This was an assignment for my Dynamics and Control Systems lab. Here, we had to verify the results of an academic paper, in which they used the Windkessel model, an RLC analog to model the blood flow through the body, where R was fluid resistance, L was blood inertia, and C was the elasticity of the veins. The heart was the voltage source.

This was the first time I had used Simulink, and created block diagrams to be able to output system behavior and graph it

On the next page, it goes into detail of howe the block diagram was derived, and the “equations of motion”, as well as the heart “input voltage”.

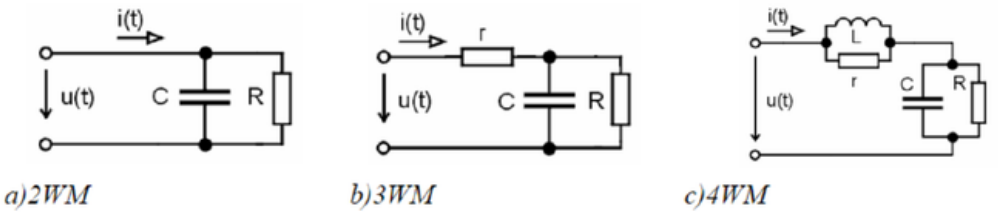
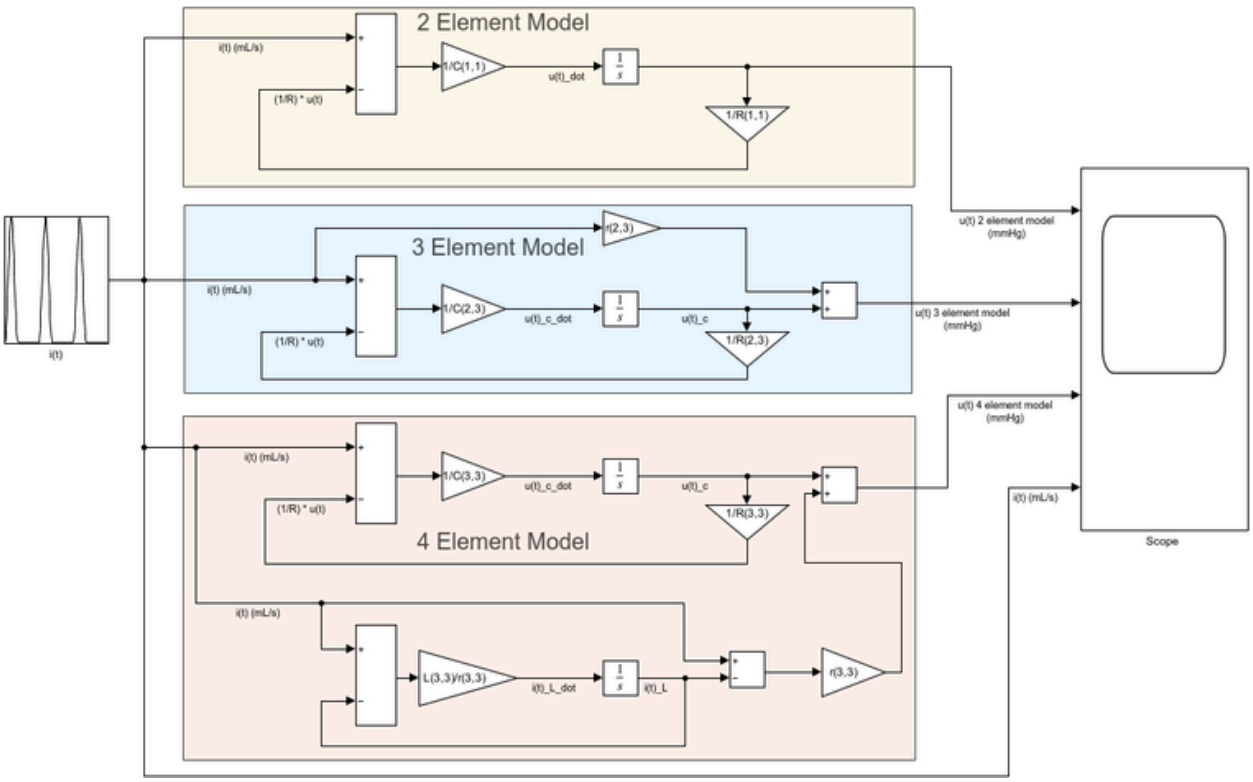
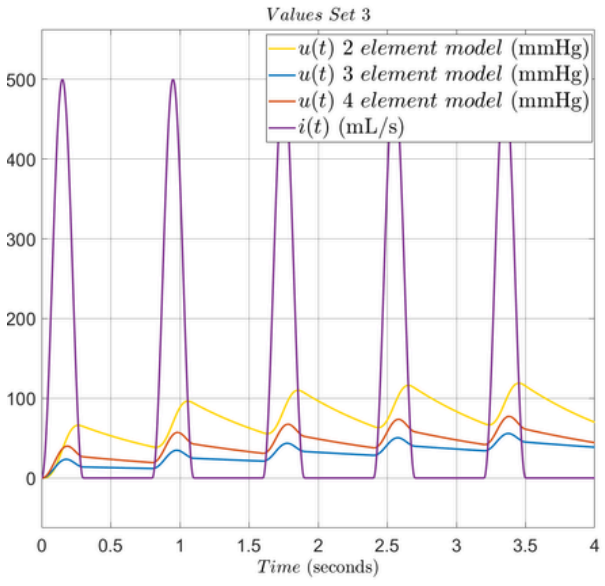
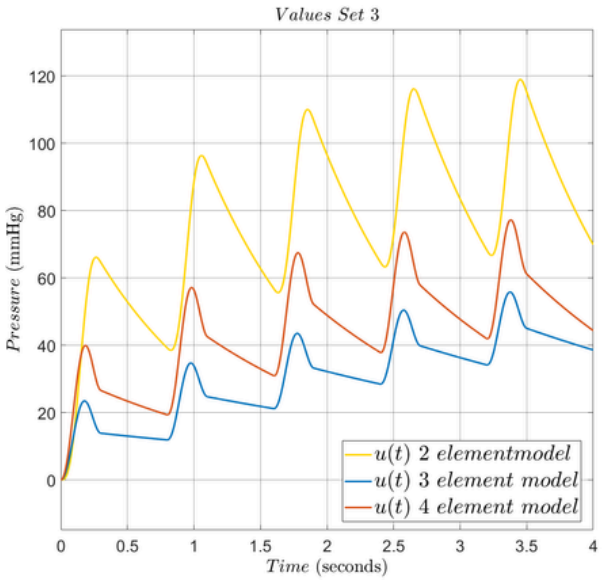


Figure 9: Windkessel Circuit Models from "Windkessel Analysis in MATLAB"







The heart's pumping was modeled as a periodic pulsing, as shown by the function and graph below. The input here was actually the heart's blood flow i(t), and the output was the heart's blood pressure, u(t). In Simulink, this was modeled using a repeating table, in which the values were coded in, as shown below:

$$i(t) = \begin{cases} I_0 \sin^2\left(\frac{\pi t}{T_s}\right) & 0 \le t \le T_s \\ 0 & T_s \le t \le T \end{cases} \quad (7)$$

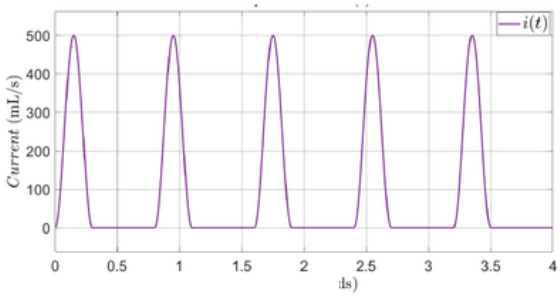
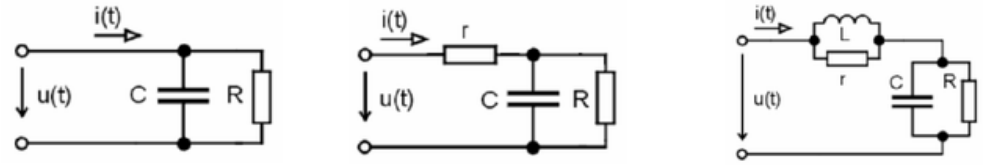


Figure 3: i(t) as a function of time, graphed as a repeated sequence

```
I_0 = 500;
t_s = 0.3;
t_end_cyc = 0.8;
resolution = 1000;
sampleTime = [t_end_cyc/resolution , 0]; %period,offset

t = linspace(0,t_end_cyc,resolution);
%t_flat = linspace(t_s,t_end_cyc,resolution);
t_humplast_index = max(find(t <= t_s));

i = zeros(1,resolution);
i(1:t_humplast_index) = I_0 * sin( pi* t(1:t_humplast_index) / t_s ) .^2;
i(t_humplast_index + 1 : resolution) = 0;

i_of_t = [t',i'];
```

Below, are the Windkessel models, and their respective equations of motion.



a)2WM *b)3WM* *c)4WM*

Figure 9: Windkessel Circuit Models from "Windkessel Analysis in MATLAB"

Windkessel Equations For 2-, 3-, 4- Element models:

- 2-Element:

$$i(t) = \frac{1}{R}u(t) + C\frac{du(t)}{dt} \Rightarrow \frac{du(t)}{dt} = \frac{1}{C}\left(i(t) - \frac{1}{R}u(t)\right) \quad (1)$$

- 3-Element:

$$i(t) = \frac{1}{R}u_c(t) + C\frac{du_c(t)}{dt} \Rightarrow \frac{du_c(t)}{dt} = \frac{1}{C}\left(i(t) - \frac{1}{R}u_c(t)\right), \quad (2)$$

$$u(t) = r \cdot i(t) + u_c(t) \quad (3)$$

- 4-Element:

$$i(t) = \frac{1}{R}u_c(t) + C\frac{du_c(t)}{dt} \Rightarrow \frac{du_c(t)}{dt} = \frac{1}{C}\left(i(t) - \frac{1}{R}u_c(t)\right), \quad (2)$$

$$i(t) = i_L(t) + \frac{L}{R}\frac{di_L(t)}{dt}, \quad (4)$$

$$u(t) = u_c(t) + r\left(i(t) - i_L(t)\right) \quad (5)$$

To create the graphs, the block diagrams needed to be created first. The Windkessel models provide a set of differential equations for each n-element model, with $i(t)$ being the input, or forcing function, and $u(t)$ being the output variable. However, being differential equations, the output is mixed in with its own derivatives. To get a clear output $u(t)$ (which is solved numerically), the lowest derivative given must be integrated several times until the output function is returned. For example, the 2-element Windkessel model gives the equations:

$$i(t) = \frac{1}{R}u(t) + C\frac{du(t)}{dt}$$

Where is $\frac{du(t)}{dt} = \frac{du(t)}{dt}$ the lowest derivative of $u(t)$. To get out $u(t)$, $\dot{u}(t)$ must be integrated once. (In the block diagram, $\dot{u}(t)$ is u_dot). $\frac{1}{s}$ is the integrator block.

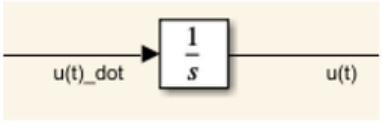


Figure 1: Integrating in the block diagram in simulink

Unfortunately, this is still meaningless, as the values of $u(t)$ are unknown since the values of$\dot{u}(t)$ are unknown. To see how $\dot{u}(t)$ is computed, the differential equation must be rearranged to solve in terms of $\dot{u}(t)$.

$$i(t) = \frac{1}{R}u(t) + C\frac{du(t)}{dt} \quad \Rightarrow \quad \frac{du(t)}{dt} = \frac{1}{C}\left(i(t) - \frac{1}{R}u(t)\right) \quad (1)$$

It can be seen that $\dot{u}(t)$ is the product of $\frac{1}{C}$ times some difference.

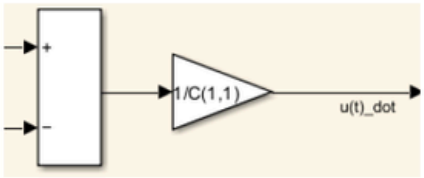


Figure 2: Multiplying a difference by gain $\frac{1}{C}$ to get $\dot{u}(t)$ in a block diagram

The difference is between $i(t)$, and $\frac{1}{R}u(t)$.

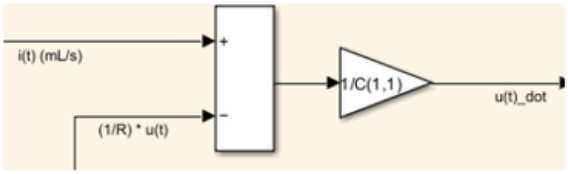


Figure 3: Seeing the feedback in a block diagram

Remember however, that $u(t)$ is the integral of $\dot{u}(t)$, so there is feedback in this block diagram. The output will feed into the inputs. Thus, the 2-element model is created

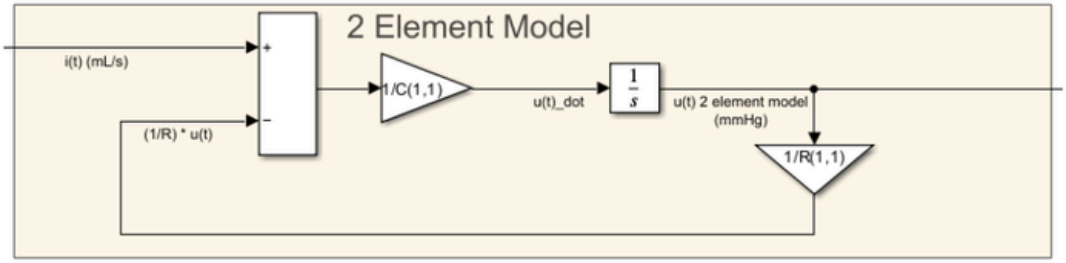
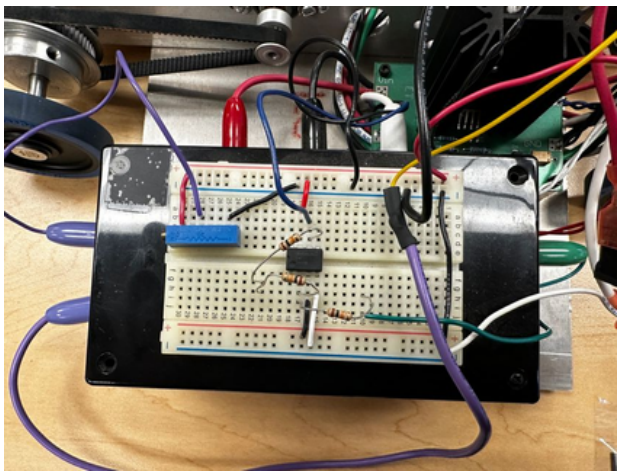


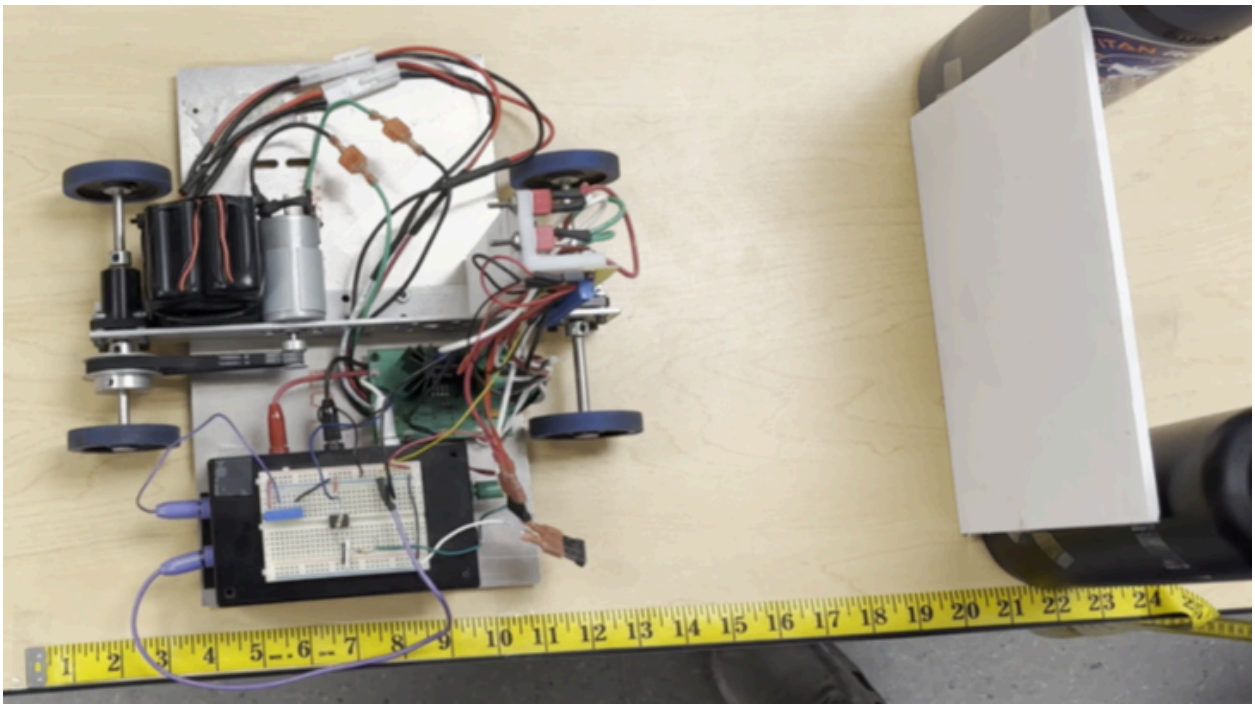
Figure 4: The full 2 − Element Windkessel Block Diagram

