3 Methods and Materials

Prior to accelerometer data collection and analysis of the oscillator, several parameters needed to be determined. The experimental methods for determining those parameters are described in this section.

In order to determine the spring constant k, a calibration was performed. A round metal plate was fitted to the collar [upper mass] and secured with screws. To measure the spring constant, masses were placed upon the metal plate in 100g [gram] increments and the deflection was measured until 1000 grams was reached. After reaching 1000g, we repeated the process, decrementing by 100g and measuring the displacement until reaching 0g. This allowed hysteresis error to be observed and accounted for in the uncertainty analysis.

Next, we calibrated the accelerometer. We had three control points for the calibration: -1g [acceleration], 0g, and +1g. To do this, we placed the accelerometer upside down for -1g, on its side for 0g, and right side up for +1g.

The final experimental parameter was to measure the total linear displacement of the lower mass. To do this, a ruler was used to measure the difference between the bottom surface of the lower mass in its lowest position, and the bottom surface of the lower mass at it highest position.

Once these parameters were determined, accelerometer data could be collected and subsequently analyzed. Accelerometer data were collected using a DAQ board and a MATLAB script to save the data. Data were collected for several several motor frequencies. To observe hysteresis, a ramp-up/ramp-down procedure was used, similar to the spring constant procedure.

The analysis was performed with the aid of a MATLAB Graphical User Interface (GUI). The GUI allows us to easily visualize results for particular datasets. This ability to visualize the data and results allowed us to quickly determine the quality of our data, if there were any problems in the analysis, and most importantly, if the results were acceptable or if more data needed to be taken. Each step in the block diagram in Figure 5 was its own function nested in the GUI that was independently verified with user generated fake data.

Once data had been loaded into the GUI, the "Analyze" button executed all of the functions as seen in the block diagram. For analyzing multiple datasets, the "Analyze All" button is pressed. This brings up a subgui (Figure 6b) that allows the user to analyze multiple datasets in a for loop. This requires that all the datasets have the same 'root' filename, and only vary by a substring in the filename containing a number. To analyze the data, selected datasets were visualized using the "Analyze" button. Once it was confirmed that the results for the selected datasets were acceptable, all the datasets could be analyzed at once using the "Analyze All" button.

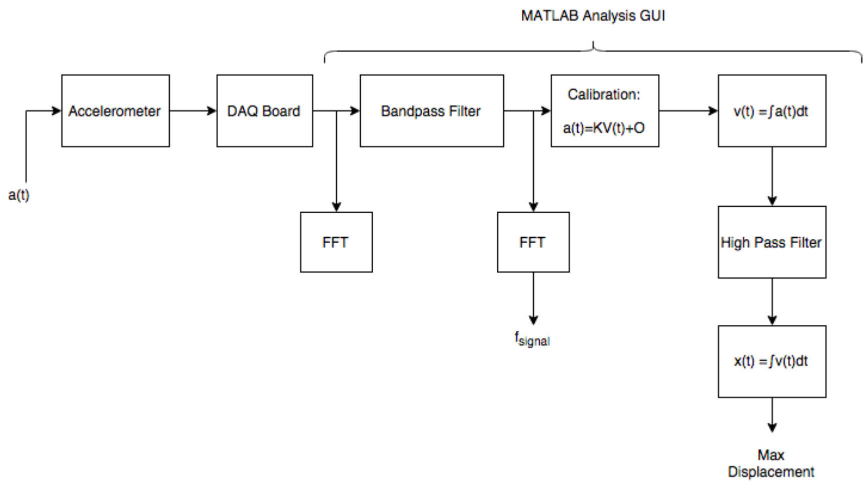
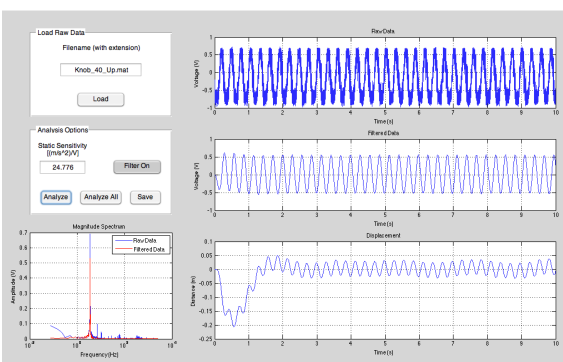
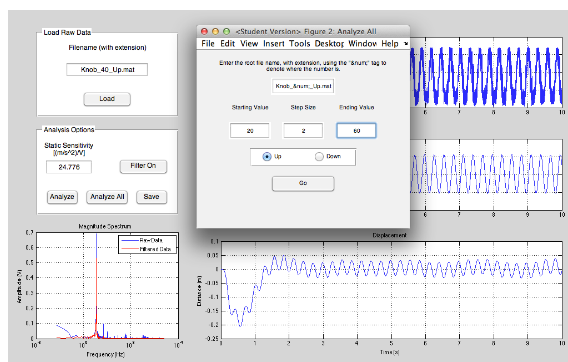


Figure 5: Block diagram of experimental setup.





(a) The main GUI window allowed for analyzing and visualizing individual datasets.

(b) Multiple datasets could also be analyzed in this subgui.

Figure 6: Screenshots of the GUI being used for data analysis.

4 Equipment List

Device	Manufacturer	Model Number	Serial Number
Analog Accelerometer Module	Silicon Designs Inc	2210-010	15489
Triple Output DC Power Supply	Agilent	E3631A	(Rack 3) MY50190084
DAQ Board	National Instruments	183468A-01	(Rack 3) A5199B
ME310 Oscillator	Boston University	N/A	#3
Ruler	–	–	–
Triple Beam Balance	OHaus	–	–

Table 1: Equipment List

5 Data

m [kg]	x [cm]
0.2762	34.7863
0.3762	32.4862
0.4762	30.8862
0.5762	27.9862
0.6762	26.8862
0.7762	25.8862
0.8762	23.2862
0.9762	22.6863
1.0762	20.3863
1.1762	17.6863
1.2762	17.3863
1.1762	17.0862
1.0762	17.7862
0.9762	18.0862
0.8762	19.8863
0.7762	21.3863
0.6762	26.6863
0.5762	27.4862
0.4762	29.1863
0.3762	30.7863
0.2762	33.3862

Table 2: Spring Constant Calibration Data. Mass m is the combined mass of the collar, plate, screws, and test mass. Distance x is the length of the spring.

6 Analysis

The analysis of data to produce results can be broken into three sections. The goal of the first section of the analysis is to determine parameters of the instrument and of the oscillator. The second section of the analysis involves using signal conditioning steps of the analysis applies the raw data to generate both the systems spring constant and the position of the top mass for different motor speeds. The third part of the analysis uses the calculated position data and calibrated spring constant to determine the other parameters of the system: damping ratio, natural frequency, and resonant frequency.

6.1 Calibrations

Spring Constant Calibration

This analysis applies to the data contained in Table 2. In order to determine the spring constant k of the system, a calibration was performed. From Hooke's Law it is shown that adding mass atop the spring increases the force on the spring, yielding the following relationship.

$$F = mg \tag{20}$$

$$(20) \rightarrow (1):\ k = -\frac{mg}{\Delta x} \tag{21}$$

In principle, Δx refers to the difference between the observed length of the spring and the relaxed length of the spring. However in the experimental setup, the true relaxed length of the spring was unknown. Thus, a surrogate reference length x_{ref} was defined as the length of the spring with the collar attached and the plate attached. The corresponding mass m_{ref} is then the mass of the collar, the plate, and the screws. Thus,

$$\Delta x_i = x - x_{ref} \tag{22}$$

$$m_i = m - m_{ref} \tag{23}$$

Where the subscript i refers to the control masses placed onto the plate. A linear regression can be performed using the control masses m_i and the measured displacements Δx_i to determine a spring constant k_{fit}, which is simply the inverse of the regression coefficient a_1.

Accelerometer Calibration

The accelerometer is a transducer which converts acceleration inputs into voltage outputs. There is an approximately linear functional relationship between the acceleration and the voltage, thus a linear calibration can be used. Three control accelerations were used in the calibration; +1g, 0g, & -1g. Voltage data for each control point were acquired through the DAQ board into MATLAB. The static sensitivity K was calculated using a linear regression.

