Computational Methods with Vibrations

Christian Arriaga-Franco

EML3041 Summer 2025

Table of Contents

Introduction:

Vibrations are everywhere. Think of when you wash your clothes or even when you go for a drive. Your washing machine is rotating, and your car engine is combusting. These are just a few examples of what can cause vibrations but as you can see, they come in many different shapes and sizes. So, are vibrations bad or good? The answer is vibrations can be both good and bad. There are scenarios where a vibration would indicate that a piece of machinery is working healthy. Back to the washing machine, you can almost always expect it to vibrate, yet it's doing exactly what you would expect it to do, wash your clothes. Now too much vibration is something that we would want to eliminate. At this point, if the washing machine is vibrating too much it might even start shaking itself around and lose balance. Believe me, you do not want to see a washing machine lose its balance. As an engineer, it's our job to make sure this would never happen off the production line because it may very well cause more problems to fix which can hinder the safety of the users and credibility of the company. The best way to prevent this from happening would be to study how these vibrations occur through simulation. This can be as simple as putting a mass at the end of a spring and seeing how it oscillates. Then, we add components such as dampeners, whose sole purpose is to dampen the vibrations, and see how the mass would vibrate now. By being able to derive the relationship of vibrations given the scenario, we have now found a way to improve such systems with the goal of safety and reliability. Now, not all data comes in its final form. We will have to extract and modify the data we record to make something out of it that's easier for all of us to understand. To do this, we can use techniques of computational methods. For example, what if all of our data is given in binary systems and we have to convert to base-10 or even finding relative approximate percent errors to see how much of the data we can trust. Most of this will make sense later if it makes no sense right now but the methods used in this report are tools that us engineers use to make it easier to interpret data. MATLAB will be used throughout various parts of this report for computation and visualization for which the code will be shown. Part one is dedicated to the movement of the object and how we determine its movement. Part two analyzes similar properties but for systems that are modeled with multiple masses and spring-mass relations.

Part One: Floating point representation, Maclaurin series, derivation, and Root Finding. How does the movement of the mass depend on time

The first part of this project is all about the moment of our mass. Like stated before, our simulation consisted of a mass oscillating on a spring and that oscillation is our vibration which we are studying. Given data in binary, we must convert that back into something that is easier to work with, base-10. After converting we can better understand how our mass was moving with respect to time as well as perform calculations that can help us determine more information on the movement of the mass such as specific locations at certain times that were not measured and even velocity which was not measured at all.

<u>Chapter One:</u> Extracting binary data, finding relative errors, and deriving series

Chapter one deals with converting data. Most likely due to outdated machinery or even storage compatibility, our data is given to us in binary which is code in base two and in 11-bit systems where the first digit is dedicated to the integer and the rest are dedicated to the decimal. Below is Table 1 which has all binary code and their conversions to base-10 for the given time intervals they were recorded at.

Table 1: Table representing the position of the mass with respect to time extrapolated from binary code

Time (s)	Position x(t)	Original Binary Code
0.00	1.976	1.1111100111
0.25	1.795	1.1100101110
0.50	1.497	1.0111111101
0.75	1.032	1.0000100001
1.00	0.658	0.1010100010
1.25	0.479	0.0111101010
1.50	0.224	0.0011100101
1.75	0.165	0.0010101001
2.00	0.096	0.0001100010
2.25	0.070	0.0001001000
2.50	0.046	0.0000101111

To extract the base-10 values from the binary code, you first must understand the binary code and what each digit truly represents. Since all values are in base-2, we must multiply each of the digits by the value it represents in base-10 and add them together to get the base-10 value. An example is shown below.

$$(1.11)_2 = 1(2^0) + 1(2^{-1}) + 1(2^{-2}) = (1.75)_{10}$$

With this in mind I hope you can get the gist of what must be done to each number to get the base-10 values. However, a slight issue was run into while coding in MATLAB. The issue was that our data was in a matrix and not a vector as you would expect. To solve this, as shown in figure 1, I created a nested for loop that would go through each row of the vector, separating the rows of the binary code making it easier to compile the base-10 values. To explain breifly, the code creates a series of each row using the math shown below:

```
load Project_Data_Chapter1.txt
BIM=Project_Data_Chapter1;
%BIM is our given data,
%just a easier variable to write
[R,C]=size(BIM); %Vector of Rows/Cols
VEC=zeros(1,R); %Final Base-10 vector

for i=1:R % Series for each row
    for j=1:C %Compiling #s in cols
        VEC(i)=VEC(i)+BIM(i,j)*2^(1-j);
    end
end
```

Figure 1: Code for converting from Binary to Base-10

$$\sum_{i=1}^{R}\left(\sum_{j=1}^{C} BIM(i,j) * 2^{1-j}\right)$$

After each row is complete, it takes that rows base-10 value and puts it in a vector which gets used later.

Like stated before, a derivation of the relationship between the components in our system has already been made. A "true" equation has been derived from previous simulation and testing. Now with our data in base-10 we can analyze any error that may have occurred which deviates from the true path that our mass must take. This can tell us whether or not our experimental values align with the theoretical values, verifying the credibility of our data.