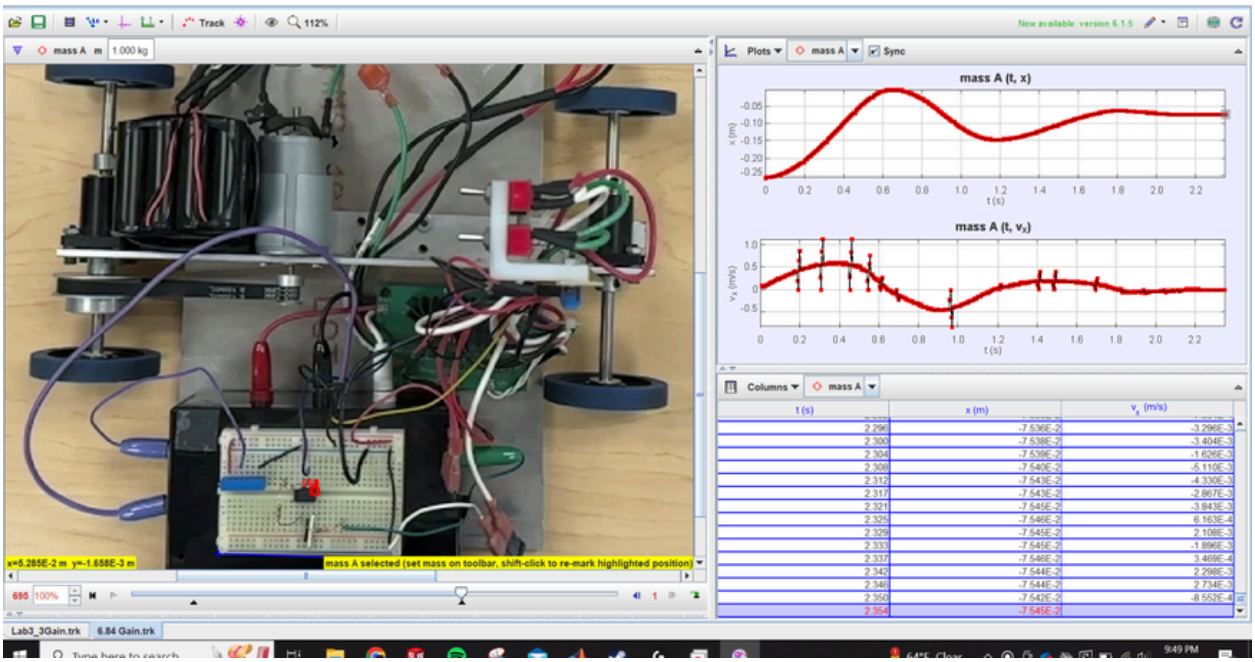
This was the main assignment in my Dynamics and Controls Systems lab. We were tasked with developing a PID Op-Amp controller to control the motor on a cart (shown to the right). We needed to get the cart to stop a certain distance from a white panel (as shown below), and we needed to use resistors and capacitors in a way (which controlled the P and D terms) so that an optimal response was reached: the car would be responsive enough (rise time) without having too much overshoot, and it would settle at an appropriate time to have the shortest possible total response time.



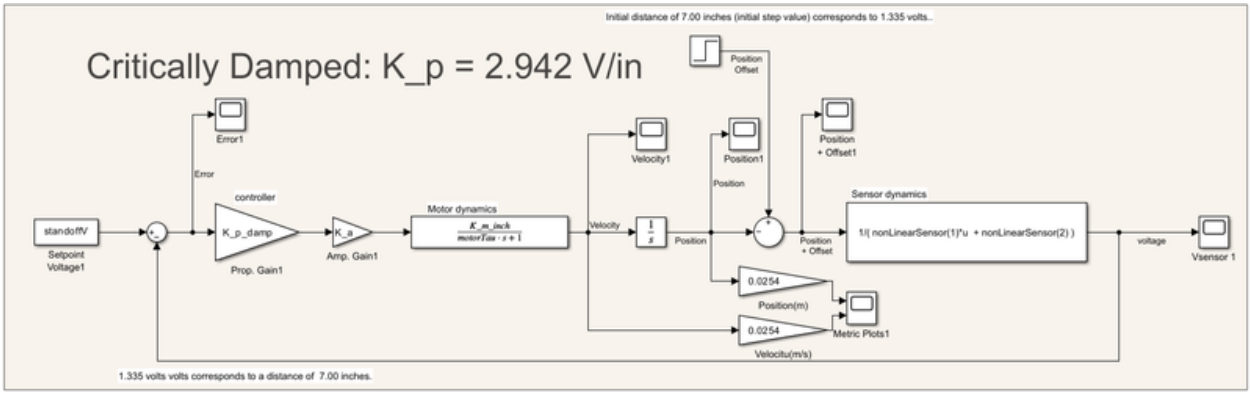
The cart gauged distance using an infrared sensor, and the motor's supply voltage was controlled using an Op-Amp, before being fed into the power electronics.

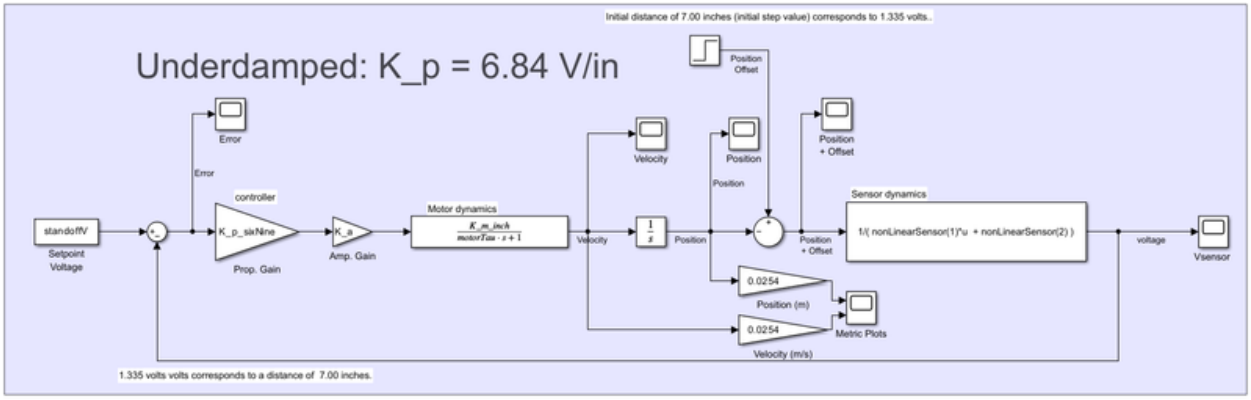
We tracked the cart's position in real time, and ran data analysis on it. We also used Simulink to tune the PD controller

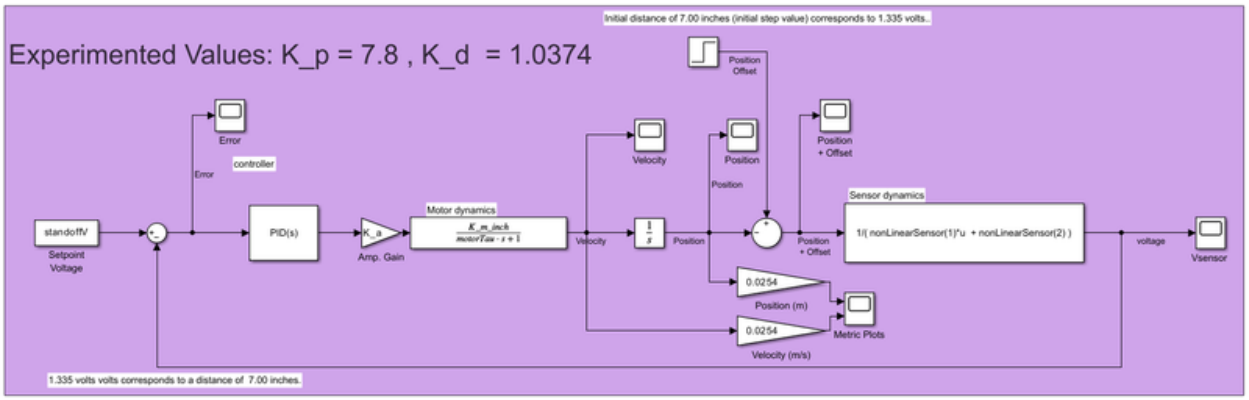


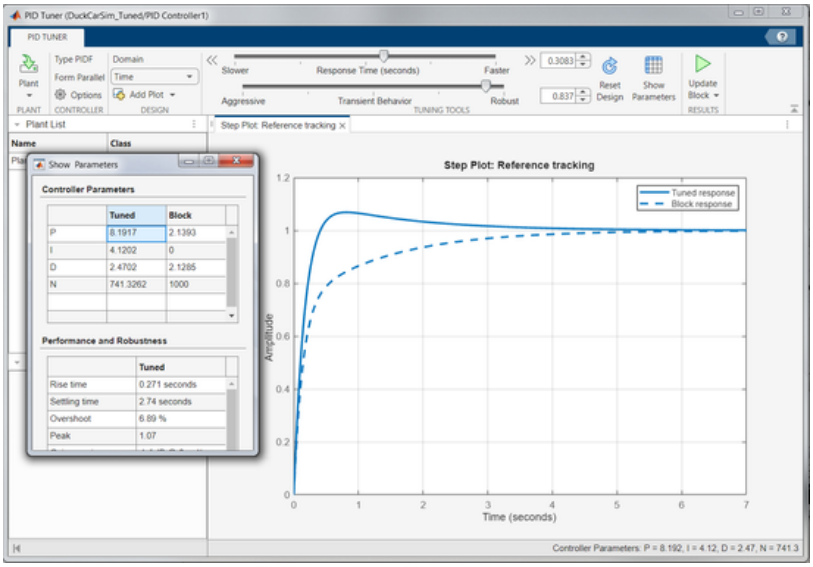


We were taught how to model electromechanical systems. We were able to neglect the electrical side of the motor, as its time constant was negligible in comparison to the motor's mechanical time constant. As shown, the motor is fed by the Op- Amp. Feeding into the op amp are the set point voltage, which is the voltage the IR sensor reads when it is at stand still from the panel (which is set via a potentiometer), and the actual read distance, which is outputted voltage from the IR sensor.









Tuning the PD controller was a matter of both using Simulink's PID Tuner, and experimenting in real life. Because the cart outputted distance, it had an integrator term, and therefore, was a type 1 system. This meant it had no steady state error, and integral control wasn't required.

So I had to manually set the I term to 0. I then used the P and D terms as launching points to determine the optimal P and D values, given the resistors and capacitors available. This was all very experimental.

Eventually, as shown on the next page, our system produced almost no overshoot, and settled very quickly because of it. It also had a fast rise time.

We then compared real life to our Simulink models. We calculated the damping coefficient and natural frequency both ways, using log-decrement, and also testing our linearized equations and plugging in constants. As seen, our Simulink model wasn't too close to real life. This may have been because the electrical side was non-negligible, especially the power electronics.

$$\zeta_{log-decrement} = -\frac{log(\frac{maxPercentOvershoot}{100})}{\sqrt{\pi^2+log(\frac{maxPercentOvershoot}{100})^2}} \qquad \zeta_{experimentally} = \frac{1}{2\tau\omega_n}$$

$$\omega_{n_{log-decrement}} = \frac{\pi}{(T)\sqrt{1-(\zeta_{log-decrement})^2}} \qquad \omega_{n_{experimentally}} = \sqrt{\frac{K_p K_a K_m K_s}{\tau}}$$

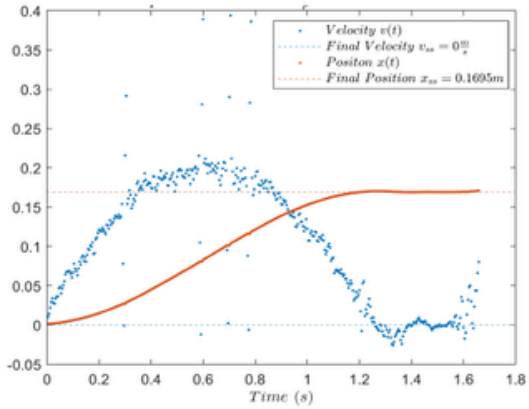


Figure 1: Actual Motion at $K_p = 2.942$

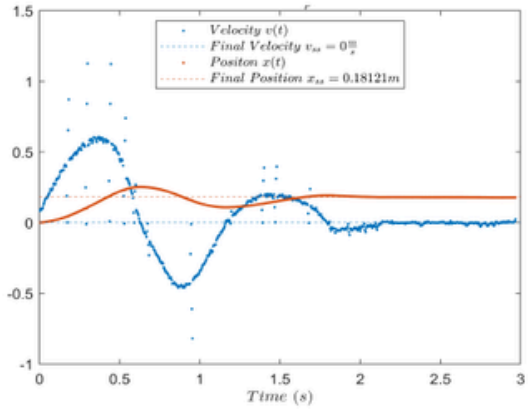


Figure 2: Actual Motion at $K_p = 6.84$

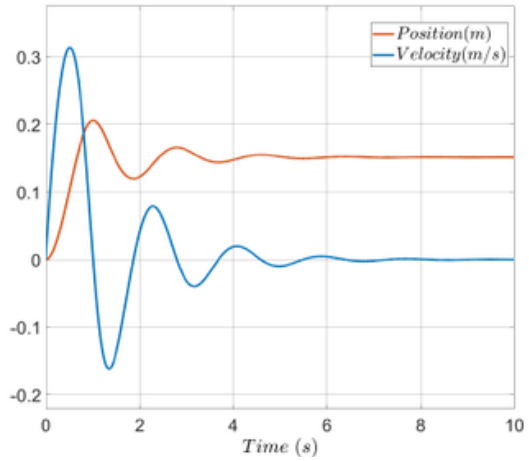


Figure 3: SIMULINK's Model at $K_p = 2.942$

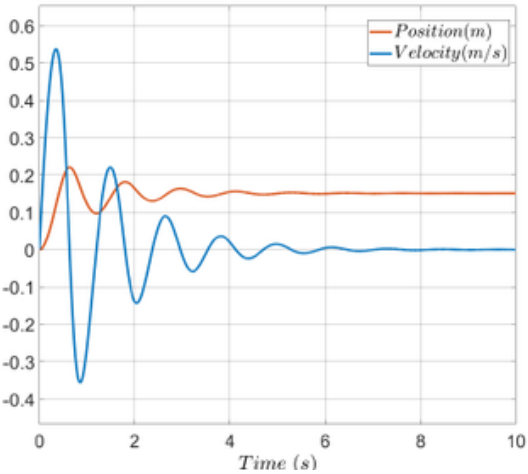


Figure 4: SIMULINK's Model at $K_p = 6.84$

My experimented and tuned PD values produced a very good result, even with the Simulink model being a little off