6.2 Signal Conditioning

Position Calculation

The input acceleration takes the form of a sinusoid with amplitude A and phase shift Φ offset by gravitational acceleration g:

$$a(t) = A\sin(\omega t + \Phi_0) + g \tag{24}$$

Accelerometer response given by 2nd order system:

$$V(t) = K'AM(\omega)\sin(\omega t + \Phi_0 + \Phi(\omega)) + K'g + O' \tag{25}$$

Because the driving frequency ω is much lower than the resonance frequency of the accelerometer ω_{r_a}, we can assume:

$$M(\omega)|_{\omega<<\omega_{r_a}} = 1 \tag{26}$$

$$\Phi(\omega)|_{\omega<<\omega_{r_a}} = 0 \tag{27}$$

K' and O' are known from the instrument calibration, thus:

$$a(t) = \frac{V(t) - O'}{K'} \tag{28}$$

A new K and O can be redefined such that

$$a(t) = KV(t) + O \tag{29}$$

Now, given the acceleration of the top mass, basic kinematics can be applied to determine position. Acceleration is the time rate of change of velocity, and velocity is the time rate of change of position, so position can be calculated through the double integration of acceleration.

$$x(t) = \int v(t)dt = \iint a(t)dt \tag{30}$$

$$x(t) = \iint (KV(t) + O)dt \tag{31}$$

To perform the integral on the raw acceleration data, double cumulative numerical integration in MATLAB was performed. Because the integration is cumulative, integrating a sinusoid centered at zero produces a sinusoid centered at the amplitude of the sinusoid. In order to integrate again, offset created by the first integration must be removed. To accomplish this in MATLAB, the results from the first numerical integration were passed through a high pass filter to remove the offset. Then, the data were integrated once more to yield the displacement. Because the filter removes any offsets, the instrument offset O can be omitted from the equation.

$$x(t) = \iint KV(t)dt \tag{32}$$

6.3 Determining System Parameters

With all the raw data analyzed and converted into a position function, the other system parameters can be determined. Frequency and amplitude of the system response can be determined from the position graph. By performing a Fourier frequency transform (FFT) on the position function, the frequency that the system was operating at could be determined by calculating which frequency was the largest contributing frequency from the FFT. Similarly, the maximum and minimum displacements can be determined from the graph to determine amplitude.

$$A_{upper\ mass} = \frac{\max[x(t)] - \min[x(t)]}{2} \tag{33}$$

By using the above calculated amplitude and the known input amplitude for the bottom mass, the magnitude plot can be made for each frequency. With the amplitude known for each frequency, the rest of the parameters can be determined. The frequency at which the maximum amplitude occurs indicates the resonant frequency ω_r. With the resonant frequency, spring constant k and the mass m known, the rest of the system parameters can be calculated:

$$\omega_n = \sqrt{\frac{k}{m}} \tag{34}$$

$$\omega_r = \omega_n \sqrt{1 - 2\zeta^2} \tag{35}$$

$$\zeta = \sqrt{\frac{1}{2}\left(1 - \left(\frac{\omega_r}{\omega_n}\right)^2\right)} \tag{36}$$

$$(28) \rightarrow (30):\ \zeta = \sqrt{\frac{1}{2}\left(1 - \left(\frac{\omega_r}{\sqrt{\frac{k}{m}}}\right)^2\right)} \tag{37}$$

$$\tag{38}$$

The damping constant c is:

$$c = 2\zeta\sqrt{km} \tag{39}$$

The quality factor Q is:

$$Q = \frac{1}{2\zeta\sqrt{1-\zeta^2}} \tag{40}$$

7 Uncertainty Analysis

7.1 Calculations

The uncertainties in each final parameter stem from three individual sources: uncertainty in the spring constant (U_k), uncertainty in determining the resonant frequency (U_{ω_r}), and the uncertainty in the voltage signal (U_V). All other uncertainties are functions of these three, and can be calculated via error propagation steps.

Spring Constant Uncertainty

Uncertainty in the spring constant is a function of the precision uncertainty of the spring constant values and the bias uncertainty of the displacement values.

$$U_k = \mathrm{f}(U_{p,k}, U_{b,x}) \tag{41}$$

There precision uncertainty $U_{p,k}$ is calculated as a fit uncertainty, where each data point has its own value:

$$k_i = \frac{m_i g}{\Delta x_i} \tag{42}$$

The k value resulting from the linear regression is treated as the k_{fit} in the uncertainty calculation, which yields:

$$U_{p,k} = \pm t_{\nu_{fit}} \frac{\sqrt{\sum_{i=1}^{N}(k_i^2 - k_{fit}^2)}}{\nu_{fit}} \tag{43}$$

The bias uncertainties comprising $U_{b,x}$ are hysteresis and resolution of the ruler used. The total uncertainty. An error propagation was used to combine the precision uncertainty in the units of k with the bias uncertainty in the units of x into a total uncertainty in units of k. The average of the values of the partial derivative term was used in the calculation.

$$U_k = \pm\sqrt{U_{p,k}^2 + \left(\frac{F}{\Delta x^2}U_{b,x}\right)^2} \tag{44}$$

This uncertainty method was also used to determine the uncertainty in the instrument sensitivity K.

Resonant Frequency Uncertainty

Uncertainty in the resonant frequency is primarily due to resolution of the input frequencies.

$$U_{\omega_r} = \pm U_f \tag{45}$$

$$\tag{46}$$

Voltage Signal Uncertainty

Uncertainty in the voltage signal from the DAQ board is due to bias uncertainty taken from the spec sheets for the power supply, the accelerometer, and the DAQ board.

$$U_V = \pm\sqrt{U_{p,V}^2 + U_{b,V}^2} \tag{47}$$

$$\tag{48}$$

Other Uncertainties

The remainder of the uncertainties can be found by propagating different combinations of these three uncertainties. The derivations for each uncertainty are lengthy and repetitive and will not be shown. The final expressions are shown below.

$$U_K = \pm K U_V t^2 \tag{49}$$

$$U_{\omega_n} = \pm\sqrt{\frac{1}{km}} U_k \tag{50}$$

$$U_\zeta = \pm\sqrt{\left(\frac{-\frac{\sqrt{2}\omega_n}{\omega_r}}{2\sqrt{\omega_n^2 - \omega_r^2}} U_{\omega_r}\right)^2 + \left(\left[\frac{\sqrt{2}}{2\sqrt{\omega_n^2 - \omega_r^2}} + \frac{\sqrt{2(\omega_n^2 - \omega_r^2)}}{2\omega_n^2}\right] U_{\omega_n}\right)^2} \tag{51}$$

$$U_Q = \pm\left(\frac{1}{2(1-\zeta^2)^{\frac{3}{2}}} - \frac{1}{2\zeta^2\sqrt{1-\zeta^2}}\right) U_\zeta \tag{52}$$

$$U_c = \pm\sqrt{\left(2\sqrt{km}U_\zeta\right)^2 + \left(2\zeta\sqrt{\frac{m}{k}}U_k\right)^2} \tag{53}$$

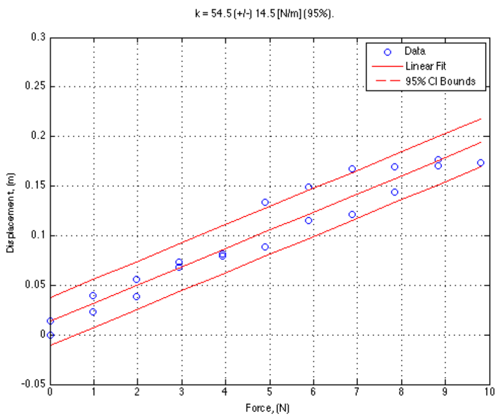
7.2 Bias Uncertainty Values

When calculating the bias uncertainty involved in our apparatus, the data sheets of the equipment were used. In order to determine the maximum uncertainty, the worst case scenario was used. The uncertainty in the accelerometer was first calculated in terms of gravity (g) and then converted to voltage (mV) using the calculated static sensitivity and the nominal value of gravity in meters per second. The uncertainties for both the DAQ board and power supply were determined using each of the specified full scale voltages.

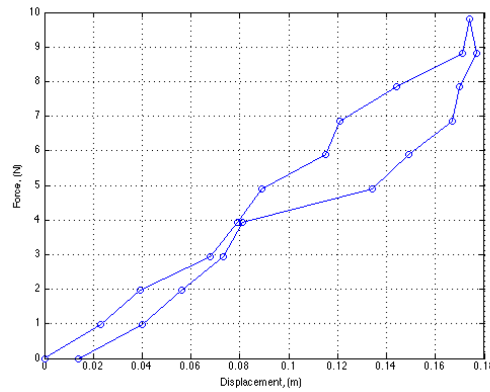
Source of Uncertainty	Value	Units
Accelerometer	± 185	mV
DAQ Board	± 0.229	mV
Power Supply	± 27.5	mV
Spring Hysterisis	± 0.046	m
Ruler	± 0.0005	m

Table 3: Bias uncertainties.

8 Results



(a) This graph shows the linear regression line and the error bounds of the spring constant calibration.



(b) This graph shows the plot of the raw data connected by a line to show the hysteresis.

Figure 7: Spring constant calibration results.