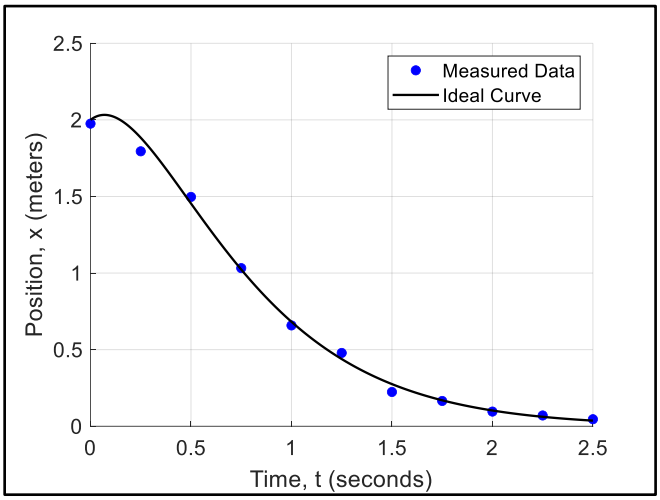

Figure 2: Measured and True values for Position, x(meters) vs. Time, t(seconds) data

As shown in figure 1, the true equation or ideal curve is similar in shape meaning the mass performed as expected. However, it's clear to note that the experimental values are not the same as our theoretical values, although they follow the same shape, none are exactly on the line and have some error. A clear depiction of just how much error is present can be seen in figure 3, where our measured values are compared relative to their respective true values. At the end you see a spike and what most likely provides this error would be the fact that our binary digits only go so far. Even though having 10 digits for the decimal may sound like a lot, when it comes to wanting the truest value like the ones from the true equation, a limitation in number of digits doesn't allow that. Something else that may amplify this error is the fact that we need to be accurate and precise so that we can have the least amount of error, but small deviations in digits and lack of digits don't do that. Even with small true errors, like the one at the end, dividing again by the true value which is also small will just make it bigger. To give a mathematical perspective of how precise we need to be, an example is shown below:

$$|\varepsilon_t|\% = \left|\frac{0.03668 - 0.046}{0.03668}\right| 100 \approx 25\%$$

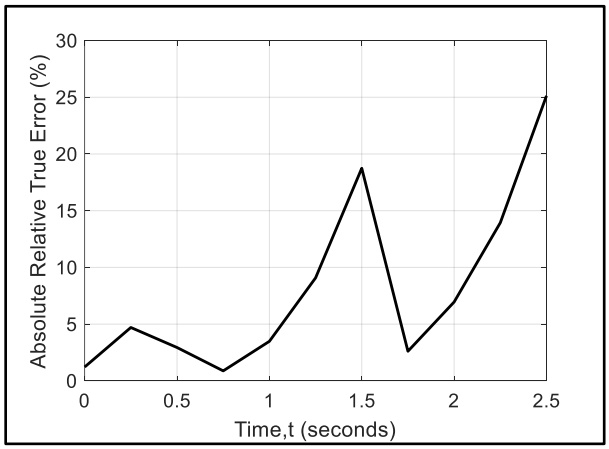

Figure 3: Absolute Relative True Error for measured values

Thinking back to the true equation or ideal curve that is shown in figure 2 and shown below, we were given the equation for that curve to be made but there is also times were engineers are only given equations to approximate our result and these such equations can often be in the form of Maclaurin series which by definition, equals the true function, however, they only become approximation by a limitation in the number of terms that the user utilizes. Below is the derivation of the two-term Maclaurin Series for our true equation.

$$x(t) = x_o e^{-\omega_n t} + (v_o + x_o\omega_n)te^{-\omega_n t}$$

$$Part\ 1: x_o e^{-\omega_n t}\ ; \quad Part\ 2: (v_o + x_o\omega_n)te^{-\omega_n t}$$

$$x(t) = x(0+h) \approx x(0) + x'(0)h$$

$$Part\ 1: x_o e^{-\omega_n t} \approx x_o e^{-\omega_n t} - \omega_n x_o e^{-\omega_n t} \approx x_o - x_o(\omega_n)h$$

$$Part\ 2: (v_o + x_o\omega_n)te^{-\omega_n t} \approx (v_o + x_o\omega_n)te^{-\omega_n t} + (v_o + x_o\omega_n)(e^{-\omega_n t} - \omega_n te^{-\omega_n t})h$$

$$\approx 0 + (v_o + x_o\omega_n)h$$

$$h = t$$

$$\boldsymbol{x(t) = x_o e^{-\omega_n t} + (v_o + x_o\omega_n)te^{-\omega_n t} \approx x_o - x_o(\omega_n)t + (v_o + x_o\omega_n)t}$$

Now with a clear derivation of what the two-term Maclaurin Series looks like for our true function, this approximation is most likely not going to be a very good representation or approximation of the true function. The reason for this would be the number of terms we used which is just two. With such a small number of terms, we are leaving out an enourmous amount of terms that very well make the result more accurate. For example, part of the function, $e^{-\omega_n t}$, will always create a derivative that is negative of the previous one which means that each additional term either adds or subtracts which is how it can get an accurate number but by just leaving two terms and plugging in our coefficients, we get:

$$x(t) = 2 + t$$

By nature, the position should decrease from the initial position when dampening. In no way can this function represent that. To improve this approximation, the best option would be to increase the number of terms in the function, making it more accurate and usable.