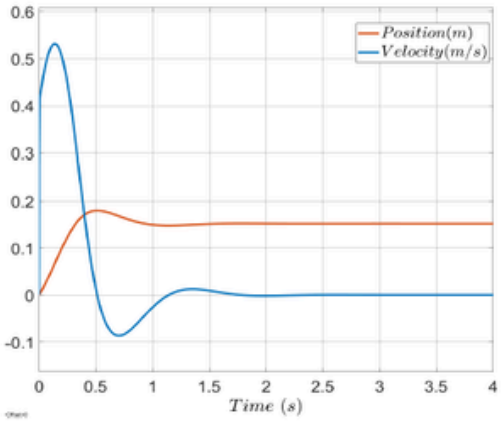


Figure 5: SIMULINK'S Model at $K_p = 7.8, K_d = 1.0374$

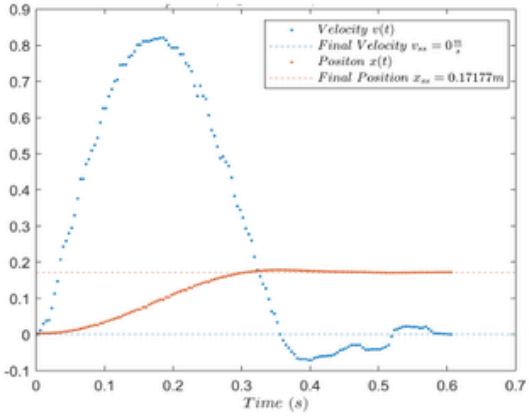
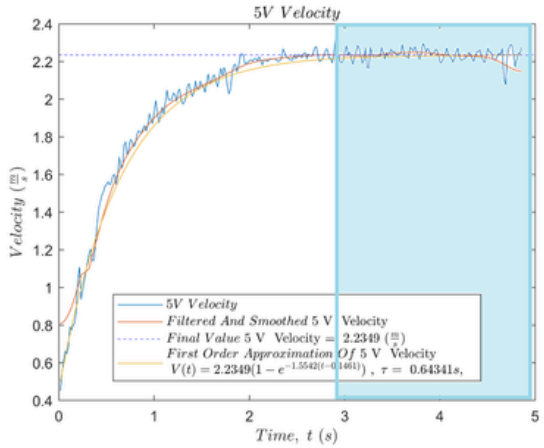


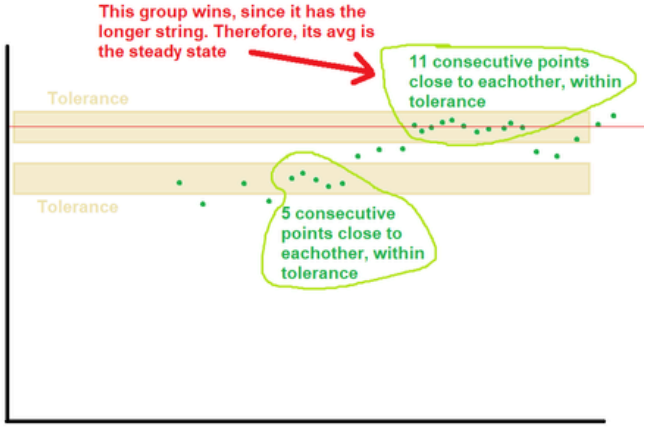
Figure 6: Actual Motion at $K_p = 7.8, K_d = 1.0374$

Because our tracking software could be a little buggy at times, it produced NaN positions for certain time points. This caused problems with the data, and the NaNs had to be removed

```
function [dataWithoutNaNs] = NaNRemover(dataInput)
    dataWithoutNaNs = dataInput; % can't pass by reference :(
    [a,~] = find(isnan(dataWithoutNaNs));
    a = unique(a(:).'); %removes any duplicate values, incase there is a NaN twice in one row
    for i = 1:length(a)
        dataWithoutNaNs(a(i),:) = [];
        a = a - 1;
    end
end
```

Manually finding the steady state position and velocity would have been subject to a lot of human bias, and calculation error. So that is why I had MATLAB find the steady state for me. Zooming in on a chosen window (usually the last third of the x axis), the program looks for a continuous string of values that are close to each other (they are given a tolerance). There could be many of these strings though, and so it looks for the longest consecutive string, and whatever those values are, they are averaged to output the steady state.

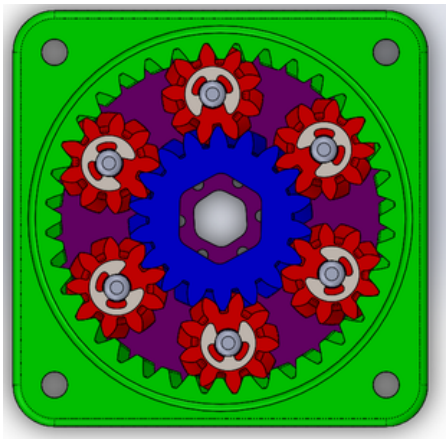





```
function [finalValue] = finalValueFinder(dataInputted,steadyStateTolerance,timeSection)
    %will only look at last set of time values for final value
    lastSectionOfTime = [floor( timeSection *height(dataInputted)),ceil( timeSection *height(dataInputted))];
    if lastSectionOfTime(1) == 0
        lastSectionOfTime(1) = 1;
    end
    a = diff(dataInputted(lastSectionOfTime(1):lastSectionOfTime(2),2))./diff(dataInputted(lastSectionOfTime(1):lastSectionOfTime(2),1));
    steadyestLastIndecies = find(abs(a)<=steadyStateTolerance) + lastSectionOfTime(1) - 1;
    %checks for any changes smaller than the tolerance, so that change is close to 0
    consecutiveSteadyIndecies = [;]; %  [ start1,start2 ; end1,end2 ; ... ]
    itt = 1;
    %will now check for the longest streak of consecutive indecies that fall within tolerance
    while itt <= length(steadyestLastIndecies)
        a = steadyestLastIndecies(itt);
        iitt = itt+1; %check for consecutive
        while iitt <= length(steadyestLastIndecies) && steadyestLastIndecies(iitt) - steadyestLastIndecies(itt) < 2
            iitt = iitt+1;
            itt = itt + 1;
        end
        b = steadyestLastIndecies(itt);
        consecutiveSteadyIndecies = [consecutiveSteadyIndecies ; [a,b]];
        itt = itt+1;
    end
    consecutiveSteadyIndecies;
    longestStreakPairs = find(max( consecutiveSteadyIndecies(:,2) - consecutiveSteadyIndecies(:,1) )); %might return multiple indecies if there is a tie
    longestStreakPairs = longestStreakPairs(end); %use the last longest steady streak
    finalValue = mean(dataInputted(consecutiveSteadyIndecies(longestStreakPairs,1):consecutiveSteadyIndecies(longestStreakPairs,1),2));
    %runs these indecies through the filtered function, and averages the values at these indecies
end
```

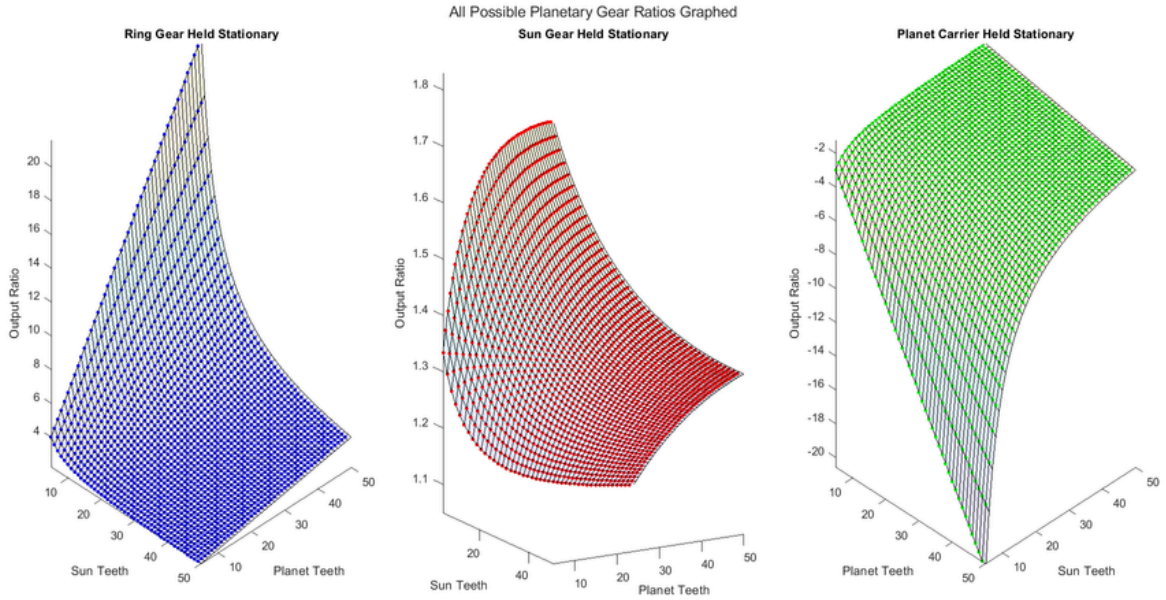
As seen, the data was also noise filtered to produce smoother curves, so that a more accurate equation of motion could be fit onto it.

```
function [filteredData] = dataNoiseRemover(dataIn,cutoffFreq)
    %Runs a low pass filter for each non-time column of the data, to filter
    %out nise
    dataIn = NaNRemover(dataIn);
    samplesCount = height(dataIn);
    filteredData = zeros(height(dataIn),width(dataIn));
    filteredData(:,1) = dataIn(:,1);%all time entries for position and velocity are the same
    for i = 2:width(dataIn)
        filteredData(:,i) = lowpass(dataIn(:,i),cutoffFreq,samplesCount);
    end
end
```



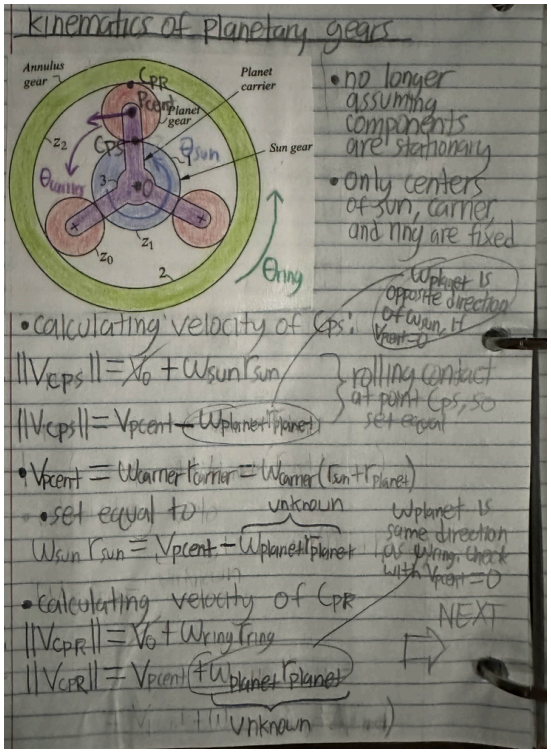
For my Gearshift transmission, before on deciding what ratios to even use, I had to see what ratios were possible out of a planetary gear set. This was for space constraints, and I also wanted to see if a planetary gear ratio, holding the carrier, could have an output ratio of 2:1. As shown below, it is impossible (the blue graph). Although it does asymptote towards it as the sun gear gets much much larger than the planet gear. The graph is plotted as number of teeth of each gear (the sun and planet are usually the gears which you design first) with the z axis being the output ratio.

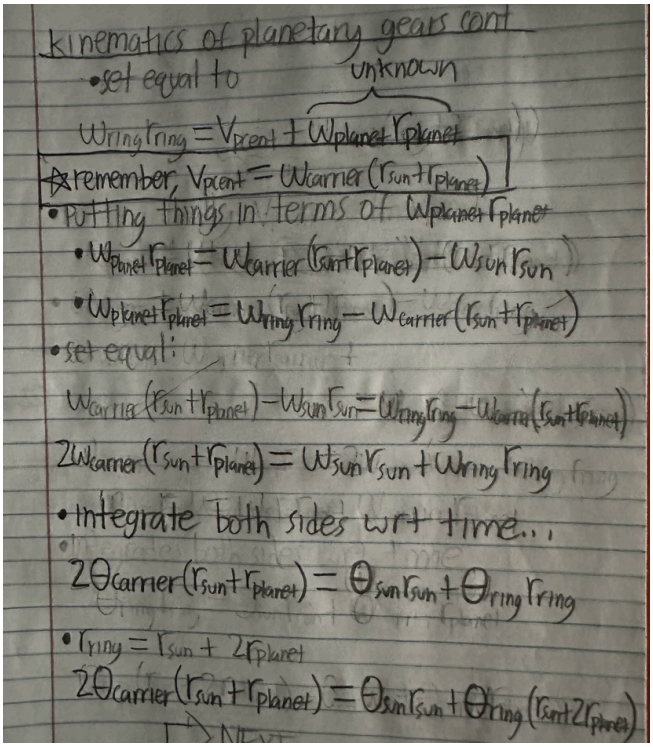
I was able to find the equation of a planetary gear set online when I first made this in my freshman year of college, but it wasn't until my junior year, when I took dynamics the previous semester, that I was able to derive the equations on my own. They are color coded for easy reference. Z is the number of teeth per gear, and although the derived equations use r for radius, if the gears actually mesh together (same module), the the teeth is proportional to the radius, and that constant is the same for all mating gears, meaning it cancels out.



$$2\,\theta_{carrier}(z_{sun}+z_{planet}) = \theta_{sun}z_{sun} + \theta_{ring}(z_{sun}+2z_{planet})$$

$$2\,\omega_{carrier}(z_{sun}+z_{planet}) = \omega_{sun}z_{sun} + \omega_{ring}(z_{sun}+2z_{planet})$$





I had just finished my semester of Integral Calculus, and I really loved the class. While working at Mathnasium, during the lull hours, I had come to a realization about the power rule: As shown below, repeatedly taking the derivative of a polynomial using the power rule resulted in some sort of factorial pattern.

Remembering that an integral is just a “negative count” derivative, I realized that this pattern could go the other way, assuming each time I integrated, the extra constant C = 0, so that it did not carry over into the next integration.

This led me to an interesting question: What if instead of having whole derivatives and integrals - what if instead of taking the 1st derivative of a function, what if I took the 0.5th derivative? All I had to do was plug in 0.5 for n, and graph. Realizing it was a smooth graph, I had the idea of “connecting” the 1st derivative to 0.5th derivative, then to the original function by plotting this in 3D space.