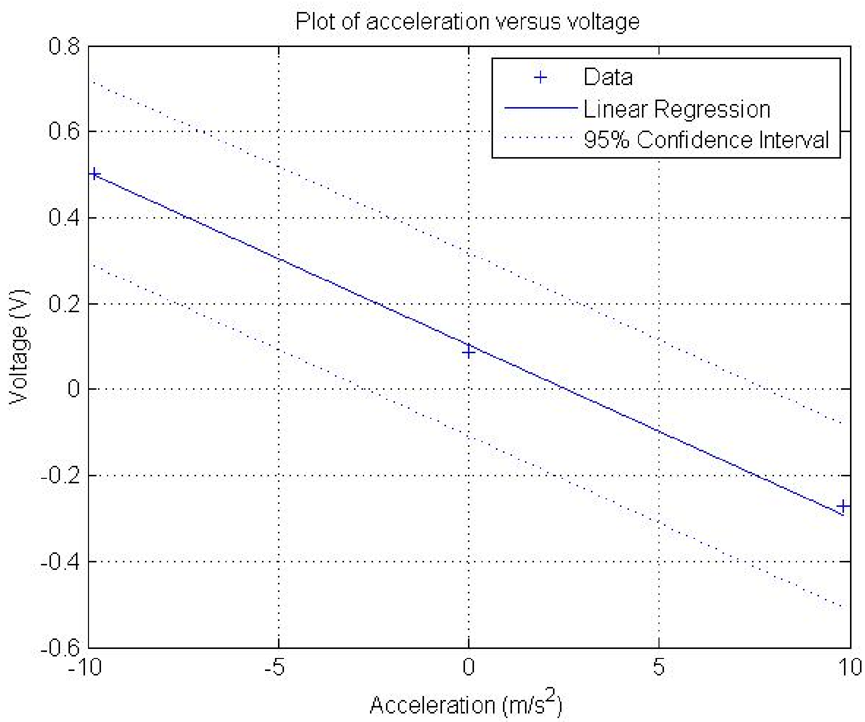


Figure 8: This graph shows the linear regression and the error bounds of the accelerometer calibration.

Parameter	Variable	Value	Uncertainty	Units
Slope of Fit	a_1	-0.0404	–	V/(m/s^2)
Offset of Fit	a_o	0.1036	0.213	V
Static Sensitivity	K	-24.7792	3.5	(m/s^2)/V
Instrument Offset	O	-2.5676	5.28	m/s^2

Table 4: Accelerometer Calibration Results

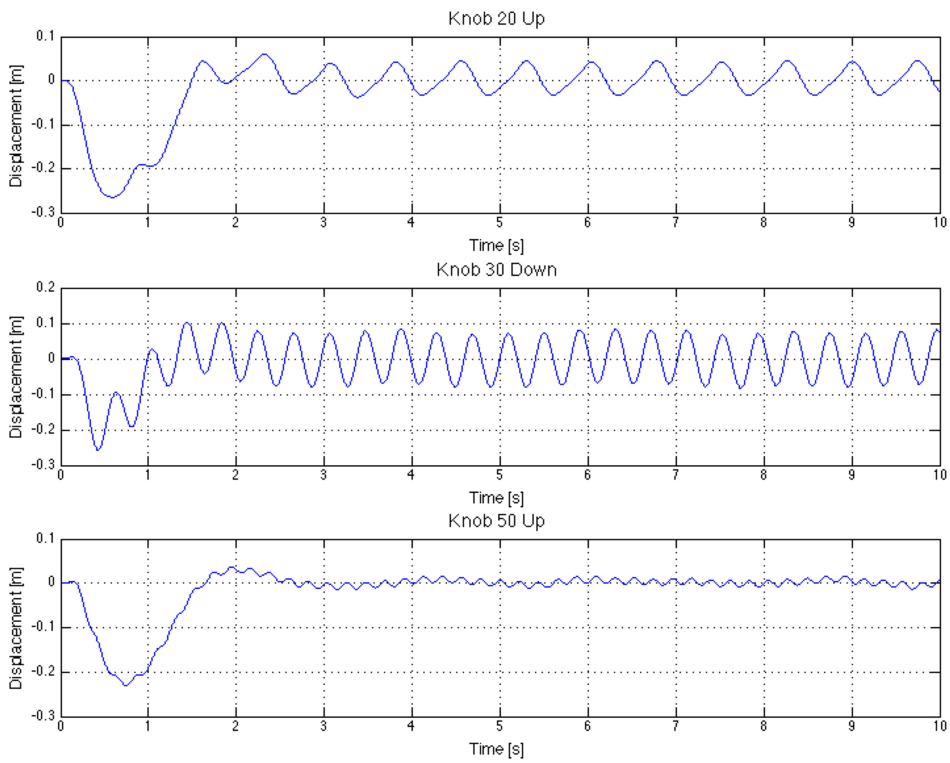


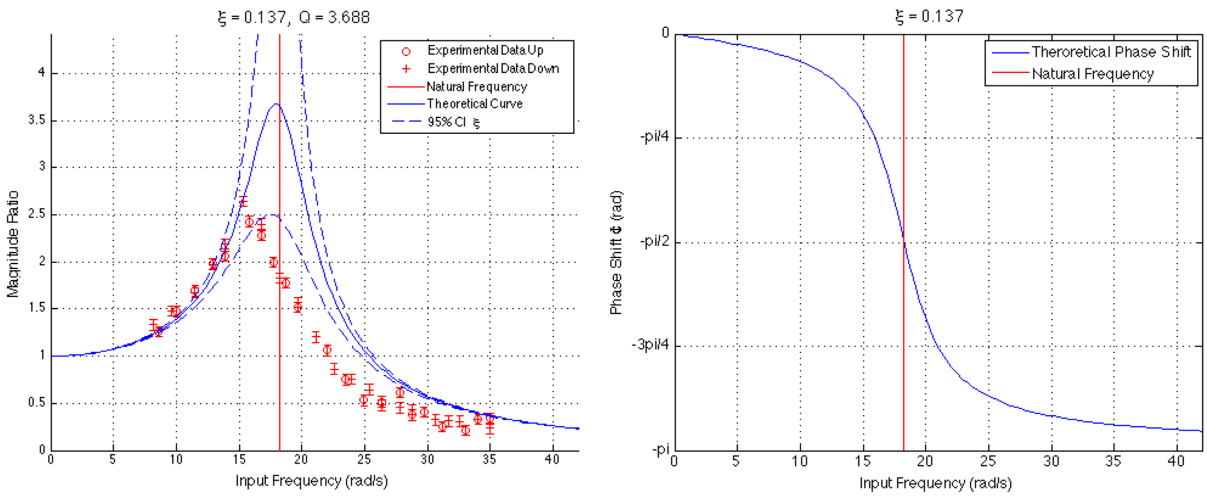
Figure 9: Selected displacement plots at knob settings 20 Up, 30 Down, and 50 Up; respectively. Corresponding frequencies are 1.37 Hz, 2.44 Hz, and 4.58 Hz.

Description	Symbol	Value	Uncertainty	Units
Spring Constant	k	54.5	$\pm$ 14.5	N/m
Natural Frequency	ω_n	18.28	$\pm$4.90	rad/s
Damping Ratio	ζ	0.137	$\pm$0.067	–
Damping Constant	c	0.815	$\pm$0.395	kg/s
Mass	m	0.1629	–	kg
Linear Displacement	X_b	6.3	$\pm$0.05	cm
Max Amplitude	A	8.33	$\pm$0.35	cm
Resonant Frequency	ω_r	15.58	$\pm$0.63	rad/s
Quality Factor	Q	3.69	$\pm$0.18	–

Table 5: Parameter values and uncertainties. All uncertainties calculated using a 95% confidence interval.

Knob Setting	Average Frequency (Hz)
20	1.335
22	1.565
24	1.83
26	2.06
28	2.21
30	2.48
32	2.67
34	2.86
36	3.055
38	3.245
40	3.55
42	3.775
44	4.005
46	4.2
48	4.43
50	4.58
52	4.805
54	5
56	5.225
58	5.42
60	5.57

Table 6: Knob setting frequencies. Uncertainty: .1067 Hz (95%)



(a) Magnitude Ratio

(b) Phase Shift

Figure 10: Experimental data and theoretical curves for magnitude ratio. Phase shift only displays theoretical curve; no data were taken.

9 Discussion and Conclusions

Ultimately, the resulting displacement graphs are consistent with 2nd order system theory. The values are a little off theoretical because of uncertainty, but given the set up we had, everything looks pretty good.

Numerical integration is tricky, so perhaps if we used active circuit filters and integrators, we could have gotten better data.

We were not able to calculate phase lag because we did not know the input function directly. One way this could be accomplished is by attaching another accelerometer to the lower mass and simultaneously recording data from both accelerometers. However, the only information needed to determine phase shift is the relative time between peak of the lower mass and the peak of the upper mass. The peak of the upper mass can be determined from the accelerometer data. The timing of the peak of the lower mass can recorded by attaching something to the lower mass to make an electrical connection or push a button when the mass reaches the bottom of its stroke. This data would be much simpler to analyze and could be read into MATLAB on a separate analog input channel.

Ultimately, the resulting displacement graphs generated through our data analysis are consistent with 2nd order system theory. The magnitude ratios experimentally recorded are off from the theoretical values; however, this difference can be accounted for by the experimental setup. Large sources of error were caused by uncertainty in the spring constant k and the uncertainty caused by numerical integration.

The experimental setup could be improved by replacing the digital filtering and numerical integration with active circuit filters and integrators. This process would generate a continuous signal and prevent interference the computer from using a discrete signal.

Our experimental setup, as is, was not able to calculate phase lag because the input function is not known directly. One way this input function could be measure is by attaching another accelerometer to the lower mass and simultaneously record data from both accelerometers. This process can be simplified, however, because the only information needed to determine phase shift is the relative time between peak of the lower mass and the peak of the upper mass. The peak of the upper mass can be determined from the accelerometer data. The timing of the peak of the lower mass can be recorded by attaching a device to the lower mass to make an electrical connection every time the mass reached the bottom of its stroke. This data would be much simpler to analyze and could be read into MATLAB on a separate analog input channel.