<u>Chapter Two:</u> Derivation methods for velocity and acceleration

Chapter two is about finding values that we are not given. This includes being able to find the velocity of the object at a given time. Of course, we already have the true equation for the position of the object so getting its velocity only requires a simple derivative, however, we might not always have that. By using computational methods, such as forward, backward, and central difference differentiation, we can still find the values of values given that we have our values for the objects position. Below, table 2 shows the respective approximate velocities for the time values that were shown in table 1.

Table 2:Table of velocity values for the position values in table one

Time (s)	Velocity v(t)
0.00	-6.107
0.25	-2.454
0.50	-2.557
0.75	-2.335
1.00	-1.585
1.25	-1.092
1.50	-0.792
1.75	-0.351
2.00	-0.259
2.25	-0.145
2.50	-0.235

To calculate the values of velocities at the given times we must go through our three derivation methods and determine which one provides the most accurate answer. Below are the derivation methods that us engineers use to find derivates, or in this case, velocities:

$$FDD: \frac{f(x+h)-f(x)}{h} \qquad BDD: \frac{f(x)-f(x-h)}{h} \qquad CDD: \frac{f(x+h)-f(x-h)}{2h}$$

The main rules that can help us determine what method to use is the size of the h and the data points available. H would be the difference in time and the smaller the h, the more accurate the result because you're closer to the point of interest, but, in this case, all time differences are the same. So, by relying on the data points available, the only valid methods for the first and last data points are FDD and BDD respectively since the first data point doesn't have anything prior and the last data point doesn't have anything after it. As shown in figure 4, the MATLAB code checks the vector of our position values and if it is either the first or last value in the vector, it will perform FDD and BDD respectively. Now, for the values in the middle, there are data points before and after the one of interest so we do CDD to best capture the functions curve and get the most accurate result. In figure 4, if the values of the position vector lie between the initial position and final position, it will execute CDD. All these values are stored in their own vector while the for

function is being run and can be seen in figure 5 along with the ideal curve of the velocity values which is obtained by the derivative of the true position function.

```
for i=1:length(VEC)
    if i==1 %Inital value(only FDD possible)
        VELVEC(i)=(VEC(i+1)-VEC(i))/((Time(i+1)-Time(i)));
    end
    if i>1 && i<length(VEC)%Middle values(CDD best approx)
        VELVEC(i)=(VEC(i+1)-VEC(i-1))/((Time(i+1)-Time(i-1)));
    end
    if i==length(VEC) %Final value(only BDD possible)
    VELVEC(i)=(VEC(i)-VEC(i-1))/((Time(i)-Time(i-1)));
    end
end
```

Figure 4: MATLAB code for the velocity values of the position vector

With a clear picture of the comparison between the velocity values we approximated and the true velocity values, there is error in the sense that the approximations are not on the true velocity values' function but that should be what we expect. What does stand out is the approximation for the velocity value at the beginning of the graph being extremely off from the ideal curve that was created. This resulted in the absolute relative error value of the first data point in figure 6 to be well over 100%. The best explanation for why this initial velocity that we approximated is so far off is because of our position data points for which we created the instantaneous slope. If you look at the true position curve in figure 2, there is actually an increase or slope upwards for the initial time of 0 but with the first two data points that we have, it's impossible to capture that in the approximation.

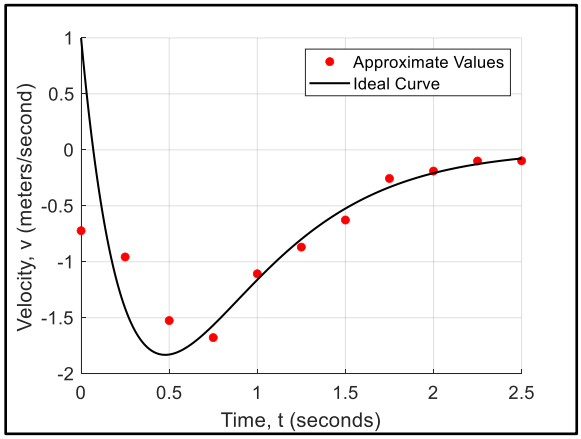

Figure 5: Approximated and True values for Velocity, x(m/s) vs. Time, t(seconds) data

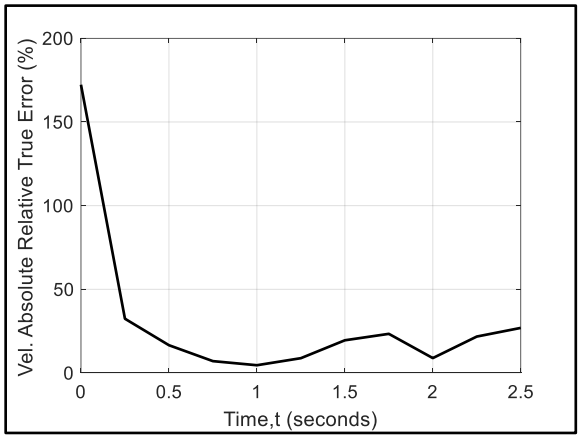

Figure 6: Velocity Absolute Relative True Error for approximated values

The best way to improve this error would be decrease the size of H. If more data points were recorded for this simulation, we would be able to capture the upwards slope at the beginning instead of measuring two points that show a downward slope in the approximation. Doing this would also decrease the overall absolute relative true error because not only would the initial slope

benefit from this, but so would the rest of the data points that also rely on h. Like stated before, a smaller h leads to a more accurate approximation.