Graphed below, each “pane” is some derivative of the function, 0th being the original function. I interpolated values in between each pane, to see what the “derivatives in between” looked like for the function f(x) = x.

$$f(x) = x^4$$

$$\frac{d}{dx} f(x) = 4x^3$$

$$\frac{d^2}{dx^2} f(x) = 4 \cdot 3 \cdot x^2$$

$$\frac{d^3}{dx^3} f(x) = 4 \cdot 3 \cdot 2 \cdot x^1$$

$$\frac{d^n}{dx^n} f(x) = \frac{4!}{(4-n)!} x^{(4-n)}$$

$$\Rightarrow$$

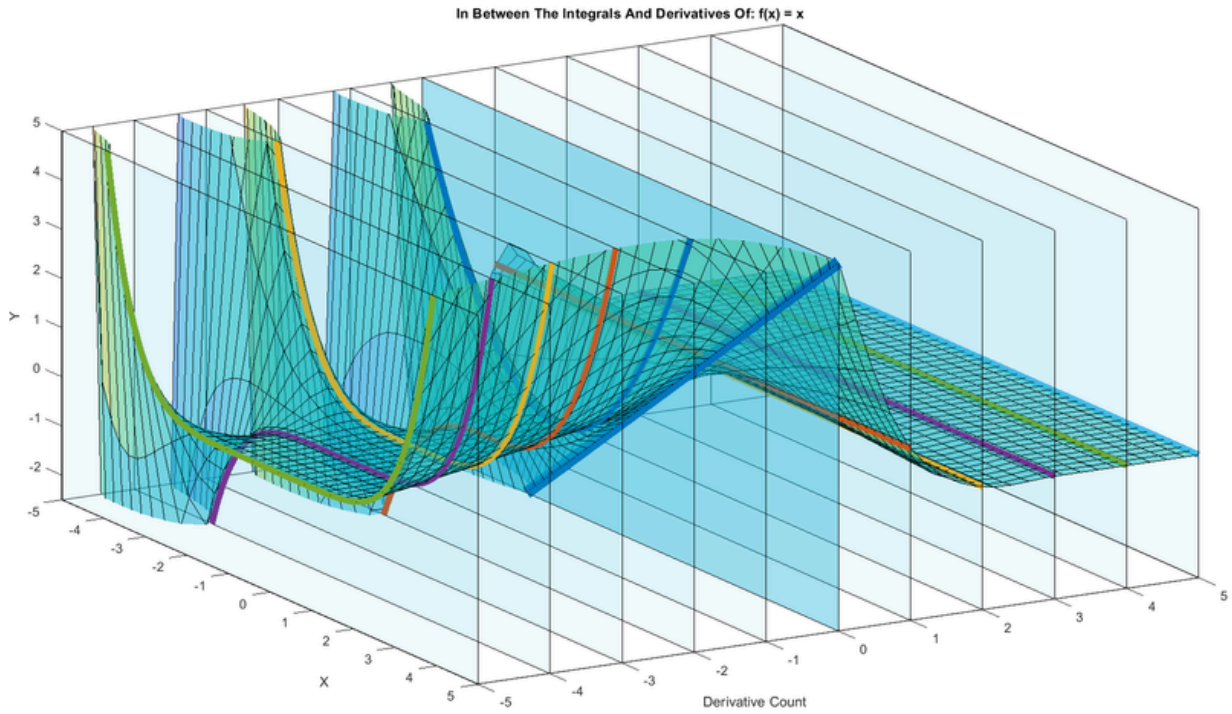
$$f(x) = ax^b$$

$$\frac{d}{dx} f(x) = a \cdot bx^{b-1}$$

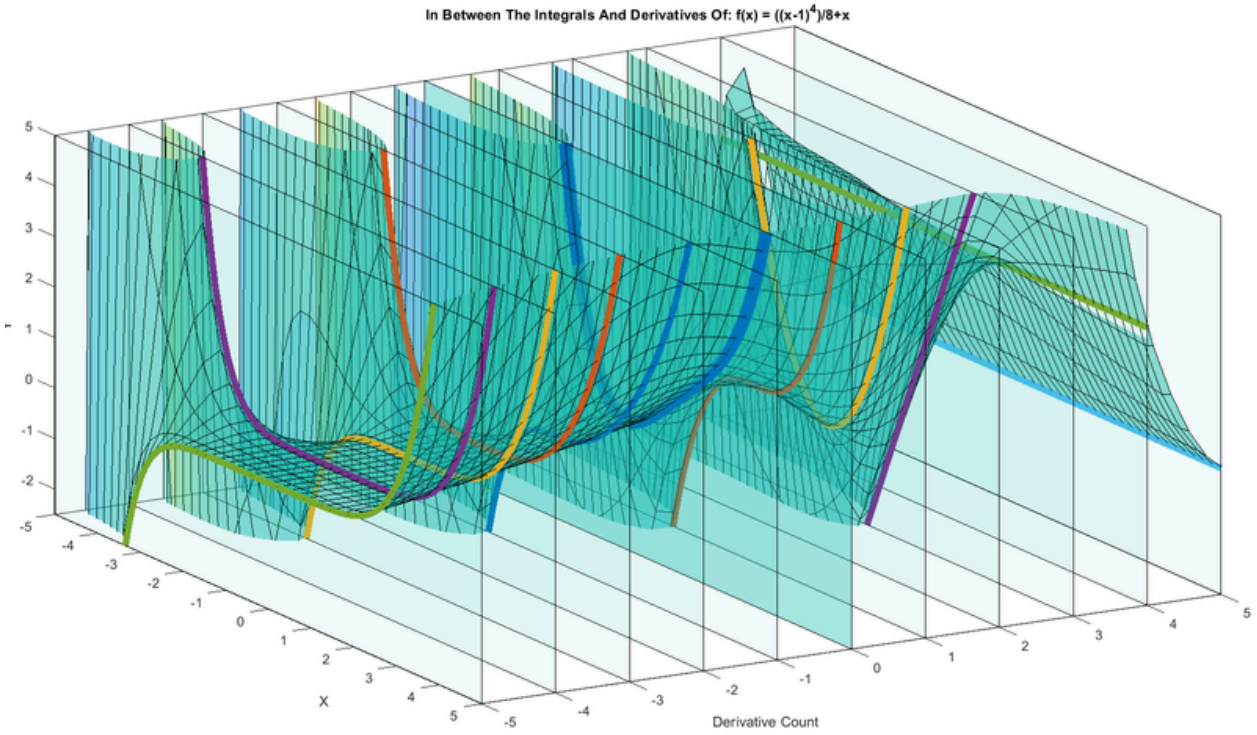
$$\frac{d^2}{dx^2} f(x) = a \cdot b(b-1)x^{b-2}$$

$$\frac{d^3}{dx^3} f(x) = a \cdot b(b-1)(b-2)x^{b-3}$$

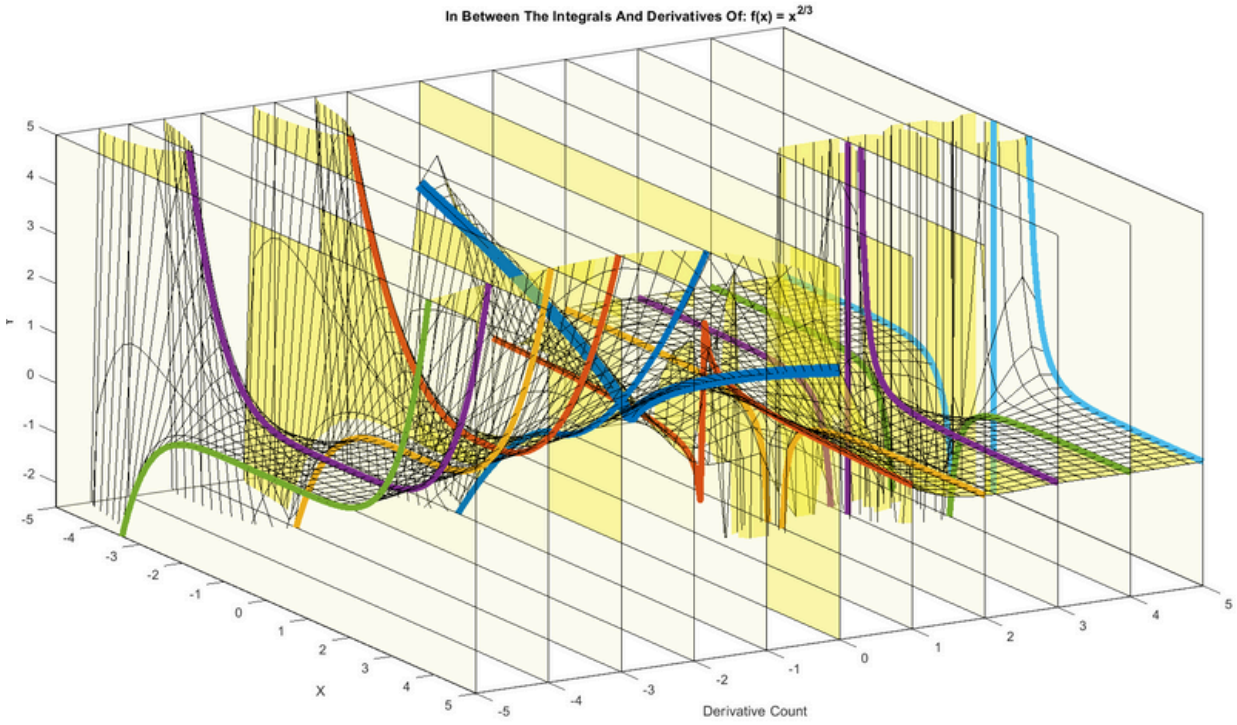
$$\frac{d^n}{dx^n} f(x) = a \frac{b!}{(b-n)!} x^{(b-n)}$$



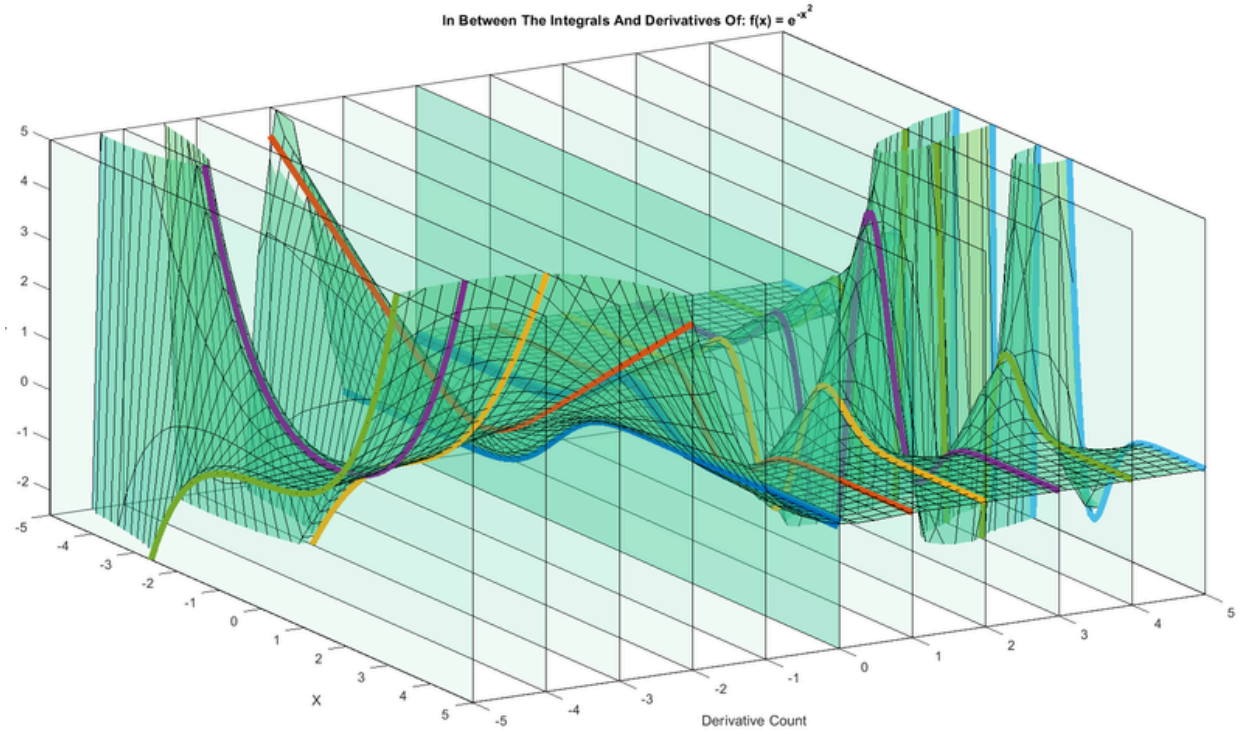
Seeing that my graph worked for f(x)=x, I wanted to test a more complex polynomial. I went with a quartic.



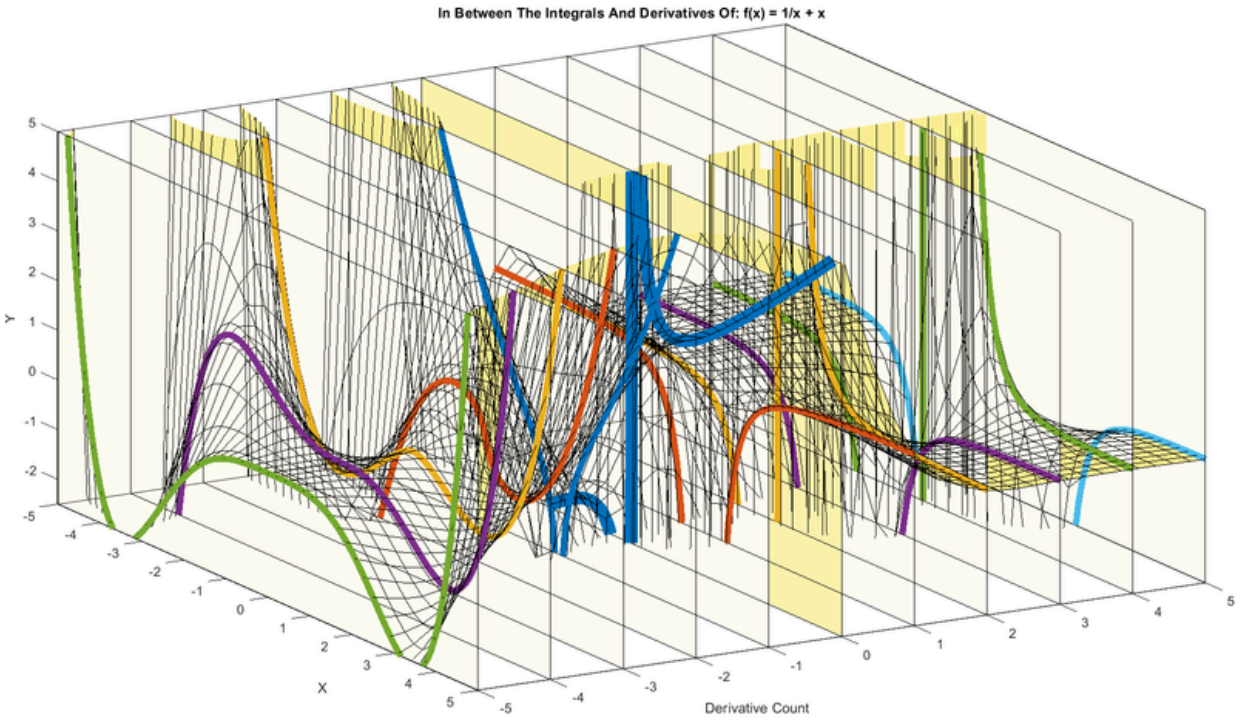
Wanting to see how a fractional power affected the power rule and inbetweens, I graphed f(x)=x^(2/3)



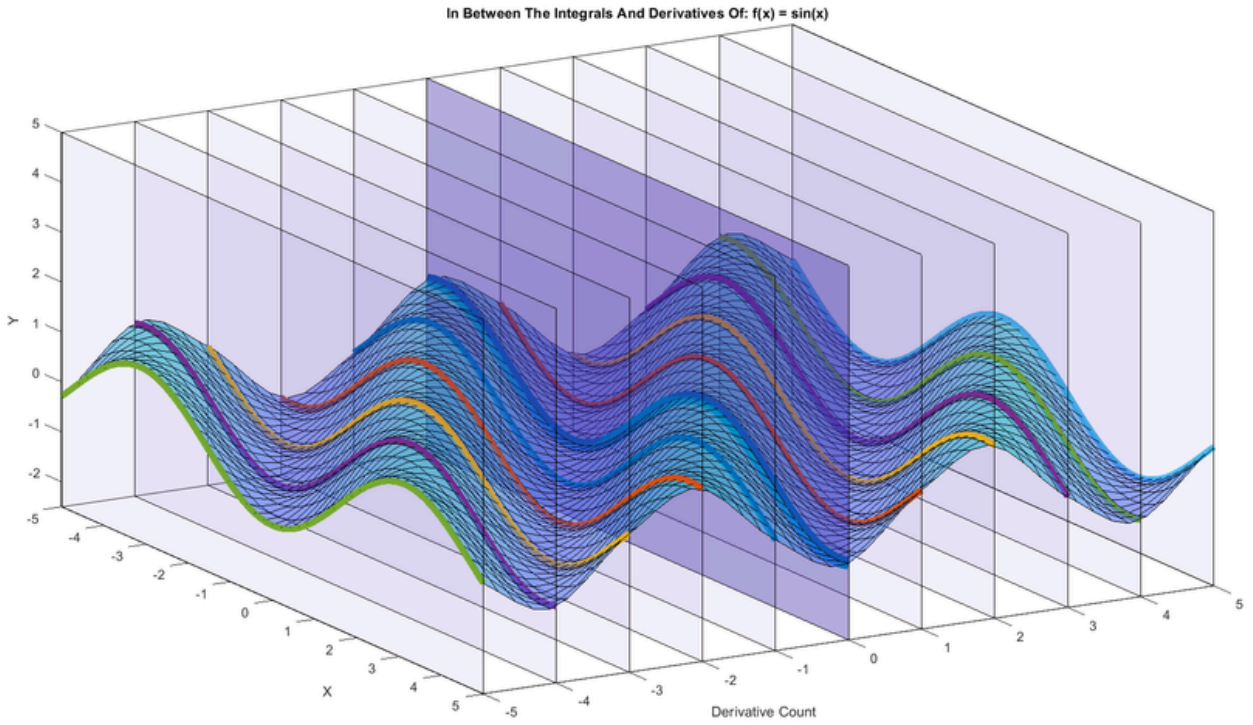
Not wanting to limit myself to polynomials, I tested out not only an exponential, but an analytically unintegrable function: the gaussian curve.