References

[1] Figliola, Richard S., Beasley, Donald E. *Theory and Design for Mechanical Measurements*. John Wiley and Sons 2011.

[2] Verplaetse, Christopher. *Can A Pen Remember What it Has Written Using Inertial Navigation? An Evaluation of Current Accelerometer Technology.*, MIT 1995, from mit.edu, accessed April 2015.

[3] Professor Farny

Appendix

MATLAB Code

All MATLAB code used to perform data acquisition and analysis for the project. In order of appearance:

1. daq_accel.m: Used to acquire the data from the DAQ Board and save it as .csv and .mat files.

2. accelerometer_calibration_v3.m: Performs accelerometer calibration; determines K, O, and U_K.

3. statick.m: Perform calibration to determine spring constant k and uncertainty U_k.

4. accelGUI.m: User interface for analyzing datasets and saving results.

5. filter_code.m, calibration_code, accel2disp.m: Functions used in the accelGUI.m. Each function is a signal conditioning step.

6. project_results.m: Loads results saved by accelGUI.m, produces plots for final report.
7. fft_freq.m, mag_ratio.m, phase_shift.m: Functions used within project_results.m and accelGUI.m to perform more analyses and calculations that do not constitute signal conditioning steps.

dag_accel.m

```
% MATLAB supports M-Series, E-Series, and USB hardware from
% National Instruments with the Data Acquisition Toolbox.  This basic code
% example shows you how to use MATLAB to acquire and analyze data from
% National Instrument hardware in 10 commands.  Additional commands you
% may find useful are included here but commented out.

% Modified: C Farny, ME310

% Use this command to determine Board IDs in system, if needed
hw = daq.getDevices

% Create an analog input object 'handle' using Board ID "Dev1".
ai = daq.createSession('ni');

% Data will be acquired from hardware (BNC) channel 1
ai.addAnalogInputChannel('Dev1','ai1','Voltage');

% Configure the analog input channel for single-ended or differential mode
ai.Channels.InputType='SingleEnded';
% set the full scale input range (note this range is variable and can be
% changed)
ai.Channels.Range = [-1 1]; % (V)

% --- set triggering ---
% Set the sample rate and samples per trigger
% Note: These are settable options and you might want to use different
% values!
ai.Rate = 50e2; % [Hz]
ai.DurationInSeconds = 10; % [s]

% Review the basic configuration of the acquisition by typing
% the name of the 'handle' variable. Note that the handle is responsible
% for transferring information to and from the board and Matlab
ai
% Acquire data
data = startForeground(ai);

fs = ai.Rate;%sampling rate
SampleTime = ai.DurationInSeconds;
dt = 1/fs;%time step [s]

% Graphically plot the results
t = 0:dt:SampleTime-dt;
n = length(t);

%Filter the data
%N.B fs = 50e2 Hz
low_cutoff = 1; %Lower cutoff frequency Hz
high_cutoff = 7.5;%Higher cutoff frequency Hz
order = 3; %Order of the filters being used

%Low Pass
[b,a] = butter(order,high_cutoff/fs,'low');
filtered_y = filter(b,a,data);
```

```
%Band Pass (pass the already filtered data through another filter)
[b2 a2] = butter(order,low_cutoff/fs,'high');
band_y = filter(b2,a2,filtered_y);


%Plot Low Pass filtered data
figure(1)
subplot(2,1,1)
plot(t,data);
title('Raw Data')
xlabel('Time (s)')
ylabel('Voltage (V)')
subplot(2,1,2)
plot(t,filtered_y)
title('Low Pass Filter')
xlabel('Time (s)')
ylabel('Voltage (V)')


%Plot Band Pass filtered data
figure(2)
subplot(2,1,1)
plot(t,data);
title('Raw Data')
xlabel('Time (s)')
ylabel('Voltage (V)')
subplot(2,1,2)
plot(t,band_y)
title('Band Pass Filter')
xlabel('Time (s)')
ylabel('Voltage (V)')

%Figure 3 displays the fft() performed on the data
axis_values = [0,15,0,1]; %Vector for the purpose of ploting everything
fvec = [1:n/2-1]*fs/n; %Frequecny vector

figure(3)
subplot(3,1,1)
raw_data=fft(data);
plot(fvec,abs(raw_data(2:n/2)));
axis(axis_values);
title('Raw Data')
xlabel('Frequency')

subplot(3,1,2)
low_data = fft(filtered_y);
plot(fvec,abs(low_data(2:n/2)))
axis(axis_values);
title('Low Pass Filter')
xlabel('Frequency')

subplot(3,1,3)
band_data = fft(band_y);
plot(fvec,abs(band_data(2:n/2)))
axis(axis_values);
```

```
title('Band Pass Filter')
xlabel('Frequency')

%Save the data into a csv file
user_input = input('Save aquired data? (Y/N): ','s');

if user_input == 'Y'
    file_name = input('Name of file: ','s');
    %Create a data matrix which saves all the acquired and filtered data
    %Data is saved in a csv file with the first column being time, second
    %is raw data, third is low pass filtered data, fourth is band pass
    %filtered data
    data_matrix = zeros(length(t),4);
    data_matrix(:,1) = t;
    data_matrix(:,2) = data;
    data_matrix(:,3) = filtered_y;
    data_matrix(:,4) = band_y;
    csvwrite(sprintf('%s%s',file_name,'.csv'),data_matrix);
else
    fprintf('%s','No data will be saved.')
end


% Clean up
stop(ai);
```

accelerometer_calibration_v3.m

```
%Analysis for the accelerometer calibration
%The data was taken and saved in csv files for three input accelerations:
%-g,0,g

close all
clc
clear

%% Load Data from calibration

%Load all the seperate csv files into their respective matricies
%Columns of each matrix:
%Column 1: Time
%Column 2: Raw Data
%Column 3: Data through a low pass filter
%Column 4: Data through a band pass filter
negative_g_matrix = csvread('Negative_G_Applied.csv');
zero_g_matrix = csvread('Zero_G_Applied.csv');
positive_g_matrix = csvread('Positive_G_Applied.csv');

%Ignore the first second to insure that the data taken only after the
%steady state is reached
negative_g_volts = negative_g_matrix(5000:50000,3);
zero_g_volts = zero_g_matrix(5000:50000,3);
positive_g_volts = positive_g_matrix(5000:50000,3);

%% Linear Fit
acceleration = [-9.81,0,9.81];
```

```
voltage = [mean(negative_g_volts),mean(zero_g_volts),mean(positive_g_volts)];


% Linear Fit for the Daya V(x) = O + K*x = a_0 + a_1*x

%Variables used for calculating the sensitivity and offset
N = length(positive_g_volts); %Number of data points for each voltage
tau = 1.960; %T Value for 95% confidence interval

%Uncertainty for each acceleration point.
U_i(3) = tau.*std(positive_g_volts)./sqrt(N - 1);
U_i(2) = tau.*std(zero_g_volts)./sqrt(N - 1);
U_i(1) = tau.*std(negative_g_volts)./sqrt(N - 1);


w = 1./(U_i).^2;

B = sum(w)*sum(w.*acceleration.^2) - sum(w.*acceleration)^2;

a_o = (sum(w.*acceleration.^2)*sum(w.*voltage) -
sum(w.*acceleration)*sum(w.*acceleration.*voltage))/B;

a_1 = (sum(w)*sum(w.*acceleration.*voltage)-
sum(w.*acceleration)*sum(w.*voltage))/B;

%Y fit for the calibration
y_fit = a_1.*acceleration + a_o;

%% Fit Precision Uncertainty
S_yx = sqrt((sum((y_fit - voltage))^2)/(N-2)); %Standard Deviation for curve
fit

U_p = tau * S_yx/sqrt(N);


%% Fit Bias Uncertainty

%Bias Uncertainty is 213 mV
U_b = .213; %V


%% Total Fit Uncertainty
U_total = sqrt(U_p^2 + U_b^2);

%Uncertainty Lines for a 95% CI
y_up = y_fit + U_total;
y_down = y_fit - U_total;

%% Static Sensitivity

%Using the equation of fit, get sensitivity values so we can get
%acceleration from a voltage
```

```
K = 1/a_1; %(m/s^2)/V

O = a_o/a_1; %(m/s^2)

%% O Uncertainty

U_O = abs(K*.213);

%% K Uncertainty

%2 Degrees of freedom, see fit worksheet page 6
S = sqrt((sum(K - acceleration/voltage).^2)/2);
U_K = 4.303*S/sqrt(3);

%% Results

%Print Relavent Results
fprintf('%s%f%s\n','Slope of fit a1: ',a_1,' V/(m/s^2)')
fprintf('%s%f%s\n','Offset of fit ao: ',a_o,' V')
fprintf('%s%f%s\n','Total Fit Uncertainty: +/-',U_total,' V')
fprintf('%s%f%s%f%s\n','Static Sensitivity K: ',K,' +/-',U_K,' (m/s^2)/V')
fprintf('%s%f%s%f%s\n','Instrument Offset O: ',O,' +/-',U_O,' (m/s^2)')

%Plot Voltage as a function of acceleration
figure(1)
plot(acceleration,voltage,'+')
hold on
plot(acceleration,y_fit)
hold on
plot(acceleration,y_up,':')
hold on
plot(acceleration,y_down,':')
grid on
legend('Data','Linear Regression','95% Confidence Interval')
title('Plot of acceleration versus voltage')
ylabel('Voltage (V)')
xlabel('Acceleration (m/s^2)')

%STILL NEED TO FIGURE OUT THE UNCERTAINTY FOR EACH K AND O
%NEED TO MAKE AN EQUATION THAT JUST SIMPLY GIVES US ACCELERATION GIVEN A
%CERTAIN VOLTAGE
```