With a few minor changes to how we calculated our values for position, it's clear that we can calculate the velocity as our approximations with no changes followed the ideal curve relatively well, but can we calculate the acceleration also. Sure, it would be as easy as taking the derivative with the true velocity equation but what about our approximations, is there a way to find the acceleration values with our data points. The answer is yes. By using Taylor series and some algebra, a formula for the second derivative of a function at a given point is derived.

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)h^2}{2!} + \frac{f'''(x)h^3}{3!} + \ldots$$

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)h^2}{2!} - \frac{f'''(x)h^3}{3!} + \ldots$$

$$f(x+h) + f(x-h) = 2f(x) + \frac{2f''(x)h^2}{2!} + \ldots$$

$$f''(x) \approx \frac{f(x+h) + f(x-h) - 2f(x)}{h^2}$$

Now we have a way to find the second derivation, however, it has its flaws. As you can tell, we are introducing a lot of truncation error by only limiting to 4 terms in each series so an important assumption to be made which will make our values more accurate would be that our h-values are small and, in this case, they are. Apart from the error, this formula is also incapable of finding the approximation for the acceleration values at t=0 and t=2.5 because there is no value in front of beginning or behind the end.

Chapter Three: Using Root Finding Methods for Time

Chapter three is about finding the time that it takes for an object to reach a certain position. Without using the data that we measured, we will use the true equation as the function for which the object follows. Interpolation can be used to find the time when the object reaches a certain spot, but we want to find the most accurate time for when the object reaches that certain point.

To perform this task, we simply set our function with respect to time equal to the position that we want and solve for the time, but to solve it, we will use two separate computational methods and compare. These include the Newton-Raphson method and Bisection method. These are called root-finding methods and although we are not finding any roots, we can easily manipulate the equation so that the function is equal to 0 and solve for time in that sense.

Table 3: Table with time found and number of iterations for Newton Raphson and Bisection Method

Method	**Newton-Raphson**	**Bisection**
Time	1.23599971680	1.23599971677
# of iterations	5	36

A simple overview of how the methods work is that Newton-Raphson finds the tangent line to the guess for which it follows to the x-intercept of that line which becomes the new guess. It will continue doing this until the y-value for that guess is 0. The bisection method uses two initial guesses and works on the rules that a root exists between the guess if the multiple of the y-values of the two guesses result in a negative number. This means that somewhere in between those two x-values, the function crosses the x-axis therefore resulting in a root. So, with each new guess being the mean of the previous two guesses, it will check the sign of that y-value for which it replaces the one with the same sign. This occurs until the two guesses are essentially the same.

To see these methods working in our scenario, when we used the Newton-Raphson method to find the time, the number of iterations it took was extremely lower than for the bisection method. With such a small difference in the time of 0.00000000003, I can assure you that it was also accurate. I can say with confidence that the best method is Newton-Raphson. It's quicker, just as accurate, and only requires one guess. The reason why the Newton-Raphson method excels in this situation is because of the shape of the curve. If you look at figure 2, most of the curve takes on a sort of parabolic curve for which the derivatives of the guesses will benefit from as the x-intercepts of the tangent lines will quickly go towards the root. On the other hand, the bisection methods approach to finding the root will take longer. By the repetitive act of finding the mean for the new guesses, it takes more time and overall, more iterations. As seen in figure 7 the Bisection method doesn't even get close until the 8th iteration and by then the Newton-Raphson method had already found the time. However, the Newton-Raphson method isn't always the best method to use. What if we had a bad guess to start with?

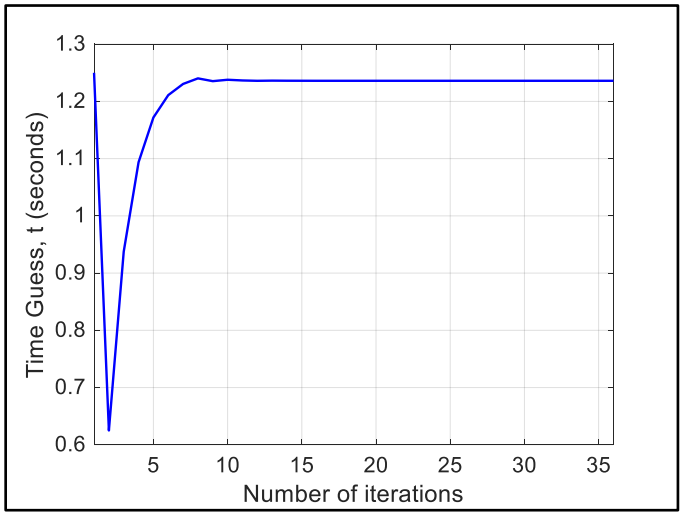

Figure 7: Time Guess vs. # of iterations using the Bisection Root Finding Method

Both the Newton-Raphson method and Bisection method are very dependent on their initial guesses but a bad guess while using the Newton-Raphson method can result in misleading time values or even no time value at all. Below is the formula used for Newton-Raphson method:

$$t_{n+1} = t_n - \frac{x(t_n)}{x'(t_n)}$$

As you can see, the function requires the function is continuous along the interval of interest and it is. Obviously, we wouldn't choose a guess outside of the interval because it's not what we are looking for so any guess other than within 0-2.5 is invalid. Next is the derivative of the function and how a local maximum would affect the function. Taking a look at figure 2, it's clear that a local maximum is present at the beginning of the true equation. Imagine taking the tangent line of point at that local maximum. Do you think that the x-intercept of that tangent line would be anywhere near the root. The answer is no but to explain it mathematically, below would be how the function is performed if we choose the initial guess to be the time at the local maximum.

$$x(t) = 2e^{-\sqrt{6}t} + \left(1 + 2\sqrt{6}\right)te^{-\sqrt{6}t}$$

$$x'(t) = v(t) = -2\sqrt{6}e^{-\sqrt{6}t} + \left(1 + 2\sqrt{6}\right)(e^{-\sqrt{6}t} - \sqrt{6}te^{-\sqrt{6}t})$$

$$t_{n+1} = t_n - \frac{x(t_n)}{x'(t_n)};\ \ Guess \rightarrow t = 0.0692$$

$$x(0.0692) = 2e^{-\sqrt{6}*0.0692} + \left(1 + 2\sqrt{6}\right) * 0.0692 * e^{-\sqrt{6}*0.092} \approx 2.033$$

$$v(0.0692) = -2\sqrt{6}e^{-\sqrt{6}*0.0692} + \left(1 + 2\sqrt{6}\right)\left(e^{-\sqrt{6}*0.0692} - \sqrt{6} * 0.0692 * e^{-\sqrt{6}*0.0692}\right) \approx .000$$

$$t = 0.069 - \frac{2.033}{0.000} = UNDEFINED$$

Looking at the function above, the result is undefined leading to us not being able to get another guess and continue the method with the initial guess that we chose. This is the con of using this method where critical points such as when velocity is equal to 0, using that as an initial guess will result in error. We can even see this in MATLAB when we set the initial guess to be t=0.062, the result is NAN, suggesting that there is no x-intercept that can be found.

<u>Part Two:</u> Matrices in systems of equations and regression. Analyzing systems with multiple masses and spring behavior.

Part two is about mechanical systems that have more than one mass and analyzing spring behavior. In our case, we will be analyzing a system with three masses connected by 4 springs as well as one with only two springs. This is the model of something common we say on a day-to-day basis, a building. Understanding the behavior of them and how they move is important because

we wouldn't want any buildings to collapse due to any vibrations they may experience such as when hurricanes or earthquakes occur. To do this, we will use computational methods that involve matrices as well as regression models.

Chapter Four: Using Matrices to solve systems of equations to determine displacements of the masses.

Chapter four is similar to chapter one where our end goal is to find the displacement of the masses in the system. This scenario, however, deals with three masses which are connected by springs. The masses represent the floors, and the springs represent the vertical columns supporting each floor. Understanding this system also requires a system of equations that relates the displacements of the floors with each other given that masses and spring constants are known and they are. Below are the given equations.

$$k_1x_1 + k_2(x_1 - x_2) = m_1g \rightarrow 5000x_1 + 10000(x_1 - x_2) = 9.81 * 1000$$

$$\rightarrow 15000x_1 - 10000x_2 = 9810$$

$$-k_2(x_1 - x_2) - k_3(x_3 - x_2) = m_2g \rightarrow -10000(x_1 - x_2) - 12000(x_3 - x_2) = 9.81 * 2000$$

$$\rightarrow -10000x_1 + 22000x_2 - 12000x_3 = 19620$$

$$k_3(x_3 - x_2) + k_4x_3 = m_3g \rightarrow 12000(x_3 - x_2) + 5000x_3 = 9.81 * 1500$$

$$\rightarrow -12000x_2 + 17000x_3 = 14715$$

With the given equations, we can use simple algebra to solve for the unknowns because we do have three equations and three unknowns but the computational method that can best help us quickly solve for the unknowns all at once is with matrices. Below are the matrices that are formed with the equations that we have:

$$\underline{A} * \underline{x} = \underline{B}$$

$$\begin{bmatrix} 15000 & -10000 & 0 \\ -10000 & 22000 & -12000 \\ 0 & -12000 & 17000 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 9810 \\ 19620 \\ 14715 \end{bmatrix}$$

Now that we have the values in their respective matrices, we can solve for the unknowns and normally, methods such as LU decomposition or Gaussian elimination would be required to solve for the unknown matrix "x," but with MATLAB, a simple line of code can be used, x=A\B, to solve for the unknowns that were found, shown in table 4.

Table 4: Table of Displacement, x (meters), values found from solving equations

Displacement	x_1	x_2	x_3
x (meters)	4.2043	5.3254	4.6247

Now an engineer has suggested a different model to best simulate the vibration of the building in scenarios such as hurricanes or even earthquakes. Figure 8 is the previous model that was used for the calculations above and figure 9 is the new model that we will analyze for vibrations.