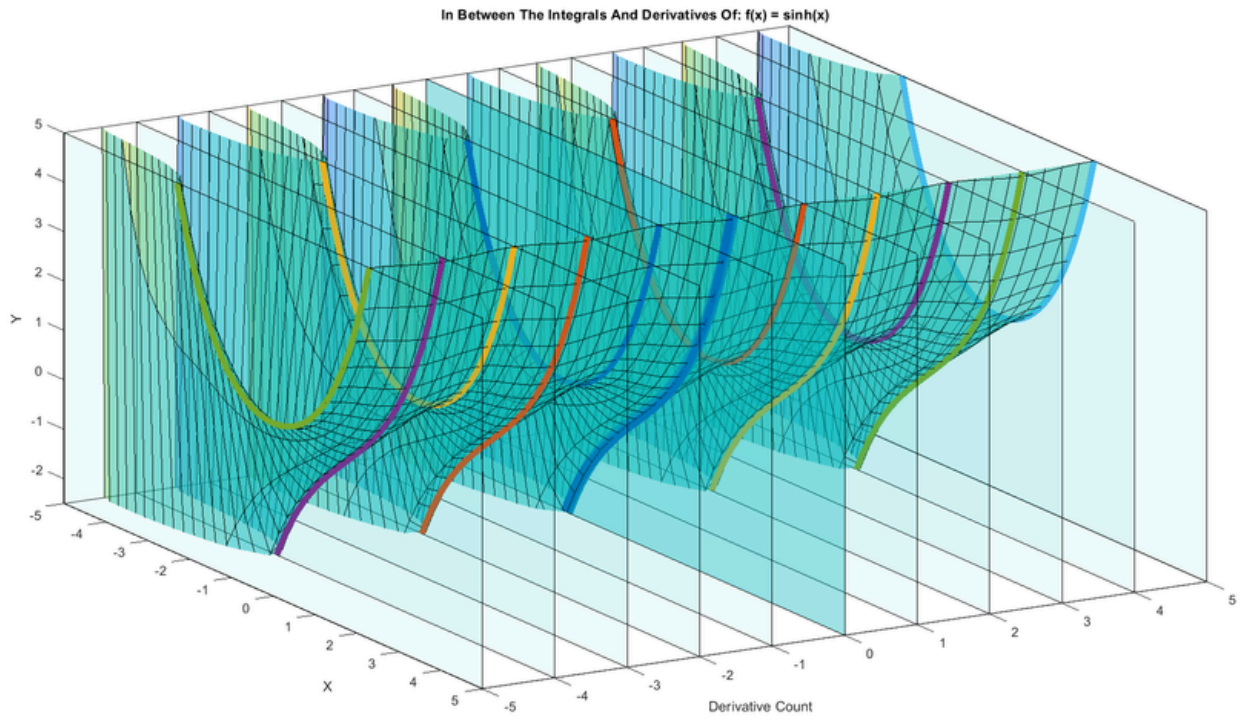
I even tried rational functions!



Knowing that the derivatives and integrals of sin(x) were cyclical, I tested it out. You can see that the “inbetween” derivatives and integrals are just the sine wave shifted over, and no special asymptotes or geometries exist.



Keeping with the cyclical patterns, I decided to try a hypoerbolic trig function.



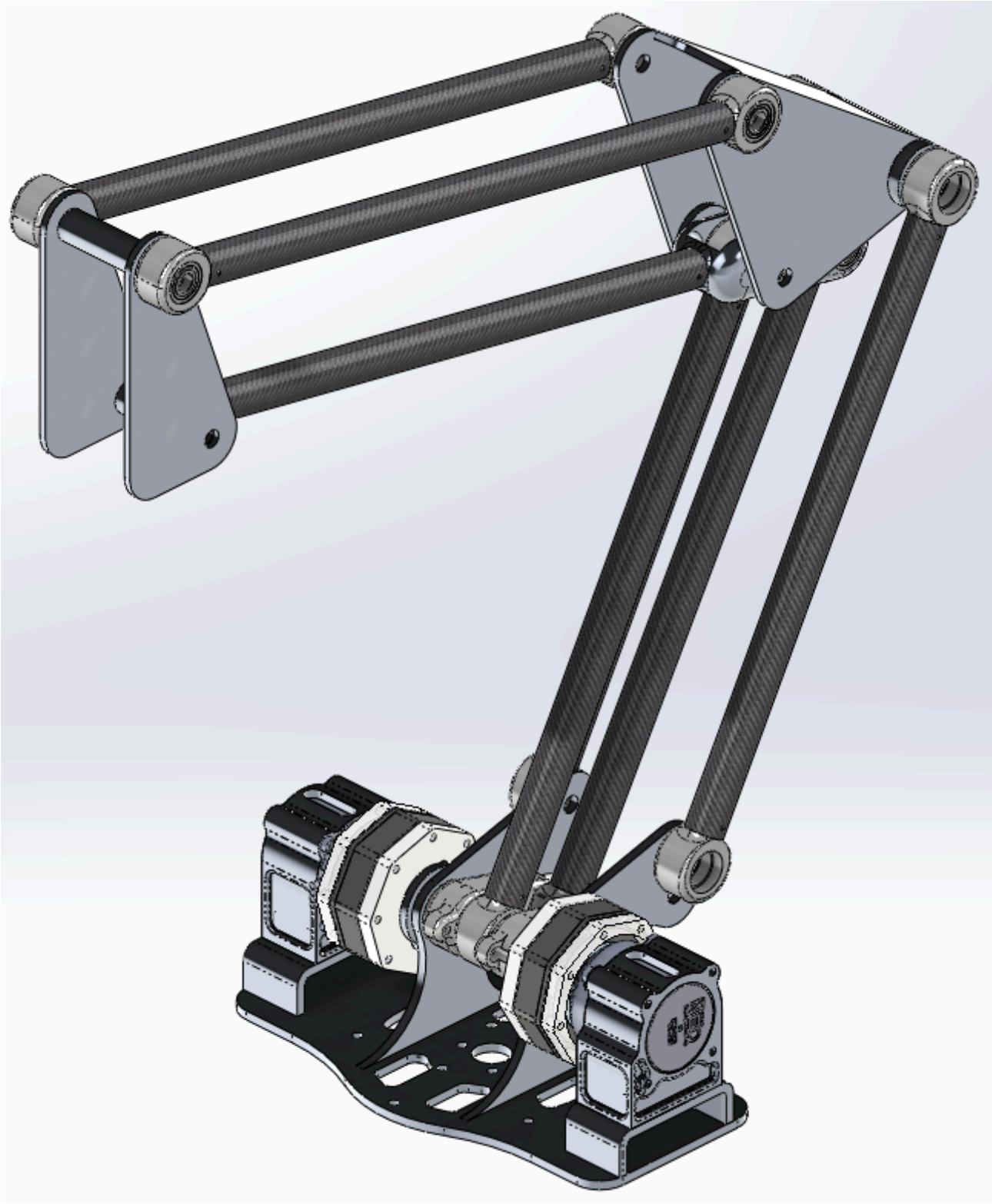
More functions can be found on the Github page, by clicking the links in the titles

ROVER PANTOGRAPH ARM — SEPTEMBER 2023 - MARCH 2024

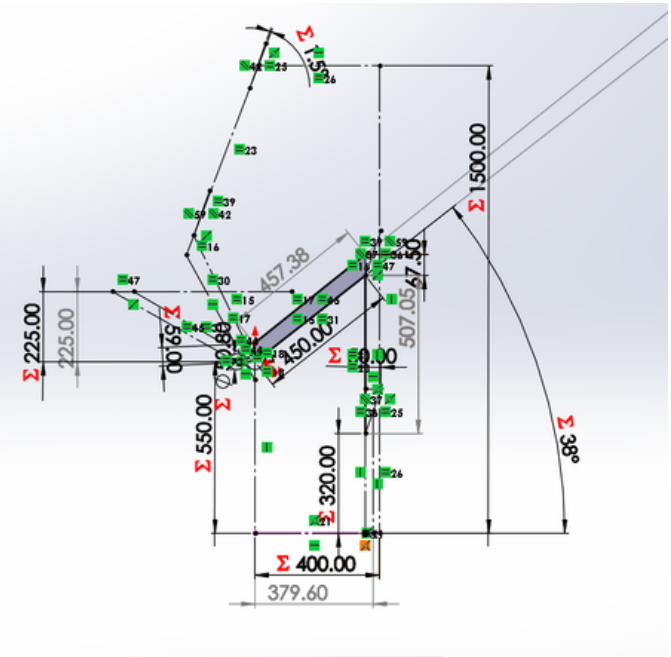
I was involved with Titan Rover for my senior design project, and I was in charge, and worked completely alone on the robotic arm. The design took several iterations, and had strict requirements. Although I didn't have time to program the arm due to being stuck in the machine shop, and dealing with my other academic classes, I still believe I outputted a good product, and the analysis that went into it is invaluable, and applicable to other projects.

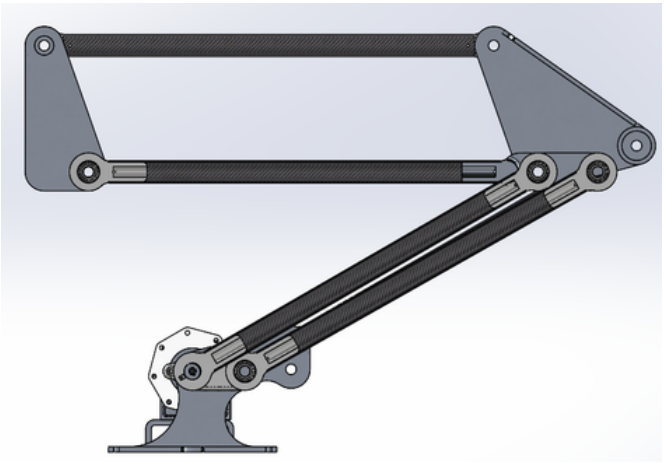
This project primarily focused on 4-bar linkages, and their mechanical advantage. But it also tested my spatial ability, when I tried to balance the arm's range, and robustness, while also trying to get the arm to not run into itself. It also tested my CAD skills, because I ad to design things as if they could be manufactured, and not just 3D printed.

The next few pages will go over in much more detail, all the intricacies of the arm, and the analysis that went into it.

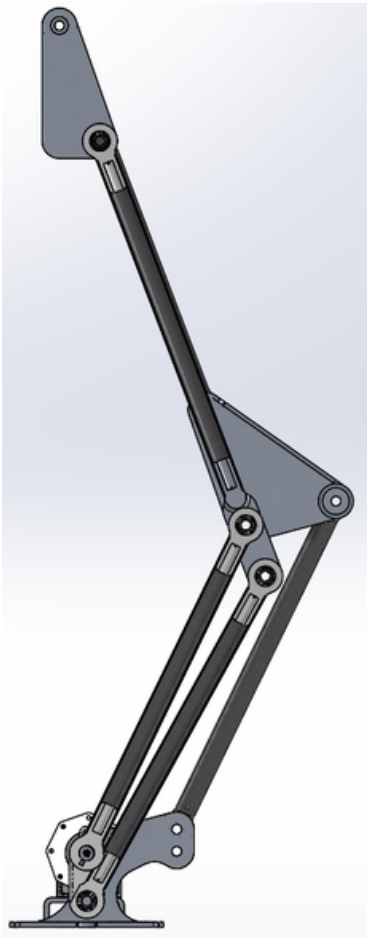


The arm had 3 main requirements: It had to be able to easily reach the ground, It had to easily be able to reach 1.5m up from the ground, and it had to fold up when not in use (especially when the rover itself was moving). Those geometric constraints are shown in the sketch below. All subsequent parts and assemblies were defined around that sketch.

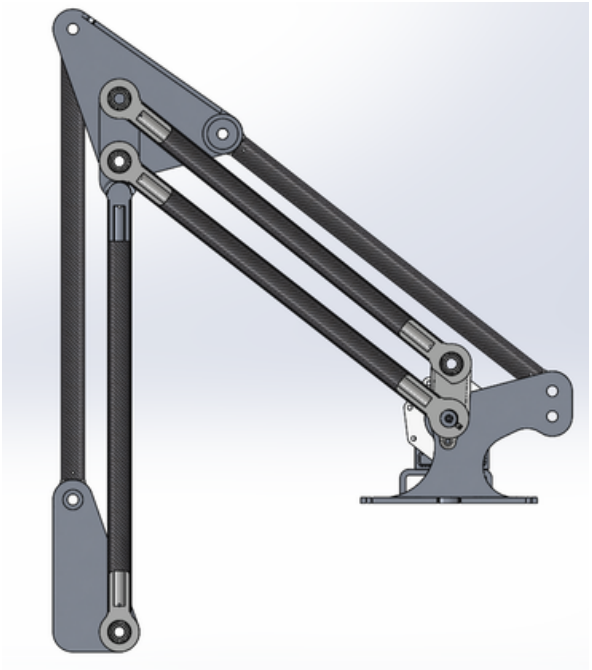




The arm folded up. This moves the center of gravity lower, and closer to the middle of the rover, which would help against tipping over



The arm reaching all the way up. The wrist would have covered the horizontal distance.



The arm reaching down. The wrist would have covered the rest of the distance to the ground. As will be shown later, the wrist's orientation won't be affected by the orientation of the other linkages.

Notice that tall these arms are in a cut plane view. Take a while to appreciate that the linkages do not run into each other when in these positions. They also never run into each other transitioning between positions.

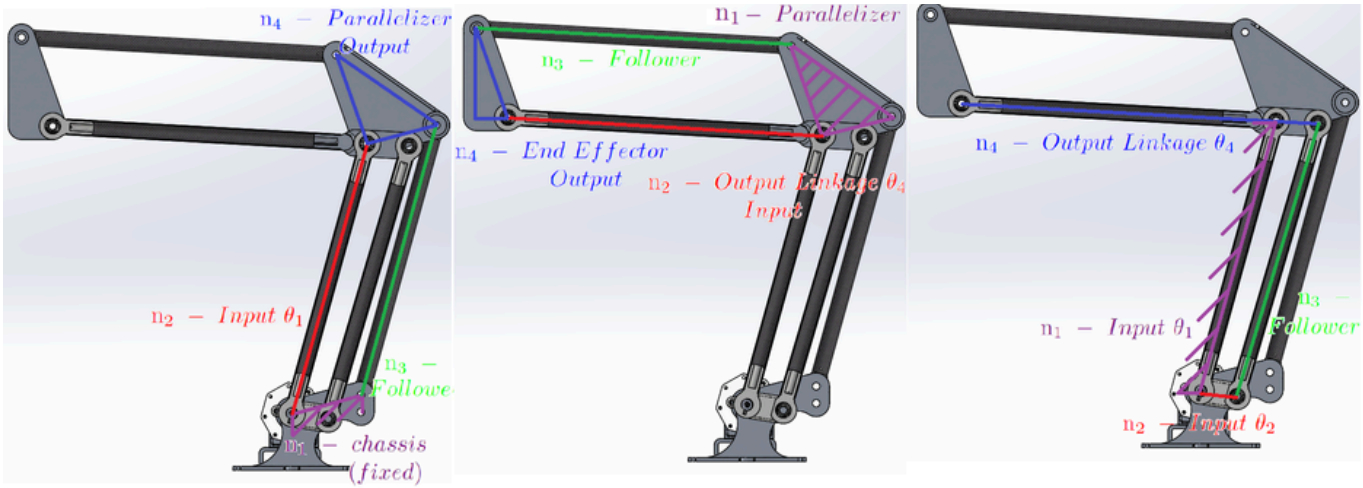
The arm is made of three different 4 bar mechanisms. Shown in the first and second pictures are the parallel arm mechanisms. As shown in the previous page, no matter how the linkages are moved or oriented, the output triangle (at the very tip) always stays parallel to the ground. This is useful, because the wrist was supposed to be mounted onto it. This meant that the wrist's orientation would not be influenced in any way by the position and orientation of the other linkages.