Statick.m

```
%David Miller
%4/8/15
%ME310 Project

%Determining the static spring constant.

%Theory:
%F=-kx
%We want to find spring constant k, so
% k=-(F/x)
%F is the force applied; x is the distance moved in response to force F
```

```
%Equipment:
% Oscillator with plate installed
% Set of masses (up to 1kg)
% Meter stick
% Triple beam balance or digital scale (optional)

%Procedure:
%1. Record lower collar offset distance from the base. Record mass of the
%collar, supporting plate, and any screws. Record the distance between
%collars (unextended spring length).

%2. Place a 100g mass on the plate very gently. You do not want the inertia
%of the collar and mass to deflect the spring more than it would under
%frictionless conditions.

%3. Measure the distance between the collar and the base.

%4. Increase mass in increments of 100g until you reach 1000g. When
%changing the masses on the plate, get someone to hold the collar still so
%that it doesn't move while you change the masses. This way the hysteresis
%measurements will not be disrupted.

%5. When you reach 1000g, remove masses in 100g increments and follow the
%same measurement procedure in step 3. Repeat until the plate is fully
%unloaded again.

%% Data:
load('kcalibrate.mat')

%% Analysis

%calculate delta x
x=x/100;                                    %convert from [cm] to [m]
x_0=x(1);                                   %unextended spring length (no
masses, only the collar+plate+screws)
x=-(x-x_0);                                 %make a delta x vector

%calculate force
g=9.81;                                     %gravitational acceleration
F=m.*g;                                     %convert masses to force [N]
F_0=F(1);                                   %this is the force of the
collar+plate+screws
F=(F-F_0);                                  %subtract the initial value from
the rest

%linear regression
p=linreg(F,x);                              %get regression coefficients
x_fit=polyval(p,F);                         %evaluate polynomial at control
points

%% Uncertainty Analysis

%X Precision Fit Uncertainty U_p_fit.x
```

```matlab
N=10;                                          %number control points
nu_fit=N-2;                                    %nu_fit used to determine t
t=2.306;                                       %t_fit, 95% CI
U_p_fit.x=uprecision(t,x,x_fit,'fit');         %precision uncertainty of fit

%X Bias Uncertainty U_b.x
x_hyst=x;                                      %define surrogate x vector for
hysteresis calcs
x_hyst(11)=[];                                 %get rid of middle point to make
it symmetrical
diff=x_hyst(1:10)-fliplr(x_hyst(11:20));       %take the difference coming up
and down
hysteresis=max(diff);                          %find the max of that difference
U_bh.x=hysteresis/2;                           %define bias uncertainty
U_bres.x=.001/2;                               %resolution of meterstick was 1mm
U_t.x=RSS([U_bh.x U_bres.x U_p_fit.x],2);               %calculate total
uncertainty in x

%Spring Constant Uncertainty
k_avg=1/p(1);                                  %the spring constant is 1/a1
because F is on the x axis rather than the y axis. This is essentially out
'average' value for k.
k_data=F(2:end-1)./x(2:end-1);                 %determine k values for each data
point
n=length(k_data);                              %number of k values
nu_k=n-2;                                      %nu to determine t value
t_k=2.110;                                     %determine t value
U_pk=uprecision(t_k,k_data,k_avg*ones(1,n),'fit'); %precision uncertainty of
'fit' of k vals. note that difference being RSS'd is the difference btwn the
k values at each point and the k value determined from the regression

U_k=RSS([U_pk U_bh.x],2);                       %use RSS to get total
uncertainty

%% Results

%plot the calibration regression curve
plot(F,x,'bo',...
    F,x_fit,'r-',...
    F,x_fit+U_t.x,'r--',...
    F,x_fit-U_t.x,'r--')
xlabel('Force, (N)')
ylabel('Displacement, (m)')
legend('Data','Linear Fit','95% CI Bounds')
ax_1=gca;
grid on

%make a title
resultstr = sprintf('k = %.1f (+/-) %.1f [N/m] (95%%).\n',k_avg,U_k); %print
the results
title(ax_1,resultstr)

%plot the raw data to show hysteresis
figure;
F_fit=k_avg.*x_fit-p(2);
```

```
h=plot(x,F,'bo',...
    x,F,'b-');
ylabel('Force, (N)')
xlabel('Displacement, (m)')
grid on
```