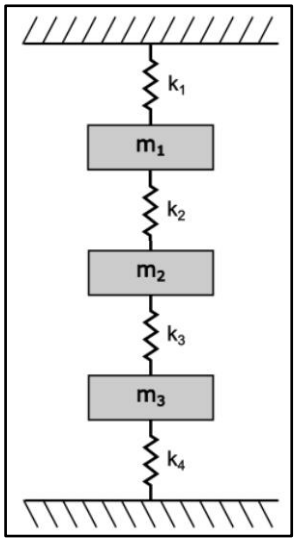

Figure 8: Vibration model of building

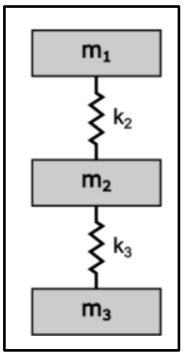

Figure 9: Suggested model from a new engineer

Notice that the new model in figure 9 has no spring attached to the bottom or top of the system like in figure 8 so to get the equations and solve for the displacements of the masses, we will consider k1 and k4 as 0.

$$k_1x_1 + k_2(x_1 - x_2) = m_1g \rightarrow\ 0x_1 + 10000(x_1 - x_2) = 9.81 * 1000$$

$$\rightarrow 10000x_1 - 10000x_2 = 9810$$

$$-k_2(x_1 - x_2) - k_3(x_3 - x_2) = m_2g \rightarrow -10000(x_1 - x_2) - 12000(x_3 - x_2) = 9.81 * 2000$$

$$\rightarrow -10000x_1 + 22000x_2 - 12000x_3 = 19620$$

$$k_3(x_3 - x_2) + k_4x_3 = m_3g \rightarrow 12000(x_3 - x_2) + 0x_3 = 9.81 * 1500$$

$$\rightarrow 12000x_3 - 12000x_2 = 14715$$

All that is left is to solve for the unknowns and similar to the previous model, we will use matrices, but it seems that for this model, there may not be any solution because the matrix is singular, meaning that the determinant is 0. If the determinant is 0, it means that it's impossible to get an inverse for the matrix A, therefore making it impossible to solve for x. Below shows our determinant coming out to 0 by method of forward elimination.

$$\underline{A} * \underline{x} = \underline{B}$$

$$\begin{bmatrix} 10000 & -10000 & 0 \\ -10000 & 22000 & -12000 \\ 0 & -12000 & 12000 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 9810 \\ 19620 \\ 14715 \end{bmatrix}$$

$$\underline{A} = \begin{bmatrix} 10000 & -10000 & 0 \\ -10000 & 22000 & -12000 \\ 0 & -12000 & 12000 \end{bmatrix}$$

$$E_{1,mod} = \left(\frac{-10000}{10000}\right)[10000 \quad -10000 \quad 0] = [-10000 \quad 10000 \quad 0]$$

$$E_{2,new} = [-10000 \quad 22000 \quad -12000] - [-10000 \quad 10000 \quad 0] = [0 \quad 12000 \quad -12000]$$

$$\underline{A} = \begin{bmatrix} 10000 & -10000 & 0 \\ 0 & 12000 & -12000 \\ 0 & -12000 & 12000 \end{bmatrix}$$

$$E_{2,mod} = \left(\frac{-12000}{12000}\right)[0 \quad 12000 \quad -12000] = [0 \quad -12000 \quad 12000]$$

$$E_{2,new} = [0 \quad -12000 \quad 12000] - [0 \quad -12000 \quad 12000] = [0 \quad 0 \quad 0]$$

$$\underline{A} = \begin{bmatrix} 10000 & -10000 & 0 \\ 0 & 12000 & -12000 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\det(\underline{A}) = \underline{A}_{11} * \underline{A}_{22} * \underline{A}_{33} = 10000 * 12000 * 0 = 0$$

After completing the forward elimination, the determinate of the new matrix A comes out to 0, meaning that the matrix is in fact singular so no solutions will be possible in this scenario. Including the springs in the model for which the new engineer took out is most likely the better option and talking about springs, we will be analyzing that next.

<u>Chapter Six:</u> Using regression to Analyze the relationship between the force and displacement in a spring,

Chapter six focuses on one of the components that we use to simulate vibrations. This would be the spring which allows the masses to oscillate in the way we want them to show vibrations. In most cases a spring can be modeled by using an equation like Hooke's Law where the force and displacement are linearly proportional through the springs stiffness as shown below

$$F = kx$$

However, this is not always the case. For example, when we increase the load on the springs, the equation now becomes nonlinear where there is an additive force with a spring constant that is nonlinear. This is shown below:

$$F = k_1 x + k_3 x^3$$

With a handful of measurements including displacements with their respective forces, we can now derive an equation to find the spring constants, both linear and nonlinear. To do so, we will use the method of regression. After hearing this multiple times throughout the report, you're probably curious as to what it is. To put it simply, it's the line of best fit. Being able to derive an equation that best models the relationship between the displacement and the force is important because it can help us best determine values that were not measured. To do this, we start all equations on the basis of residuals, the distance between our measured values to the line of best fit. If you can imagine a bunch of points scattered in a graph with a clear linear pattern, the line would be perfectly in the bunch of the points, representing the ideal path. Optimizing the equation for the sum of the squares of residuals is what can help us determine the equations for the values for the spring constants. From calculus we know that in order to optimize we must take the derivative and set the function equal to 0. This enables us to find local minimums and maximums but since we are squaring the residuals, we will always be looking at minimizing. In the case of our function, it has more than one variable so to optimize this function, we must take partial derivates of both, set both equal to 0, and solve. Eventually we have a system of equations with two unknowns with two equations and just like before, we will use computational methods to solve. Below is the derivation for constants k1 and k3 using regression and matrices(F=y):

$$S_r = \sum_{i=1}^{N} [y_i - f(x_i)]^2 = \sum_{i=1}^{N} [y_i - k_1 x_i - k_3 {x_i}^3]^2$$

$$\frac{\partial S_r}{\partial k_1} = 0 = \sum_{i=1}^{N} 2(y_i - k_1 x_i - k_3 {x_i}^3)(-x_i) \rightarrow 0 = \sum_{i=1}^{N} (y_i - k_1 x_i - k_3 {x_i}^3)(x_i)$$

$$\rightarrow 0 = \sum_{i=1}^{N} y_i x_i - k_1 \sum_{i=1}^{N} {x_i}^2 - k_3 \sum_{i=1}^{N} {x_i}^3 \rightarrow \boldsymbol{k_1 \sum_{i=1}^{N} {x_i}^2 + k_3 \sum_{i=1}^{N} {x_i}^3 = \sum_{i=1}^{N} y_i x_i}$$

$$\frac{\partial S_r}{\partial k_3} = 0 = \sum_{i=1}^{N} 2(y_i - k_1 x_i - k_3 {x_i}^3)(-{x_i}^3) \rightarrow 0 = \sum_{i=1}^{N} (y_i - k_1 x_i - k_3 {x_i}^3)({x_i}^3)$$

$$\rightarrow 0 = \sum_{i=1}^{N} y_i {x_i}^3 - k_1 \sum_{i=1}^{N} {x_i}^4 - k_3 \sum_{i=1}^{N} {x_i}^6 \rightarrow \boldsymbol{k_1 \sum_{i=1}^{N} {x_i}^4 + k_3 \sum_{i=1}^{N} {x_i}^6 = \sum_{i=1}^{N} y_i {x_i}^3}$$

$$\underline{A} * \underline{x} = \underline{B}$$

$$\begin{bmatrix} \sum_{i=1}^{N} {x_i}^2 & \sum_{i=1}^{N} {x_i}^3 \\ \sum_{i=1}^{N} {x_i}^4 & \sum_{i=1}^{N} {x_i}^6 \end{bmatrix} * \begin{bmatrix} k_1 \\ k_3 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} y_i x_i \\ \sum_{i=1}^{N} y_i {x_i}^3 \end{bmatrix}$$

All that's left is to solve for the spring constants and we will utilize MATLAB like before where we create the matrices and solve for the matrices of unknowns, x. This gives us the values in table 5.

Table 5: Table of Spring constant values

Spring Constants	$\boldsymbol{k_1\ (N/m)}$	$\boldsymbol{k_3 (N/m^3)}$
Values	60.387	5.656

$$F = 60.387x + 5.565x^3$$

This is the result of using regression and matrices to find the line of best fit and in figure 10 you can see how well the equation we created fits the data points that we measured. Not every point is on the line, but we create that line best fits the relationship between the springs force and the displacement it experiences. Now with this equation, calculations of points that were not measured can be done (Ex. x=1.1) as well as performing derivatives or integrals of the function but is this truly the line of best fit.

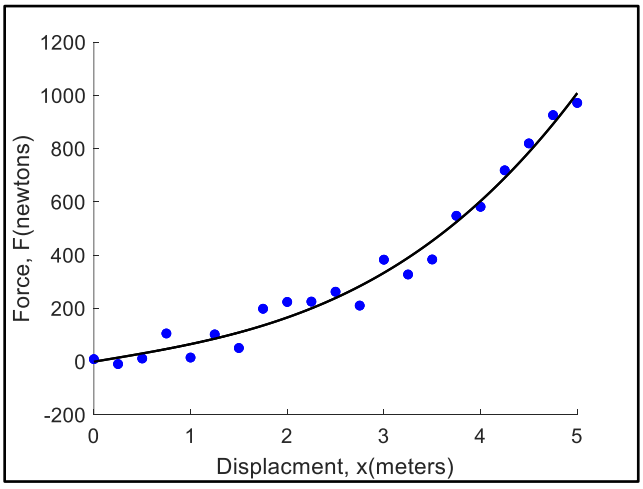

Figure 10: Force (N) vs. Displacement (m) of nonlinear spring with scattered data points and regression line

MATLAB has a built-in function called polyfit which will create a line of best fit in the order that you want it to, the only difference is that all functions with increasing order are generic and not special or limited to certain coefficient values like the function that we see in figure 10. For this problem, if we use polyfit, it will create a function of 3rd order in the form below and with the respective coefficient values in table 6.

$$F(x) = k_3x^3 + k_2x^2 + k_1x + C$$

Table 6: Polyfit MATLAB function coefficients

Spring Constants	k_3 (N/m^3)	k_2 (N/m^2)	k_1 (N/m)	$C(const.)$ (N)
Values	7.644	-15.12	90.17	-10.24

Although the curve for the polyfit function follows the spring regression curve and points almost the same as shown in figure 11, it would still be best to use the function that we used for the regression. In general, if the spring force is based off the regression equation, creating a new one with more coefficients might cause error in finding derivatives like the velocity or acceleration. Also, having a constant at the end just doesn't make any sense because why would the spring have a force before even moving. Limiting the function with coefficients that are already known will produce better results than having to use a function with imaginary coefficients that may alter important values are needed in later calculations.