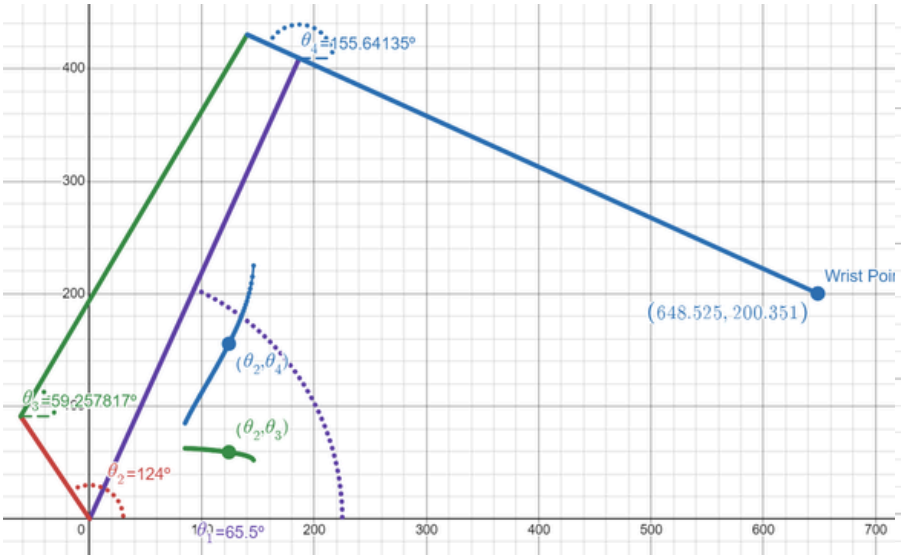
The first picture has a motor that powers theta1. This motor controls the "shoulder" of the arm

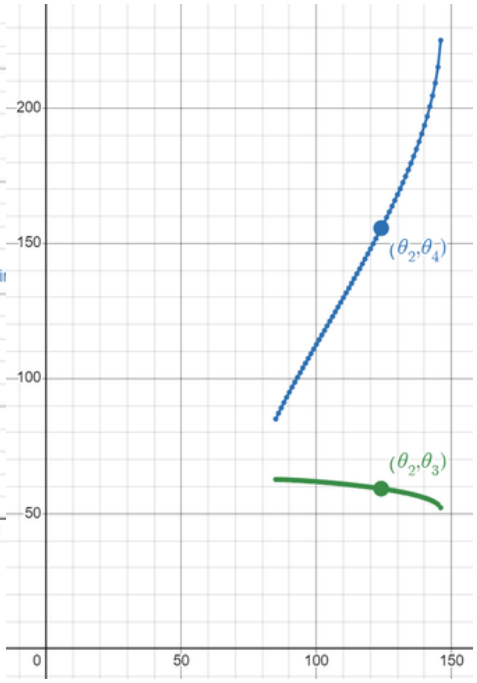
The third picture shows how the "elbow" of the arm is actuated. Instead of having a motor mounted onto the elbow itself, which would cause a heavy mass to hang and swing at a long distance, the vibrations of the motor would have severely amplified the moment put on the arm. This is why the elbow's motor is also placed at the shoulder, at the exact level of the first motor. The elbow motor drives theta2, and through a 4 bar linkage, actuates and changes the angle of theta4.

In addition to the range, compactness, and other geometric requirements, I had to consider the mechanical advantage that my 4-bar linkage would produce. While I could theoretically reach anywhere with the longest possible arms, I would also be sacrificing a lot of mechanical advantage, possibly negating any of the precious torque the motors powering it could provide. The nature of it being an arm meant that I was already carrying heavy loads out at a long distance from a pivot, already heavily fighting the torque of the motor.

This is why I created a graphing calculator to analyze the 4 bar kinematics (of the third picture) https://www.desmos.com/calculator/s3wx35sbtg

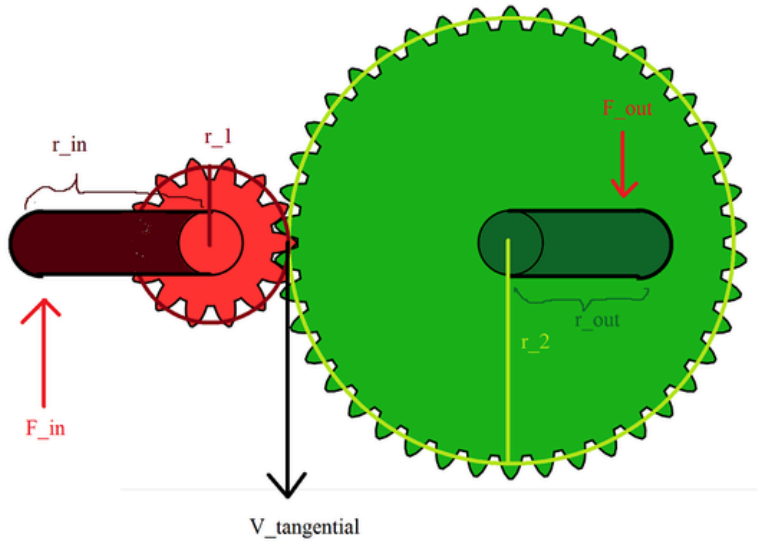






The arm had inputs of torque, but the load it dealt with was a force. Although mechanical advantage is typically used to describe gears and 4-bar mechanisms, they are usually ratios of input force to output force, or input torque to output torque. There is not measurement or term for input torque to output force.

This is why I created my own Mechanical Advantage_forceToTorque number. But to understand it, I want to show how it is derived by first showing how mechanical advantage is derived, using the example of a simple gear pair.



Starting off with the definition of Mechanical Advantage (MA):

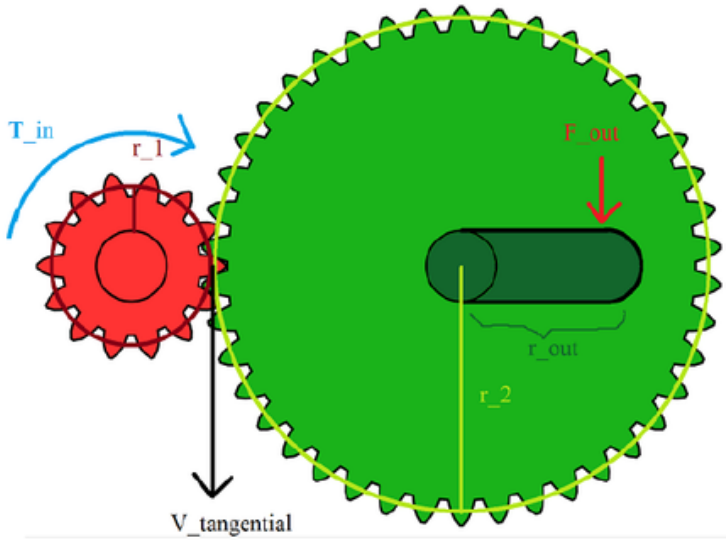
- $MA = \frac{F_{out}}{F_{in}}$
- $MA = \frac{F_{out} \times r_{out}}{F_{in} \times r_{in}} \left(\frac{r_{in}}{r_{out}}\right) = \frac{T_{out}}{T_{in}} \left(\frac{r_{in}}{r_{out}}\right)$

Assuming no losses of power, the power in should equal the power out. Then substitute into MA

- $P_{in} = P_{out} \Rightarrow T_{in}\omega_{in} = -T_{out}\omega_{out} \Rightarrow \frac{T_{out}}{T_{in}} = -\frac{\omega_{in}}{\omega_{out}}$
- $MA = -\frac{\omega_{in}}{\omega_{out}} \left(\frac{r_{in}}{r_{out}}\right)$

Knowing that tangential velocity is equal to angular velocity cross radius (assuming a constant radius, which spur gears don't change radius), and knowing that tangential velocity at the pitch radii of both gears must be the same...

- $V_{tangential} = r_1\omega_{in}, V_{tangential} = -r_2\omega_{out} \Rightarrow -\frac{\omega_{in}}{\omega_{out}} = \frac{r_2}{r_1}$
- $MA = \left(\frac{r_2}{r_1}\right)\left(\frac{r_{in}}{r_{out}}\right)$



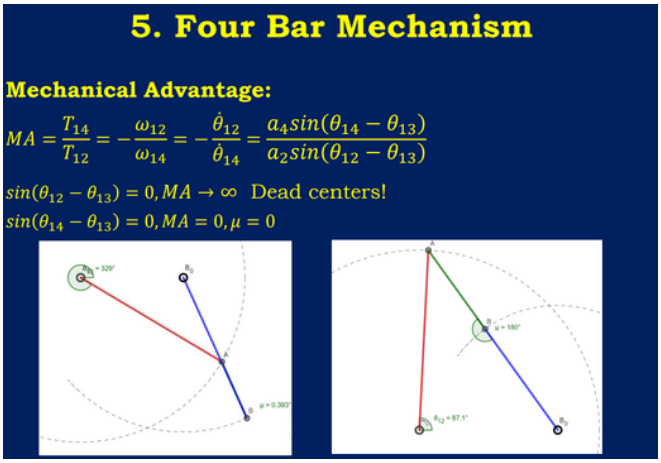
Now, instead of having an input force, we can describe T_in as F_in x r_in, and have an input torque. In order to keep the mathematical ratio of F_out to T_in, if we turn F_out into T_out by multiplying it by r_out, we must also divide by r_out.

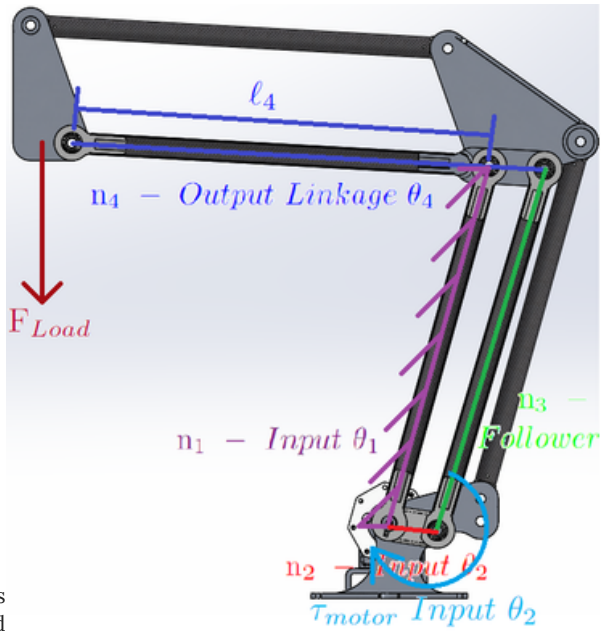
Eventually, the ratio of input torque to output force can be described purely with the geometry of the system: the radii of the gears, compared with the radius of the output lever.

$$MA_{force-to-torque} = \frac{F_{out}}{T_{in}} = \frac{F_{out} \times r_{out}}{T_{in}} \left(\frac{1}{r_{out}}\right) = \frac{T_{out}}{T_{in}} \left(\frac{1}{r_{out}}\right) = -\frac{\omega_{in}}{\omega_{out}} \left(\frac{1}{r_{out}}\right) = \left(\frac{r_2}{r_1}\right)\left(\frac{1}{r_{out}}\right)$$

The concept can now be expanded to the more complex 4-bar mechanism. The input torque, T_motor, which controls theta_2 comes from the elbow motor. The output force is F_Load, which will come from whatever the arm is carrying. The MA_torqueToForce will be the ratio of T_motor to F_Load.

Also shown below is a velocity analysis of any 4-bar linkage. Again, it is defined by the system's geometry. This time however, it is also defined by the system's position itself.





$$-\frac{\omega_{1,2}}{\omega_{1,4}} = \frac{r_4 sin(\theta_{1,4} - \theta_{1,3})}{r_2 sin(\theta_{1,2} - \theta_{1,3})}$$

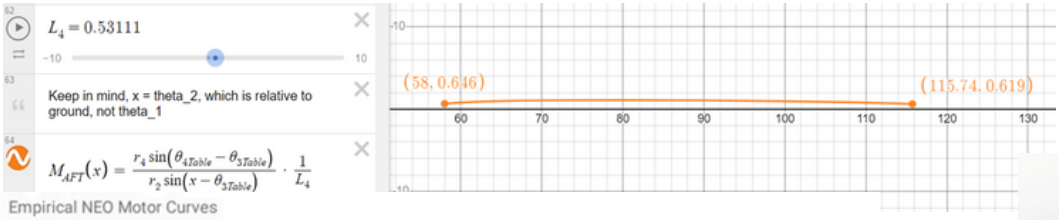
Shown below is the calculation for torque to force. The same logic applies as last time. In the end, the mechanical advantage depends on the actual speed of the linkages themselves. Plugging in the velocity analysis from above, MA is clearly and easily defined.

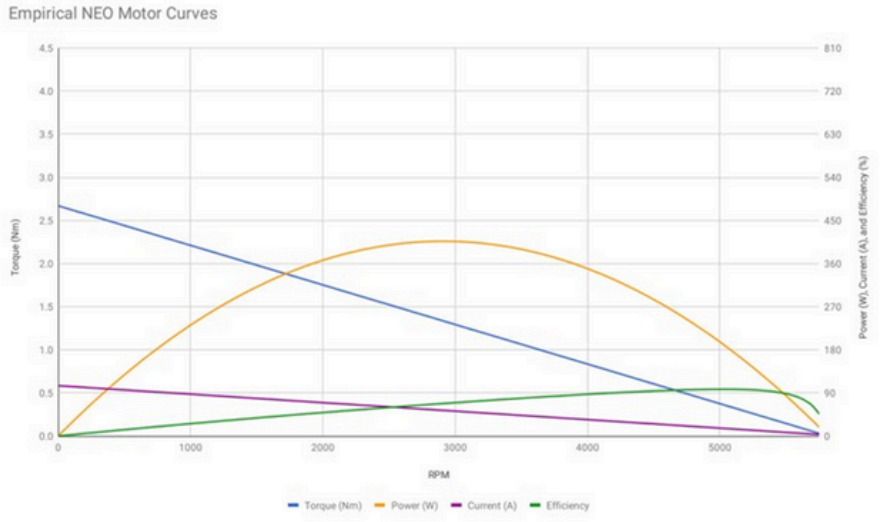
It can be seen that the MA changes as the position of the arm changes. The graphing calculator also keeps track of that,

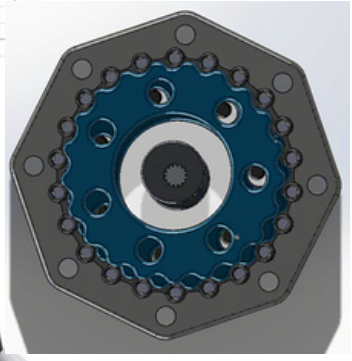
$$MA_{force-to-torque} = \frac{F_{load}}{T_{in}} = \frac{F_{load} \times \ell_4}{T_{in}}\left(\frac{1}{\ell_4}\right) = \frac{T_{out}}{T_{in}}\left(\frac{1}{\ell_4}\right) = -\frac{\omega_{in}}{\omega_{out}}\left(\frac{1}{\ell_4}\right) = -\frac{\omega_{1,2}}{\omega_{1,4}}\left(\frac{1}{\ell_4}\right)$$

$$MA_{force-to-torque} = \frac{r_4 sin(\theta_{1,4} - \theta_{1,3})}{r_2 sin(\theta_{1,2} - \theta_{1,3})}\left(\frac{1}{\ell_4}\right)$$

This became useful when sizing the gearbox around the motors.

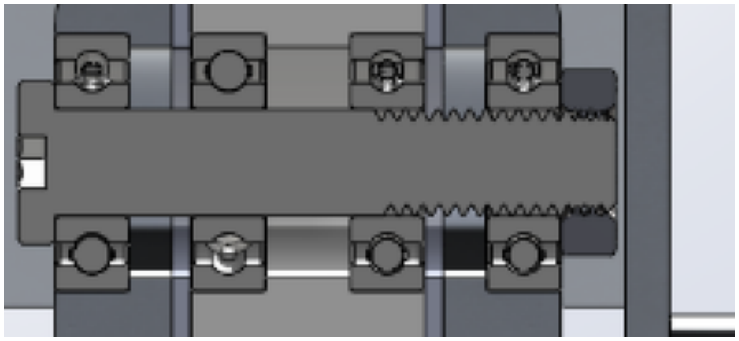






ROVER PANTOGRAPH ARM — SCREWS AS AXLES AND BEARING CREATIVITY

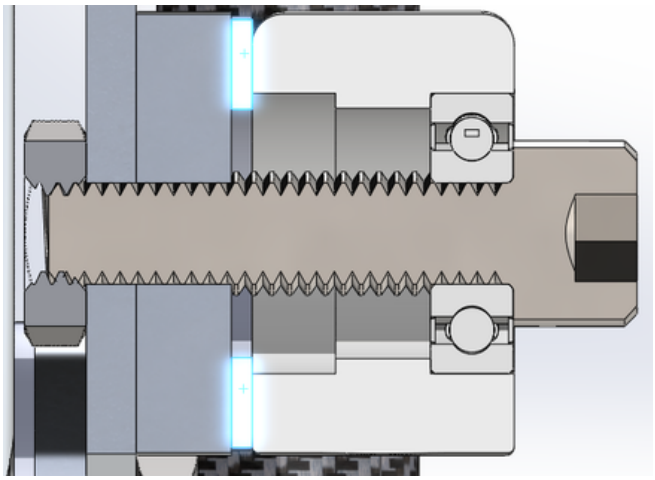
I thought it would be efficient to use heavy M12 screws as both a way to secure parts together, as well as an axle. Normally, tightening something would make it resistant to movement, and would add heavy friction. However, I did some creative things to get around this.