project_results.m

```
clear
close all
clc


%Define Parameters
k=54.4591; %N/m
m_accel=28.5;
m_collar=134.4;
m=(m_collar+m_accel)/1000; %mass, kg
X_0=6.3; %linear displacement of scotch yoke
w_n=sqrt(k/m); %natural frequency

%Load Results from GUI
fid=fopen('batch_results_24-Apr-2015.txt');
C=textscan(fid,'%s %s %s'); %Read them into cell arrays of strings
f=str2double(C{2}); %The 2nd and 3rd cells contain frequencies and
displacements
n=length(f)/2; %number of knob settings
X_b=str2double(C{3});

f_up=f(1:n);
f_down=f(n+1:end);
x_up=X_b(1:n);
x_down=X_b(n+1:end);

i_resonance_up=find(x_up==max(x_up)); %Find the index corresponding to
resonanant frequency
i_resonance_down=find(x_down==max(x_down)); %Find the index corresponding to
resonanant frequency

f_r_up=f_up(i_resonance_up); %resonant frequency (Hz)
f_r_down=f_down(i_resonance_down); %resonant frequency (Hz)
f_r_avg=mean([f_r_up f_r_down]);

w_r_up=2*pi*f_r_up; %resonant frequency (rad/s)
w_r_down=2*pi*f_r_down; %resonant frequency (rad/s)
w_r_avg=2*pi*f_r_avg;

w_up=2*pi.*f_up;
w_down=2*pi.*f_down;

xi_avg=.5*(1-(w_r_avg^2)/(w_n^2)); %damping ratio
Q=1/(2*xi_avg.*sqrt(1-xi_avg^2)); %Quality factor
c=xi_avg*2*sqrt(k*m);
```

```
%Calculate magnitude ratios
X_ratio_up=x_up/X_0; %experimental data
X_ratio_down=x_down/X_0; %experimental data

ws=linspace(0,1.2*max([w_up; w_down]))'; %frequency vector
M=mag_ratio(ws,w_n,xi_avg); %theoretical curve

%Calculate theoretical phase shift
Phi=phase_shift(ws,w_n,xi_avg); %theoretical curves

%Magnitude Ratio Plot
figure;
plot(w_up,X_ratio_up,'ro',...
    w_down,X_ratio_down,'r+',...
    ws,M,'b-',...
    [w_n w_n],[0 10],'r-')
titstr1=sprintf('%.3f',xi_avg);
titstr2=sprintf('%.3f',Q);
title(['\xi = ' titstr1 ',  Q = ' titstr2],'FontSize',14);
xlabel('Input Frequency (rad/s)','FontSize',12)
ylabel('Magnitude Ratio','FontSize',12)
legend('Experimental Data Up','Experimental Data Down','Theoretical
Curve','Natural Frequency')
grid on
axis([0 max(ws) 0 1.2*max(M)])



%Phase Shift Plot
figure;
plot(ws,Phi,'b-',...
    [w_n w_n],[0,-2*pi],'r-')
legend('Theroretical Phase Shift','Natural Frequency')
xlabel('Input Frequency (rad/s)','FontSize',12)
ylabel('Phase Shift \Phi (rad)','FontSize',12)
set(gca,'YTick',[-pi -3*pi/4 -pi/2 -pi/4 0],...
    'YTickLabel',{'-pi' '-3pi/4' '-pi/2' '-pi/4' '0'},...
    'FontSize',12);
title(['\xi = ' titstr1],'FontSize',14)
grid on
axis([0 max(ws) -pi 0])

function accelGUI()
%accelGUI GUI for analyzing and displaying data from an accelerometer.
%   David Miller Created 4-10-2015

%Sections:
%1: Define Main Figure
%2: Define Object Positions
%3: Define GUI Objects
%4: Callback Functions
%5: Initialize GUI
%6: Other Functions
```

```
clear
close all
clc

global condition timevector rawvoltage displacement velocity filtered_voltage
%declare them global so they don't have to be loaded to other functions
condition=0;
%this is how the load and analyze buttons communicate and know what's up
have_results=0;

%% 1: Define Main Figure
main_figure_h = figure(...
    'Visible','off',...
    'Units','normalized',...
    'Position',[0.15 0.20 0.8 0.85],...
    'Name','ME310 Position Transduction GUI: Accelerometer',...
    'Color',[0.8 0.8 0.8]...
    );

movegui(main_figure_h,'center')

%clear figure window
clf

%% 2: Define Object Positions
% 'Position',[left bottom width height]
% object_handle.left = ...
% object handle.right = ... etc.

buffer=.025; %buffer space bewteen objects

%------------------------panels---------------------------
panel.width=.25;
panel.height=.25;
panel1.bottom=.7;
panel2.bottom=panel1.bottom-panel.height-buffer;
panel.left=2*buffer;

%------------------------axes-----------------------------
%raw_axes
raw_ax.left=5*buffer+panel.width;
raw_ax.bottom=.7+buffer;
raw_ax.width=1-(buffer+raw_ax.left);
raw_ax.height=.2;

%filter_axes
filt_ax.left=raw_ax.left;
filt_ax.bottom=panel2.bottom+.5*buffer;
filt_ax.width=raw_ax.width;
filt_ax.height=raw_ax.height;

%fft_axes
fft_ax.left=panel.left;
fft_ax.height=1-2*(panel.height+4*buffer);
fft_ax.width=panel.width;
```

```
fft_ax.bottom=3*buffer;

%pos_axes
pos_ax.left=raw_ax.left;
pos_ax.bottom=fft_ax.bottom;
pos_ax.width=raw_ax.width;
pos_ax.height=fft_ax.height-buffer;

%----------------------buttons-----------------------------

%load button
loadb_h.bottom=4*buffer;
loadb_h.width=.35;
loadb_h.height=.2;
loadb_h.left=.5*(1-loadb_h.width);

%analyze button
analb_h.width=loadb_h.width*(2/3);
analb_h.height=loadb_h.height;
analb_h.left=.1*(1-loadb_h.width);
analb_h.bottom=loadb_h.bottom;

%analyze all button
analallb.width=analb_h.width+3*buffer;
analallb.height=analb_h.height;
analallb.left=.5*(1-analallb.width);
analallb.bottom=analb_h.bottom;

%save button
saveb_h.width=loadb_h.width*(2/3);
saveb_h.left=.9*(1-loadb_h.width)+(1/3)*loadb_h.width;
saveb_h.bottom=loadb_h.bottom;
saveb_h.height=loadb_h.height;

%---------------------labels/edits-------------------------

%load label
load_lab.width=.6;
load_lab.height=.2;
load_lab.left=.2;
load_lab.bottom=.7;

%load edit
load_edit.left=load_lab.left;
load_edit.bottom=load_lab.bottom-(load_lab.height+buffer);
load_edit.width=load_lab.width;
load_edit.height=load_lab.height;

%static sensistivity label
sens_lab.width=.35;
sens_lab.height=.2;
sens_lab.left=2*buffer;
sens_lab.bottom=.7;

%static sensistivity edit
```

```
sens_edit.width=sens_lab.width;
sens_edit.height=sens_lab.height;
sens_edit.left=sens_lab.left;
sens_edit.bottom=sens_lab.bottom-(sens_lab.height+buffer);

%toggle filter button
filter_toggle.width=.35;
filter_toggle.height=.2;
filter_toggle.left=.9*(1-loadb_h.width);
filter_toggle.bottom=sens_edit.bottom;

%% 3: Define GUI Objects

%Panel 1
panel_1=uipanel('Title','Load Raw Data','FontSize',12,...
             'BackgroundColor','white',...
             'Units','Normalized',...
             'Position',[panel.left panel1.bottom panel.width panel.height]);
%-------------------------------------------------------------------------%
%Panel 2
panel_2=uipanel('Title','Analysis Options','FontSize',12,...
             'BackgroundColor','white',...
             'Units','Normalized',...
             'Position',[panel.left panel2.bottom panel.width panel.height]);

%-------------------------------------------------------------------------%
%FFT Plot Axes
fft_axh=axes('Visible','on',...
    'Units','Normalized',...
    'Parent',main_figure_h,...
    'Position',[fft_ax.left fft_ax.bottom fft_ax.width fft_ax.height]);

%title + labels
title('Magnitude Spectrum')
xlabel('Frequency (Hz)')
ylabel('Magnitude')
grid on

%-------------------------------------------------------------------------%
%Raw Data Axes
raw_axh=axes('Visible','on',...
    'Units','Normalized',...
    'Parent',main_figure_h,...
    'Position',[raw_ax.left raw_ax.bottom raw_ax.width raw_ax.height]);

%title + labels
title('Raw Data')
xlabel('Time (s)')
ylabel('Voltage (V)')
grid on

%-------------------------------------------------------------------------%
%Filtered Data Axes
filt_axh=axes('Visible','on',...
    'Units','Normalized',...
```

```
    'Parent',main_figure_h,...
    'Position',[filt_ax.left filt_ax.bottom filt_ax.width filt_ax.height]);

%title + labels
title('Filtered Data')
xlabel('Time (s)')
ylabel('Voltage (V)')
grid on

%-------------------------------------------------------------------------%
%Position Data Axes
pos_axh=axes('Visible','on',...
    'Units','Normalized',...
    'Parent',main_figure_h,...
    'Position',[pos_ax.left pos_ax.bottom pos_ax.width pos_ax.height]);

%title + labels
title('Position')
xlabel('Time (s)')
ylabel('Position (m)')
grid on

%-------------------------------------------------------------------------%
%Load Edit Box and Label

%label
load_labh=uicontrol('Style','Text',...
    'Visible','On',...
    'Parent',panel_1,...
    'String','Filename (with extension)',...
    'Units','Normalized',...
    'BackgroundColor',[1 1 1],...
    'Position',[load_lab.left load_lab.bottom load_lab.width
load_lab.height],...
    'FontSize',12);

%edit box
load_edith=uicontrol('Style','Edit',...
    'Visible','On',...
    'Parent',panel_1,...
    'Units','Normalized',...
    'BackgroundColor',[1 1 1],...
    'Position',[load_edit.left load_edit.bottom load_edit.width
load_edit.height],...
    'FontSize',12);

%-------------------------------------------------------------------------%
%Static Sensitivity Edit Box and Label
sens_labh=uicontrol('Style','Text',...
    'Visible','On',...
    'Parent',panel_2,...
    'Units','Normalized',...
    'BackgroundColor',[1 1 1],...
    'String','Static Sensitivity [(m/s^2)/V]',...
```

```
    'Position',[sens_lab.left sens_lab.bottom sens_lab.width
sens_lab.height],...
    'FontSize',12);

sens_edith=uicontrol('Style','Edit',...
    'Visible','On',...
    'Parent',panel_2,...
    'Units','Normalized',...
    'BackgroundColor',[1 1 1],...
    'Position',[sens_edit.left sens_edit.bottom sens_edit.width
sens_edit.height],...
    'FontSize',12);

%----------------------------------------------------------------------------%
%Filter Toggle Button

filter_toggle_h=uicontrol('Style','ToggleButton',...
    'Visible','On',...
    'Units','Normalized',...
    'Parent',panel_2,...
    'Position',[filter_toggle.left filter_toggle.bottom filter_toggle.width
filter_toggle.height],...
    'FontSize',12,...
    'CallBack',@togglefilterfcn);

%----------------------------------------------------------------------------%
%Load Button
load_b=uicontrol('Style','Pushbutton',...
    'Visible','on',...
    'Units','Normalized',...
    'Parent',panel_1,...
    'Position',[loadb_h.left loadb_h.bottom loadb_h.width loadb_h.height],...
    'FontSize',12,...
    'String','Load',...
    'Callback',@startbuttonfcn);

%----------------------------------------------------------------------------%
%Analyze Button
analyze_b=uicontrol('Style','Pushbutton',...
    'Visible','on',...
    'Units','Normalized',...
    'Parent',panel_2,...
    'Position',[analb_h.left analb_h.bottom analb_h.width analb_h.height],...
    'FontSize',12,...
    'String','Analyze',...
    'Callback',@analbuttonfcn);

%----------------------------------------------------------------------------%
%Analyze All Button
analyzeall_b=uicontrol('Style','Pushbutton',...
    'Visible','on',...
    'Units','Normalized',...