An easy example where a method like regression may be useful is at the heart of engineering, physics, specifically projectile motion. While performing an experiment involving projectile motion you may not know the initial velocity but you have positions at certain times as well as your gravity coefficient. Knowing the general equation for projectile motion can help derive the formulas needed to find the values of the coefficients that are the initial velocities of the projectile. With the values and the Pythagorean theorem, you can find the total initial velocity.

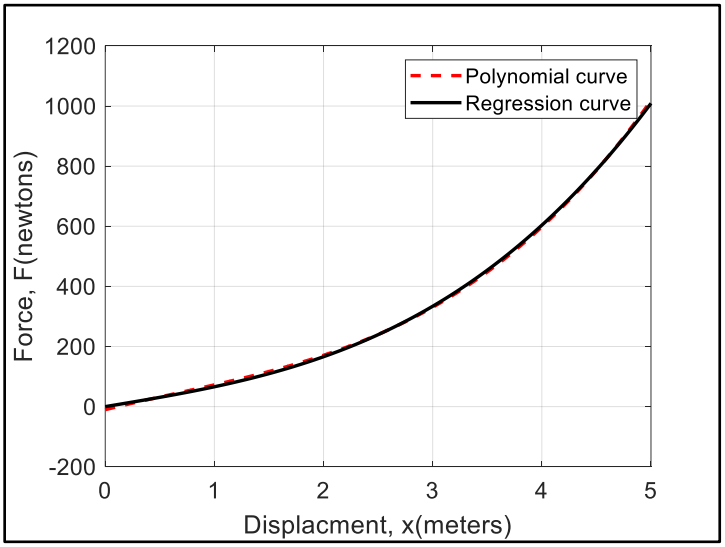

Figure 11: Polyfit and regression curve of Force ,F(newtons) vs. Time, S(seconds)

The derivation for this can be seen on the next page where we solve for the initial velocities in the x and y directions.

$$x = v_{ox}t$$

$$y = v_{oy}t - \frac{9.81}{2}t^2$$

$$S_r = \sum_{i=1}^{N}[y_i - f(t_i)]^2 = \sum_{i=1}^{N}\left[y_i - v_{oy}t_i + \frac{9.81}{2}{t_i}^2\right]^2$$

$$\frac{\partial S_r}{\partial v_{oy}} = \sum_{i=1}^{N} 2\left(y_i - v_{oy}t_i + 4.905{t_i}^2\right)(-t_i) = \sum_{i=1}^{N} y_i t_i - v_{oy}\sum_{i=1}^{N}{t_i}^2 + 4.905\sum_{i=1}^{N}{t_i}^3 = 0$$

$$v_{oy} = \frac{\sum_{i=1}^{N} y_i t_i + 4.905\sum_{i=1}^{N}{t_i}^3}{\sum_{i=1}^{N}{t_i}^2}$$

$$S_r = \sum_{i=1}^{N}[x_i - f(t_i)]^2 = \sum_{i=1}^{N}[x_i - v_{ox}t_i]^2$$

$$\frac{\partial S_r}{\partial v_{ox}} = \sum_{i=1}^{N} 2(x_i - v_{ox}t_i)(-t_i) = \sum_{i=1}^{N} x_i t_i - v_{ox}\sum_{i=1}^{N}{t_i}^2 = 0$$

$$v_{ox} = \frac{\sum_{i=1}^{N} x_i t_i}{\sum_{i=1}^{N}{t_i}^2}$$

Conclusion:

We have studied multiple systems for vibrations and now know how we can apply computational methods to them. As mechanical engineers, we must deal with a lot of moving parts when something is moving, its movement most likely changes with respect to time. We can see this in these systems that we studied and the physics example I provided. Now, I can see that computational methods provide a way to get results when not all measurements are present. What if the instrument we use can only record the location of the object every 0.5 seconds. A scenario like this is what motivates us to use these methods. And sure, maybe the error was high in some cases but the results we get are still way better than if we just guessed a value in between. Most of the time, the values that we approximate followed the curve relatively close and when we created our own curves, the accuracy as to how well they represented what truly happens was pretty close. Now that the project is done, it's now clear to me that engineering is not as precise as I thought it out to be. We can have some tolerance with measuring because of what we are able to do with that data to fill in for that tolerance. However, engineers like us should be able to predict. Predict where an object would be in some amount of time or even predict where the object was some amount of time ago. Problems that require predictions like these should take full advantage of the methods that we used throughout the project, and this is only a little of the numerous methods available.