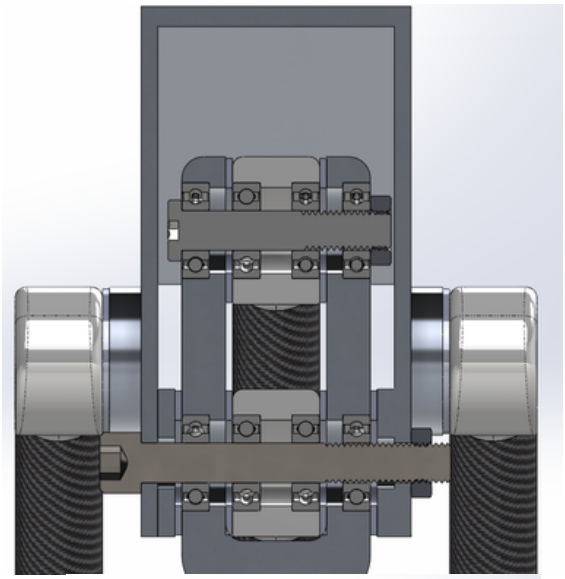


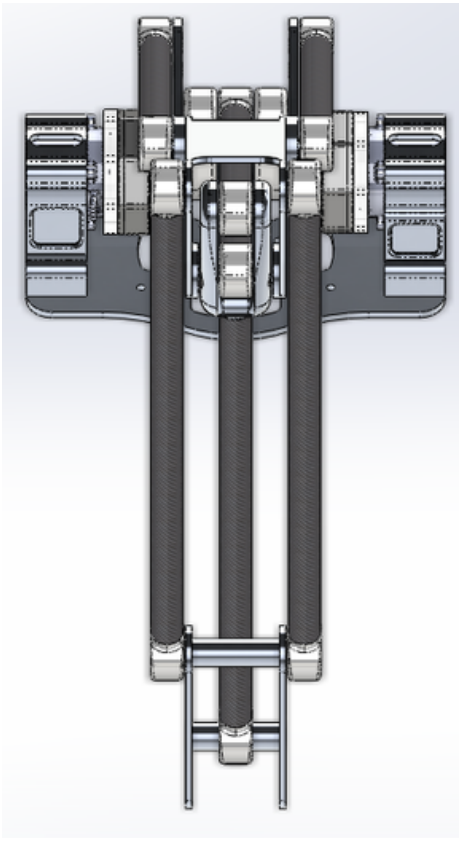
As can be seen to the left, the head of the screw, and the nut only touch the inner race of the bearing, while the outer race of the bearing is firmly embedded into the linkage. The bearings are prevented from sliding inwardly die to the "shelves", and are kept in place from sliding outwardly via the screw and nut. This therefore also keeps the linkages in place.

Think of it this way: The screw and nut are so tight, that they prevent the inner race from moving. However, the outer race is still free to move. And because it is embedded into the piece, the piece is also able to freely move around the bearing.

In cases where linkages and joints must "rub" on each other, I was still able to tighten them together axially, while allowing them to rotate freely and easily. Highlighted in blue are thin thrust needle bearings. Even if the two faces that rub on the bearing move at different speeds, there will still be smooth motion, as the needles will roll in between them with (negligible) slipping. (Negligible because they are cylindrical bearings, instead of cone shaped, meaning the radius of the rolling circle doesn't match the radius it is rolling around the screw).







Shown to the left are screenshots of the arm's linkages not running into each other, and having ample freedom and space to move, while still being compact and organized.

SOLVING A NONLINEAR DIFFERENTIAL EQUATION — APRIL 2021

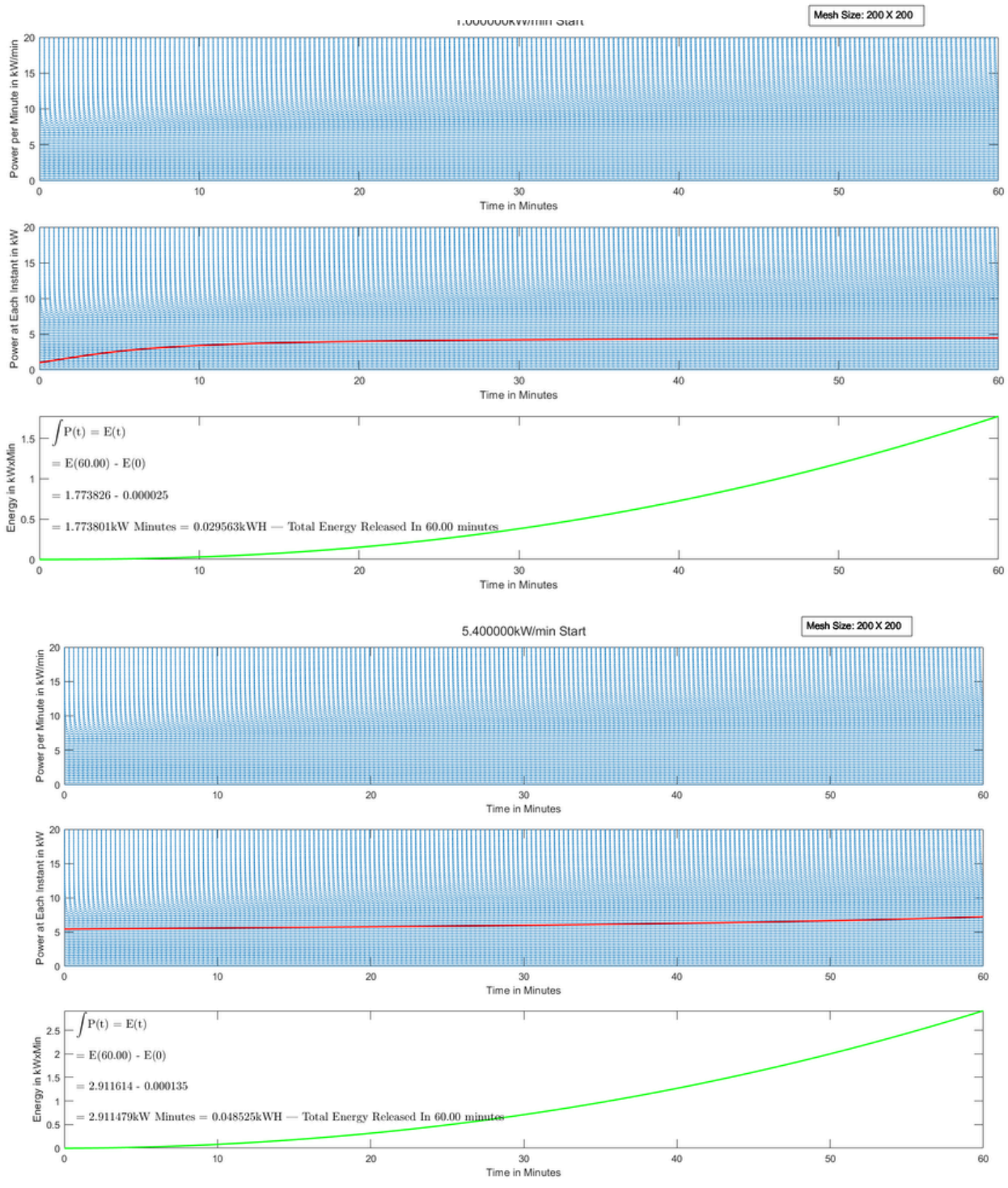
This was an extra credit assignment for my Integral Calculus class. We were just learning how to solve basic differential equations, mostly seperable, and first order linear time invariant ODEs. In this assignment, I was tasked with helping "engineers" (it was a toy problem) start a battery, which had the given power vs time curve shown below. They needed to know what power to start the battery to produce the most output, while also being stable.

It took me a while to realize that this wasn't solvable by hand, and so I took to MATLAB to solve this. It took me several hours, but I eventually figured out that you could plot a slope field for any differential equation using the ODE45 function.

Then, using a given starting point to generate a curve, I just numerically integrated to get the actual Power vs Time, and then numerically integrated again to get the actual energy used in 1 hour. I tested a 1kW, 5,4kW, 5,5kW, and an 8kW start.

I was the only one in my class to be able to get credit for any of this.

$$\frac{dP}{dt} = \frac{P^3 - 10P^2 + 25P}{50 + Pt}$$

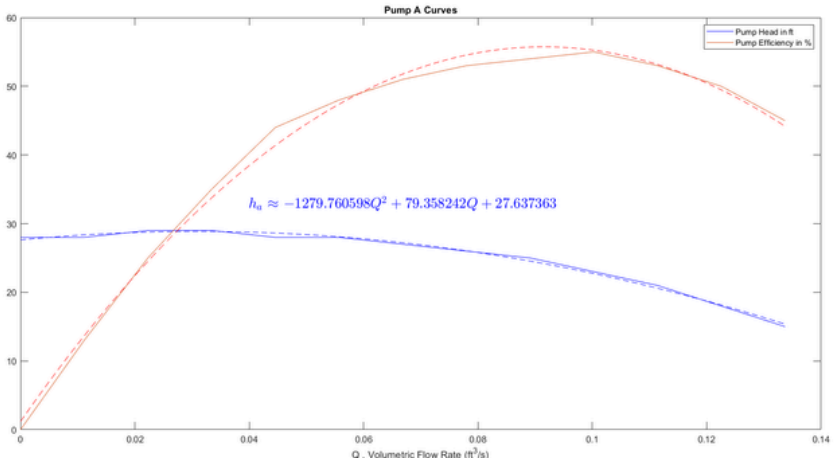


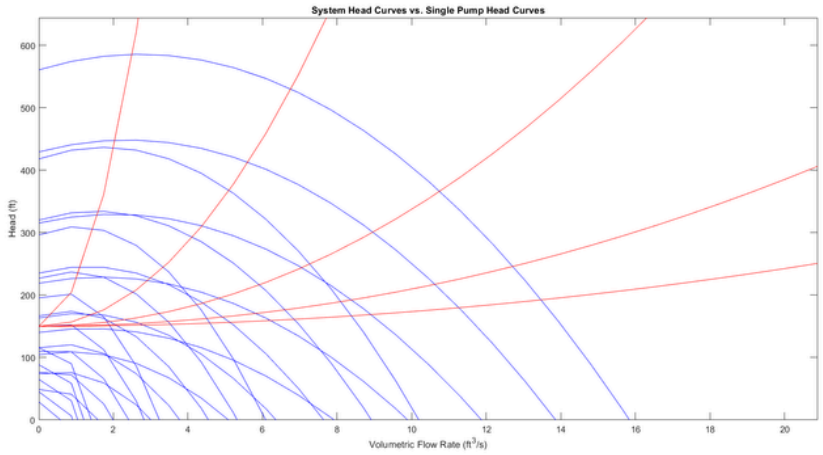
PUMP SIZING AND HEAD LOSS PROJECT — DECEMBER 2022

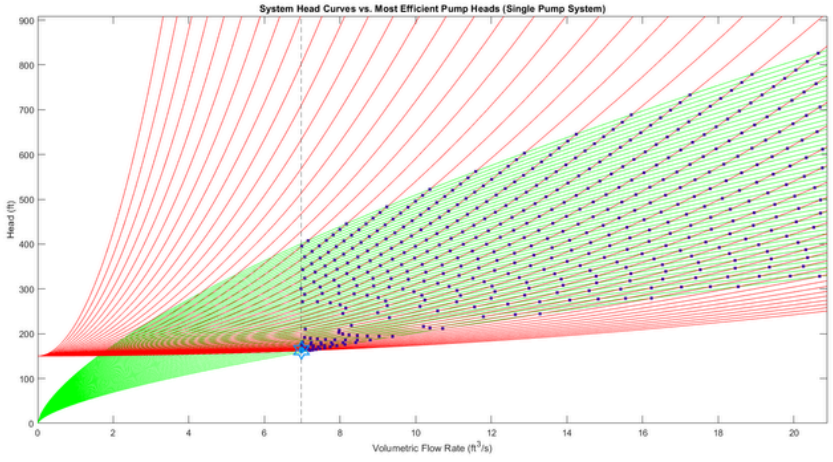
This was the final class project for Fluid Mechanics. In it, we were tasked with delivering water to a city from an aquefier. The city was far away, and above the aquifer. I was tasked with designing the best pipe system, and then after, the best pump for the job. It was immediately obvious to use a diagonal pipe system to cut on distance, as the pipes also caused head loss. However, I was able to choose on the pipe diameter. I was given a small pump, and was expected to apply scaling laws, as if we were buying similar pumps.

I was constrained to run the pump for 8 hours per day, and between 900-1800rpm. Obviously, while choosing the biggest pump around could overcome any system, it may not be efficient. This is why I plotted several scaled pumps across different system curves. Because while the pipe layout remained the same, running the pump harder would ironically also increase head loss. Eventually, the intersection point of the bare minimum pump for the job, and the corresponding system was found to choose that pump.

On the next page, the non-linear and recursive friction factors of each pipe component is calculated with my own MATLAB function.







$$\frac{P_2}{\rho g} + \frac{V_2}{2g} + z_2 = \frac{P_1}{\rho g} + \frac{V_1}{2g} + z_1 + h_{pump} - h_{losses}$$

PUMP SIZING AND HEAD LOSS PROJECT - FLOATING POINT ALGORITHM

The equation below describes the friction factor of each pipe element, which helps calculate head loss. There is no analytical solution to the equation below, and so it must be numerically calculated.

All it does is start at F_fric = 0, and then gradually increment until both sides of the equation are “equal”.

“Equal” is in quotation marks for two reasons: Firstly, even incrementing by the smallest step may skip over the actual, infinitely precise answer. And secondly, I am dealing with a computer. It has finite precision. This is why I introduced a tolerance, which helps with floating point comparisons.

$$\frac{1}{\sqrt{F_{fric}}} = -2 \cdot log\left(\frac{\epsilon/D_{Pipe}}{3.7} + \frac{2.51}{Re\sqrt{F_{fric}}}\right)$$

```
E=2;
D=3;
Re = 5;
resolution = 0.011;

fFriction = coleBrook( E, D, Re, resolution)

function [frictionFactor] = coleBrook(rough,diam,reynolds,resol)
    closeEnough = false;
    fFric = 0;
    tol = 1e-12;%for floating point numbers
    while closeEnough == false
        fFricPrev = fFric;
        if abs(1/sqrt(fFric) + 2*log10((rough/diam)/3.7+2.51/(reynolds*sqrt(fFric)))) > tol %checking inequality with floating point numbers
            fFric = fFric + resol;
        end
        fFric;%delete the semicolon to see in real time
        %check if leftside - rightside < tolerance, OR
        %if it overshot, to use the previous value
        if abs(1/sqrt(fFric) + 2*log10((rough/diam)/3.7+2.51/(reynolds*sqrt(fFric)))) < tol || abs(1/sqrt(fFric)...
                + 2*log10((rough/diam)/3.7+2.51/(reynolds*sqrt(fFric)))) > abs(1/sqrt(fFricPrev)...
                + 2*log10((rough/diam)/3.7+2.51/(reynolds*sqrt(fFricPrev))))
            closeEnough = true;
            if abs(1/sqrt(fFric) + 2*log10((rough/diam)/3.7+2.51/(reynolds*sqrt(fFric)))) > abs(1/sqrt(fFricPrev)...
                    + 2*log10((rough/diam)/3.7+2.51/(reynolds*sqrt(fFricPrev))))
                fFric = fFricPrev;
            end
        end
    end
     frictionFactor = fFric;
end
```

MAJOR HEAD LOSS MATRIX

Each Pipe Diameter is its Own System

Q

	D1	D2	D3	D4	D5	D6	D7	...
Q1	0.000362	0.000301	0.000253	0.000214	0.000182	0.000155	0.000133	
Q2	0.6692	0.550491	0.456328	0.381105	0.320387	0.270903	0.230542	
Q3	2.361235	1.93634	1.60082	1.332895	1.11763	0.943297	0.800418	
Q4	5.017986	4.10649	3.38946	2.817633	2.358845	1.98696	1.68408	
Q5	8.625594	7.049223	5.810636	4.82392	4.03315	3.394365	2.874513	
Q6	13.17865	10.75992	8.861031	7.349411	6.138928	5.161868	4.367347	
Q7	18.56782	15.23389	12.53337	10.39009	8.674517	7.286932	6.162368	
Q8	25.10012	20.47254	16.83484	13.94896	11.6344	9.77309	8.256857	
Q9	32.47771	26.47649	21.76089	18.01279	15.02345	12.60753	10.65131	
Q10	40.79918	33.22743	27.29547	22.59334	18.83423	15.80507	13.33945	
Q11	50.04386	40.75509	33.46211	27.69694	23.07688	19.35546	16.33561	
Q12	60.22041	49.04137	40.26455	33.29408	27.73968	23.25435	19.62578	
Q13	71.34324	58.09806	47.67601	39.42156	32.82806	27.51937	23.21336	
Q14	83.43252	67.87525	55.69798	46.05363	38.35011	32.13185	27.09008	
Q15	96.42007	78.43937	64.36564	53.19313	44.2725	37.1118	31.2724	
Q16	110.3498	89.76979	73.62548	60.84451	50.63984	42.42735	35.76897	
Q17	125.234	101.8765	83.55368	69.01383	57.43816	48.09823	40.54931	
Q18	141.0878	114.7151	94.03503	77.70875	64.64133	54.12927	45.61005	
Q19	157.7736	128.3434	105.2052	86.89495	72.28178	60.52645	50.9998	
Q20	175.5189	142.7064	116.9772	96.617	80.32732	67.26266	56.67509	
Q21	194.0849	157.7994	129.3473	106.8327	88.8196	74.37297	62.63352	
Q22	213.6491	173.7041	142.3115	117.539	97.71943	81.82438	68.90804	
...								