    'Parent',panel_2,...
    'Position',[analallb.left analallb.bottom analallb.width
analallb.height],...
    'FontSize',12,...
```

```
    'String','Analyze All',...
    'Callback',@analallbuttonfcn);

%-------------------------------------------------------------------------%
%Save Button

save_b=uicontrol('Style','Pushbutton',...
    'Visible','on',...
    'Units','Normalized',...
    'Parent',panel_2,...
    'Position',[saveb_h.left saveb_h.bottom saveb_h.width saveb_h.height],...
    'FontSize',12,...
    'String','Save',...
    'Callback',@savebuttonfcn);

%% 4: Callback Functions
%-------------------------------------------------------------------------%
%Load Button Pressed ===> Load Data
    function startbuttonfcn(source,evdata)
        fprintf('Load Button Pressed\n');
           if isempty(get(load_edith,'String'))==1
               error('Error: No file name entered')
               set(load_edith,'BackgroundColor',[1 0 0])
           else
           %%% load data
           filename=get(load_edith,'String');
           set(load_edith,'BackgroundColor',[1 1 1])
           file_extension=filename((end-2):end);
           switch file_extension
               case 'mat'
                   try
                   load(filename)                       %load the .mat file
                   catch
                       error('Error: File with that name does not exist.')
%if it doesn't load, display error
                       set(load_edith,'BackgroundColor',[1 0 0]) %and change
the BG color to red
                   end
                   timevector=data(:,1);                %assign the variables
                   rawvoltage=data(:,2);
                   condition=1;
                   fprintf('Data Load Successful. File type: .mat\n')
               case 'csv'
                   %load csv file
                   condition=1;
                   fprintf('Data Load Successful. File type: .csv\n')
               otherwise
                   warning('Warning: File extension not supported. No data
loaded. Please try another file.')
                   condition=0;
           end
           end
    end

%-------------------------------------------------------------------------%
%Analyze Button ===> Filters, FFT, Integration, Plot
```

```
    function analbuttonfcn(source,evdata)
        if condition==0
           warning('Warning: No data has been loaded. Please load data before 
attempting to analyze.')
        elseif condition==1
           fprintf('Data has been loaded successfully.\n');
        %---------------This is where all the MAGIC happens!---------------%
           %%% plot raw data on the raw data axes
           axes(raw_axh)                                %make the raw data axes 
current
           cla
           plot(timevector,rawvoltage)
           title('Raw Data')
           xlabel('Time (s)')
           ylabel('Voltage (V)')
           grid on

           %%% take fft of the data
           t_samp=timevector(2)-timevector(1);
           samplingrate=1/t_samp;
           [y,f,wc]=fft_freq(rawvoltage,samplingrate);
           fprintf('FFT Complete.\n')

           %%% plot the magnitude spectrum
           axes(fft_axh)                                %make the fft axes current
           cla
           semilogx(f,y)
           title('Magnitude Spectrum')
           xlabel('Frequency (Hz)')
           ylabel('Amplitude (V)')
           grid on

           %%% filter data
           state=get(filter_toggle_h,'Value');
           switch state
               case 0 %no filter
                   fprintf('Data not filtered.\n')
                   filtered_voltage=rawvoltage;
               case 1 %filter
                   filtered_voltage=filter_code(rawvoltage);
                   [y2 f2 f_sig]=fft_freq(filtered_voltage,samplingrate); 
%run data through filter(s)
                   hold on
                   semilogx(f2,y2,'r-'); %plot the fft of the filtered data
                   legend('Raw Data','Filtered Data')
                   fprintf('Filtering Complete.\n')


                   %%% plot filtered data on filtered axes
                   axes(filt_axh)                               %make the filtered 
data axes current
                   cla
                   plot(timevector,filtered_voltage)
                   title('Filtered Data')
                   xlabel('Time (s)')
                   ylabel('Voltage (V)')
```

```
                   grid on
           end



           %%% calibrate the filtered voltage to acceleration values
           acceleration=calibration_code(filtered_voltage,sens_edith);
           fprintf('Calibration Complete.\n')

           %%% double integration
           [displacement velocity]=accel2disp(timevector,acceleration);
           fprintf('Integration Complete.\n')

           %%% plot double integration (aka position)
           axes(pos_axh)                                %make the position data axes
current
           cla
           plot(timevector,displacement)
           title('Displacement')
           xlabel('Time (s)')
           ylabel('Distance (m)')
           grid on

           %%% calculate pk-pk displacement
           I=find(and(timevector>6,timevector<8));
           disp_end=displacement(I);
           X_pp=max(disp_end)-min(disp_end);
           fprintf('The peak linear displacement is %.2f
centimeters.\n',100*X_pp);

           %write the frequency and the pk-pk displacement to a
           %.txt file
           fid=fopen('frequencies.txt','a');
           if fid==-1
               warning('''frequencies.txt'' not opened.');
           else
               fprintf('File opened.\n');
           end
           filename=get(load_edith,'String');
           filename(end-2:end)=[];
           fprintf(fid,'%s    %.2f  %.2f\n',filename,f_sig,X_pp*100);

           have_results=1;

           %--------------------------------------------------------------%
        end
    end

%--------------------------------------------------------------------------%
%Analyze All Button
    function analallbuttonfcn(source,evdata)

        %------------------------------------------------------------------%
```

```
        f_new=figure('Visible','Off',...
            'Units','normalized',...
            'Position',[0.3 0.45 0.3 0.4],...
            'Name','Analyze All',...
            'Color',[0.93 0.93 0.93]);

        %-------------------------------------------------------------------%

        rootfilelabel=uicontrol('Style','Text',...
            'Units','Normalized',...
            'Position',[.1 .85 .8 .1],...
            'String','Enter the root file name, with extension, using the
"&num;" tag to denote where the number is.');

        rootfileedit=uicontrol('Style','Edit',...
            'Units','Normalized',...
            'BackgroundColor',[1 1 1],...
            'Position',[.35 .75 .3 .1]);

        %-------------------------------------------------------------------%

        istartlabel=uicontrol('Style','Text',...
            'Units','Normalized',...
            'Position',[.15 .6 .2 .1],...
            'String','Starting Value');

        iiterlabel=uicontrol('Style','Text',...
            'Units','Normalized',...
            'Position',[.4 .6 .2 .1],...
            'String','Step Size');

        istoplabel=uicontrol('Style','Text',...
            'Units','Normalized',...
            'Position',[.65 .6 .2 .1],...
            'String','Ending Value');

        %-------------------------------------------------------------------%

        istartedit=uicontrol('Style','Edit',...
            'BackgroundColor',[1 1 1],...
            'Units','Normalized',...
            'Position',[.15 .5 .2 .1]);

        iiteredit=uicontrol('Style','Edit',...
            'BackgroundColor',[1 1 1],...
            'Units','Normalized',...
            'Position',[.4 .5 .2 .1]);

        istopedit=uicontrol('Style','Edit',...
            'BackgroundColor',[1 1 1],...
            'Units','Normalized',...
            'Position',[.65 .5 .2 .1]);

        %-------------------------------------------------------------------%
```

```
gobutton=uicontrol('Style','Pushbutton',...
    'Units','Normalized',...
    'Position',[.35 .2 .3 .1],...
    'String','Go',...
    'Callback',@gofcn);

%-------------------------------------------------------------------%

bg = uibuttongroup('Visible','off',...
          'Position',[.25 .35 .5 .1],...
          'BackgroundColor',[1 1 1],...
          'SelectionChangeFcn',@SelectionChg);

r1 = uicontrol(bg,'Style',...
          'radiobutton',...
          'String','Up',...
          'Units','Normalized',...
          'Position',[.1 .2 .4 .7],...
          'HandleVisibility','off');

r2 = uicontrol(bg,'Style','radiobutton',...
          'String','Down',...
          'Units','Normalized',...
          'Position',[.6 .2 .4 .7],...
          'HandleVisibility','off');

%-------------------------------------------------------------------%

set(bg,'Visible','On');
set(f_new,'Visible','On');

%======================Callback Functions=========================%

function gofcn(source,evdata)
    %When the 'Go' button is pressed:
    fprintf('Go Button Pressed\n')
    i1=str2double(get(istartedit,'String'));
    i2=str2double(get(istopedit,'String'));
    step=str2double(get(iiteredit,'String'));
    if round(step)~=step
        error('Step size invalid. Please enter an integer step
size.')
    end
    upordown=get(get(bg,'SelectedObject'),'String');
    try %in case someone enters negatives wrong for the step size
        index=i1:step:i2;
    catch
        index=i1:-step:i2;
    end
    n=length(index);
    rootfile=get(rootfileedit,'String');
    for i=1:n %create list of filenames
        num=num2str(index(i));
```

```
                filename{i}=strrep(rootfile,'&num;',num); %replace the tag in
the root filename with proper number
            end
            fprintf('Filenames Organized\n')

            %-------------------------Analysis----------------------------%
            for i = 1:n %for all files
                load(filename{i},'data') %load data
                timevector=data(:,1);    %assign the variables
                rawvoltage=data(:,2);
                %----------This is where all the MAGIC happens!-----------%
                t_samp=timevector(2)-timevector(1);
                samplingrate=1/t_samp;
                [y,f,wc]=fft_freq(rawvoltage,samplingrate); %step 1: FFT
                state=get(filter_toggle_h,'Value');
                switch state %do different things depending on toggle button
state
                    case 0 %no filter
                        fprintf('Data not filtered.\n')
                        filtered_voltage=rawvoltage;
                    case 1 %filter

                %%% filter the voltage signal
                filtered_voltage=filter_code(rawvoltage);
                [y2 f2 f_sig]=fft_freq(filtered_voltage,samplingrate); %run
data through filter(s)
                fprintf('Filtering Complete.\n')

                end

                %%% calibrate the filtered voltage to acceleration values
                acceleration=calibration_code(filtered_voltage,sens_edith);
                fprintf('Calibration Complete.\n')

                %%% double integration
                [displacement velocity]=accel2disp(timevector,acceleration);
                fprintf('Integration Complete.\n')

                %%% calculate pk-pk displacement
                I=find(and(timevector>6,timevector<8));
                disp_end=displacement(I);
                X_pp=max(disp_end)-min(disp_end);
                %fprintf('The peak linear displacement is %.2f
centimeters.\n',100*X_pp);

                %write the frequency and the pk-pk displacement to a
                %.txt file
                results_fname=['batch_results_' date '.txt'];
                fid=fopen(results_fname,'a');
                if fid==-1
                    warning([results_fname ' not opened.']);
                else
                    fprintf('File opened.\n');
                end
                file=filename{i};
```

```
                file(end-3:end)=[]; %get rid of the extension
                fprintf(fid,'%s    %.2f  %.2f\n',file,f_sig,X_pp*100); %print
to file's FID
                fprintf('File appended.\n');

                %save graph data results to .mat file
                results=[timevector displacement velocity filtered_voltage];
                filename_save=['Results_' file '.mat'];
                try
                save(filename_save,'results');
                fprintf('Results Saved.\n')
                catch
                    warning('Warning: Results were not saved.')
                end
            end
        end
        %------------------------------------------------------------------%

        function SelectionChg(source,eventdata)
        %SelectionChg Displays selection changes for radio buttons in UI
Button
        %Groups, or returns strings of selected objects [radio buttons].
        %   To use, simple enter this function as the first line of the
        %   SelectionChangeFcn Callback Function. The output of the function
is the
        %   handle of the selected object.

            disp([get(get(source,'SelectedObject'),'String') 'Selected']);

        end
    %======================================================================%
    end

%--------------------------------------------------------------------------%
%Filter Toggle Button Pressed ===> Turn on/off filter plot and analysis
%step
    function togglefilterfcn(source,evdata)
        state=get(filter_toggle_h,'Value');
        switch state
            case 0
                set(filter_toggle_h,'String','Filter Off');
                axes(filt_axh)                      %make the filtered data
axes current
                cla
                set(filt_axh,'Visible','Off');
            case 1
                set(filt_axh,'Visible','On');
                set(filter_toggle_h,'String','Filter On');
                try
                axes(filt_axh)                      %make the filtered data
axes current
                cla
                plot(timevector,filtered_voltage)
                title('Filtered Data')
                xlabel('Time (s)')
                ylabel('Voltage (V)')
```

```
                grid on
                catch
                end

        end
    end

%------------------------------------------------------------------------%
%Save Results
    function savebuttonfcn(source,evdata)
        fprintf('Save Button Pressed\n')

        switch have_results
            case 0
                warning('No analysis has been performed. No results saved.')
            case 1
        results=[timevector displacement velocity filtered_voltage];
        filename_save=[datestr(now) '_Results.mat'];
        try
        save(filename_save,'results');
        fprintf('Results Saved.\n')
        catch
            warning('Warning: Results were not saved.')
        end
        end
    end

%% 5: Initialize the GUI

K_static=24.995694;
set(filter_toggle_h,'Value',1,...
    'String','Filter On')
set(sens_edith,'String',K_static)
set(main_figure_h,'Visible','On')   %make GUI visible

end
%% 6: Other Functions
%----------------------------------FFT-------------------------------------
function [y,f,f_max] = fft_freq(data,fs)
%fft_freq Magniture spectrum with corresponding frequencies.
%   Uses a Discrete Fourier Transform (FFT) to determine magnitude spectrum
%   fo the signal. Also finds corresponding frequencies so the frequencies
%   of the different magnitudes can be known.
%   [Y,F] = fft_freq(DATA,FS); Where DATA is a vector (or matrix) of data
%   sampled at FS Hertz. Output Y is vector containing the magnitude
%   spectrum, and output F is a vector containing the corresponding
%   frequencies.

L=length(data);
NFFT = 2^nextpow2(L); % Next power of 2 from length of y
Y=fft(data,NFFT)/L;

y = 2*abs(Y(1:NFFT/2+1));
f = fs/2*linspace(0,1,NFFT/2+1);
```

```matlab
f_max=f(find(y==max(y(y<25))));

end

%-----------------------------Filter Code-----------------------------------
function [filtered_voltage] = filter_code(rawvoltage)
%filter_code Apply bandpass filter to raw voltage data.

%Sampling Parameters
fs = 50e2;                  %sampling rate [Hz]
SampleTime = 10;            %sample duration [s]
dt = 1/fs;                  %time step [s]

%Time Vector
t = 0:dt:SampleTime-dt;

%Butterworth Filter Parameters
low_cutoff = 1; %Lower cutoff frequency Hz
high_cutoff = 7.5;%Higher cutoff frequency Hz
order = 3; %Order of the filters being used

%Low Pass Filter (removes noise)
[b,a] = butter(order,high_cutoff/fs,'low');
filtered_y = filter(b,a,rawvoltage);

%High Pass Filter (removes DC offset)
[b2 a2] = butter(order,low_cutoff/fs,'high');
filtered_voltage = filter(b2,a2,filtered_y);

%Notch Filter (also removes dc offset)
%[b2 a2]=iirnotch(.01,.5);
%filtered_voltage = filter(b2,a2,filtered_y);

end

%---------------------------Calibration Code--------------------------------
function [acceleration] = calibration_code(filtered_voltage,sens_edith)
%calibration_code Apply calibration to the filtered data to convert it to
%an acceleration.

if isempty(get(sens_edith,'String'))==1
    error('Error: Please enter a value for static sensitivity.')
else
    K=str2double(get(sens_edith,'String'));
    acceleration=filtered_voltage.*K;
end

end

%------------------------Numerical Integration------------------------------
function [r,varargout] = accel2disp(t,a)
%accel2disp Double integration from acceleration to displacement.
%   Performs a double numerical integration to transform acceleration to
%   velocity.
```

```
%   DISPLACEMENT = accel2disp(TIME,ACCELERATION,CUTOFF_FREQ); where
ACCELERATION is a
%   vector of acceleration values recorded at TIME times. DISPLACEMENT
%   contains the resulting displacements at TIME times. CUTOFF_FREQ is the
%   cutoff frequency in Hz.

%   [DISPLACEMENT,VELOCITY] = accel2disp(TIME,ACCELERATION,CUTOFF_FREQ);
optionally
%   returns the velocity as well as the displacement.

n=nargout;

%Define Sampling and Filter Parameters
t_step=t(2)-t(1);                              %get sampling rate from data
f_samp=1/t_step;                               %convert to frequency

order = 3;                                     %Order of the filters being
used

normalized_wc = (.001)/(f_samp/2);             %Normalized cutoff frequency

%Integrate Acceleration
v=cumtrapz(t,a);

%Filter Velocity
[B1 A1] = butter(order,normalized_wc,'high');  %create filter with
parameters
v_filt = filter(B1,A1,v);                      %filter data
if n==2
    varargout{1}=v_filt;
end
r=cumtrapz(t,v_filt);

end

function [y,f,f_c] = fft_freq(data,fs)
%fft_freq Magniture spectrum with corresponding frequencies.
%   Uses a Discrete Fourier Transform (FFT) to determine magnitude spectrum
%   fo the signal. Also finds corresponding frequencies so the frequencies
%   of the different magnitudes can be known.
%   [Y,F] = fft_freq(DATA,FS); Where DATA is a vector (or matrix) of data
%   sampled at FS Hertz. Output Y is vector containing the magnitude
%   spectrum, and output F is a vector containing the corresponding
%   frequencies.

L=length(data);
NFFT = 2^nextpow2(L); % Next power of 2 from length of y
Y=fft(data,NFFT)/L;

y = 2*abs(Y(1:NFFT/2+1));
f = fs/2*linspace(0,1,NFFT/2+1);

f_c=f(find(y==max(y(y<25))));
```

```
end

function [M] =mag_ratio(omega,varargin)
%mag_ratio Calculate Magnitude Ratio for 1st & 2nd Order Systems.
%   M=mag_ratio(OMEGA,TAU); For 1t Order Systems.
%   M=mag_ratio(OMEGA,OMEGA_N,XI); For 2nd Order Systems.

n=nargin;
switch nargin
    case 2
        tau=varargin{1};
        denominator=sqrt(1+(omega.*tau).^2);
        M=1./denominator;
    case 3
        omega_n=varargin{1};
        xi=varargin{2};
        denominator=sqrt((1-
(omega./omega_n).^2).^2+(2.*xi.*(omega./omega_n)).^2);
        M=1./denominator;
end
end


function [phi] = phase_shift(omega,varargin)
%phase_shift Calculate Phase Shift for 1st & 2nd Order Systems.
%   PHI=phase_shift(OMEGA,TAU); For 1st Order Systems.
%   PHI=phase_shift(OMEGA,OMEGA_N,XI); For 2nd Order Systems.

n=nargin;
switch n
    case 2
        tau=varargin{1};
        phi=-atan(omega.*tau);
    case 3
        omega_n=varargin{1};
        xi=varargin{2};
        fraction=omega./omega_n;
        numerator=2.*xi.*(fraction);
        denominator=1-(fraction).^2;
        argument=numerator./denominator;
        phi=-atan(argument);
        phi(fraction>=1)=phi(fraction>=1)-pi;
end

end
```