Flow Rate	Head Rise	Shaft Speed	Impeller Diameter	Pipe Diameter	Shaft Power	Cost in US Dollars
9.0305	194.1327	96.1712	2.5278	1.2755	269.9638	1.7118e+06
15.5505	278.9051	96.1712	3.0298	1.2755	667.8801	4.0446e+06
8.4516	185.7454	96.1712	2.4726	1.2959	241.7431	1.5474e+06
17.7932	305.1138	96.1712	3.1690	1.2959	836.0100	5.0303e+06
8.0745	180.1788	96.1712	2.4352	1.3163	224.0362	1.4447e+06
19.9043	328.7942	96.1712	3.2897	1.3163	1.0078e+03	6.0370e+06
7.8009	176.0856	96.1712	2.4074	1.3367	211.5273	1.3725e+06
7.5907	172.9065	96.1712	2.3856	1.3571	202.1104	1.3184e+06
7.4200	170.3053	96.1712	2.3676	1.3776	194.5946	1.2756e+06
7.2803	168.1605	96.1712	2.3526	1.3980	188.5252	1.2412e+06
7.1637	166.3603	96.1712	2.3400	1.4184	183.5206	1.2131e+06
7.0634	164.8035	96.1712	2.3290	1.4388	179.2568	1.1894e+06
6.9774	163.4636	96.1712	2.3195	1.4592	175.6352	1.1694e+06
10.2052	216.2587	98.0946	2.6156	1.2347	339.8536	2.1188e+06
11.7878	238.0753	98.0946	2.7444	1.2347	432.1593	2.6602e+06
8.7181	194.7036	98.0946	2.4818	1.2551	261.3918	1.6596e+06
14.8260	277.4017	98.0946	2.9624	1.2551	633.3295	3.8403e+06

AUTOMATED FILE DATA PLOTTER WITH NOISE FILTER — MAY 2023

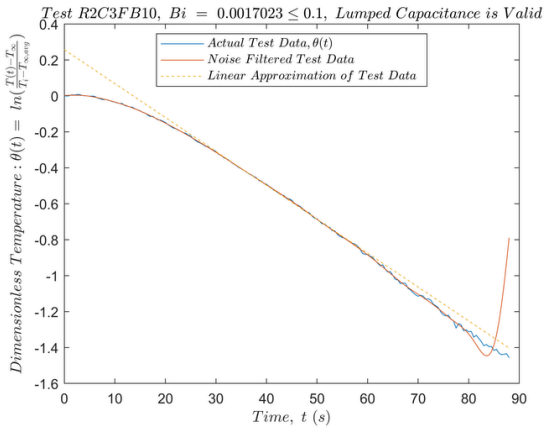
I was tasked with reading several excel files of the same exact structure, to plot very similar graphs. Rather than manually type in the actual file name, I made MATLAB "type it in" for me. I noticed that all of the data sheets were named using the same scheme, so I put the different part as an array of strings in "fileNamesList", and had MATLAB concatenate the entire file name for me, and used that to search for the actual file.

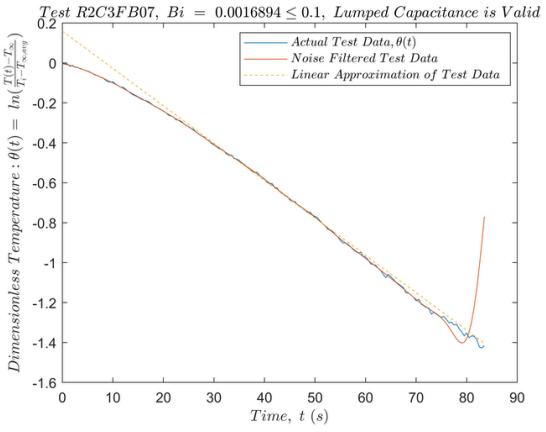
Shown below is the code to create the full file name, and add its data to a cell table for me to easily access later

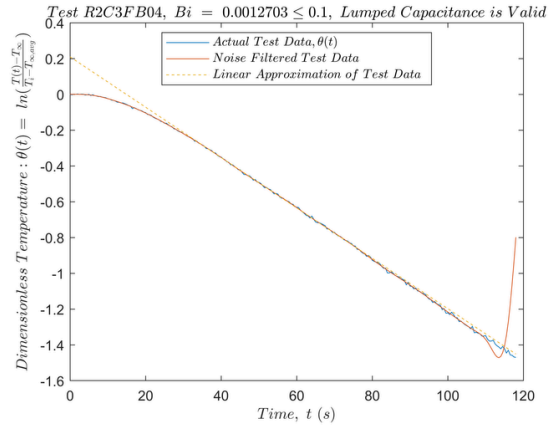
```
for i = 1:width(fileNamesList)
    excellFileName = strcat("Ex05_Data\",fileNamesList{1,i}, ".xlsx");
    labData(1,1+i) = { readtable(excellFileName) };
end
```

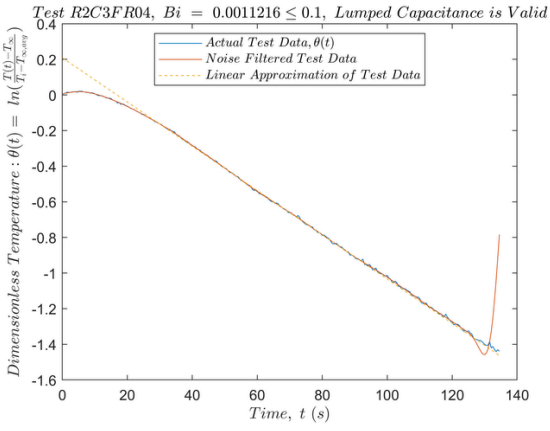
This was also the first time I played around with MATLAB's low pass noise filter to smooth out the bumps, and get rid of the spikes in the actual lab test data. I then used the smoothed data to find the most "linear" part of the graph to plot a best fit line. This was used to find the dimensionless temperature of the test subject.

```
for i = 2:length(labData) %lab data for the tests starts on index 2

    %%Cooling Curves
    %ALL TEMPS ARE IN CELCIUS
    T_SR_test = labData{1,i}{:,3};%is thermocouple #2, on test subject
    T_SR_airinlet = labData{1,i}{:,4};%is thermocouple #3, on air inlet
    T_SR_airinlet_avg = mean(T_SR_airinlet);%is average of inlet air temp
    T_SR_i = T_SR_test(1);%initial test subject temp

    t_SR = labData{1,i}{:,1};%are timestamps of test

    dimensionlessTemp = log( (T_SR_test - T_SR_airinlet) / (T_SR_i - T_SR_airinlet_avg) ); %in matlab, log is actually ln

    fs = 1e3;
    filteredTestTemp = lowpass(dimensionlessTemp,10,fs);

    figure;
    plot(t_SR,dimensionlessTemp);
    hold on
    plot(t_SR,filteredTestTemp);
    hold on
```









PRIME NUMBER COLLECTOR — SEPTEMBER 2019 & SEPTEMBER 2023

The idea for this prime number collector originally came from an assignment I had to do in my senior year of high school. I made it in python. Unfortunately, that file has been lost. I remade this in my senior year of college for another class, this time, in both MATLAB and C++. The idea behind this prime number collector is simple:

Start with two prime numbers you know: [2 , 3]

Keep counting up: 4? divisible by 2. Skip. **5**? not divisible by 2 or 3, so its prime, and lets use this as a factor to check others

now, have [2 , 3 , 5]

Count again: 6? divisible by 2. Skip. **7**? not divisible by 2,3 or 5, so its prime, and lets use this as a factor to check others

now, have [2 , 3 , 5, 7]

Count again: 8? divisible by 2. Skip. 9? divisible by 3. Skip. 10?divisible by 2. Skip. 11? not divisible by 2,3,5 or 7, so its prime, and lets use this as a factor to check others

now, have [2 , 3 , 5, 7 , 11]

... ad nauseum

```
digitsDesired = 3; %%%enter digits here!
primes = primesUpToDigit(digitsDesired) %%%call it here!!!
sprintf("Sum of primes up to %d digits: %d",digitsDesired, sum(primes))

function [primes] = primesUpToDigit(digits)
    maxNum = 10^digits - 1; %for example, if want 3 digits, then get 10^3 -1 =
 1000 -1 = 999, the largest 3 digit numbner
    primes = [2,3]; %will get populated as the loop progresses
    startNumber = primes(end)+1; %will start collecting at the end of the
 primes list + 1, so in this case, at 3+1=4

    for i = startNumber:maxNum
        for ii = 1:length(primes)
            if mod(i,primes(ii)) == 0 %modulo to check for remainders of
 dividing i by primes(ii)
                %if remainder is 0, then that number is divisible, so break
                %loop, and skip this number
                break
            end
        end
        %if it reached the end of the list without breaking loop (for ii),
 then
        %that number was not found to be divisible by anything, so its prime
        %this works because you keep adding more and more numbers to the list,
        %so bigger numbers have more factors to check against
        if ii == length(primes)
            primes(end+1) = i;
        end
    end
    primes;
end


primes =

  Columns 1 through 13

     2     3     5     7    11    13    17    19    23    29    31    37    4

  Columns 14 through 26

    43    47    53    59    61    67    71    73    79    83    89    97   101

  Columns 27 through 39

   103   107   109   113   127   131   137   139   149   151   157   163   167
```

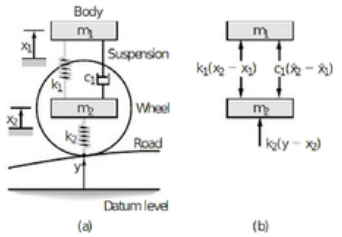
```
#include <iostream>
#include <vector> //so can add elements to arrays at runtime
#include <cmath>

using namespace std;
int desiredDigits = 3;

vector<int> primeFinder(int digitCount){
    int maxNum = pow(10,digitCount) - 1;
    //for example, if want 3 digits, then get 10^3 -1 = 1000 -1 = 999, the largest 3 digit
numbner
    vector<int> primes = {2,3};

    //cout << "maxNum: "<<maxNum << " \n";
    int ii; //need to declare this outside the loop so that stuff outside the loop can access
it

    //sizeof(primes)/sizeof(primes[0]) is the length of the array, primes with plain c++
arrays
    //primes.size() is how it is done with vectors in c++
    //will start collecting at the end of the primes list + 1, so in this case, at 3+1=4
    for(int i = primes[primes.size() - 1] +1; i<=maxNum ; i++){
        //cout << "i = " << i << " \n";
        for(ii = 0;ii<primes.size();ii++){
            if(i % primes[ii] == 0){
                break;
                //modulo to check for remainders of dividing i by primes(ii)
                //if remainder is 0, then that number is divisible, so break
                //loop, and skip this number
            }
        }
        //cout << "checked, ii = " << ii << " , length of primes: " << primes.size() << " \n";
        //if it reached the end of the list without breaking loop (for ii), then
        //that number was not found to be divisible by anything, so its prime
        //this works because you keep adding more and more numbers to the list,
        //so bigger numbers have more factors to check against
        if(ii == primes.size()){
            primes.push_back(i);
        }
    }
    return primes;
}
```

CAR SUSPENSION STATE SPACE MODELING AND SOLVER — MAY 2023

This was a small project for my Vibrations class - recently learning state space modeling, we were tasked with plotting the positions and accelerations of a 2 mass car suspension. This is the first MATLAB file in which I call another file to use as a function. This is then plugged into ODE45, and solved. On the graph, the max points are located, and indicated in the legend using max() and num2str(), as the code below the graph shows.

- $z_1 = x_1$
- $z_2 = x_2$
- $z_3 = \dot{z}_1 = \dot{x}_1$
- $z_4 = \dot{z}_2 = \dot{x}_2$
- $z_5 = \dot{z}_3 = \ddot{z}_1 = \ddot{x}_1$
- $z_6 = \dot{z}_4 = \ddot{z}_2 = \ddot{x}_2$

$$\Rightarrow \vec{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \dot{x}_1 \\ \dot{x}_2 \end{bmatrix}$$



So then...

$$\dot{\vec{z}} = \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \\ \dot{z}_4 \end{bmatrix} = \begin{bmatrix} z_3 \\ z_4 \\ \dot{z}_3 \\ \dot{z}_4 \end{bmatrix} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix}$$

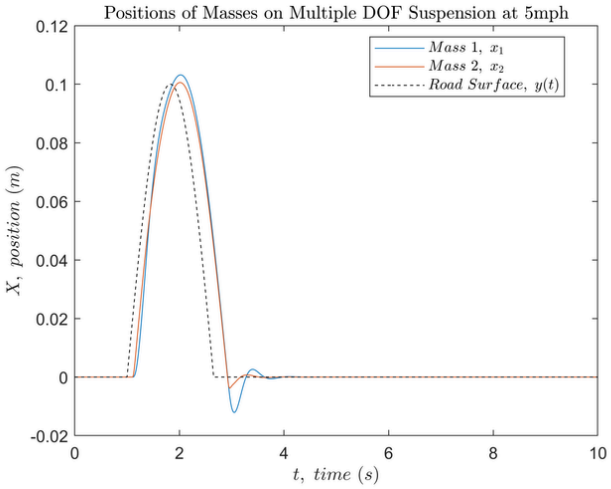
Plugging in $\ddot{x}_1$ and $\ddot{x}_2$...

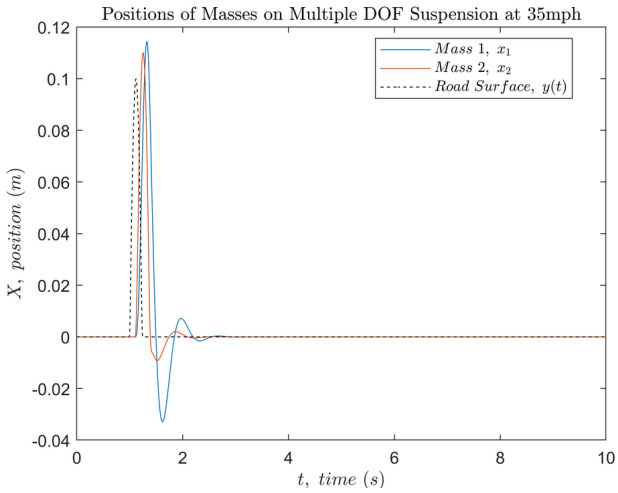
$$\dot{\vec{z}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{k_1}{m_1} & \frac{k_1}{m_1} & -\frac{c}{m_1} & \frac{c}{m_1} \\ \frac{k_1}{m_2} & -\frac{k_1+k_2}{m_2} & \frac{c}{m_2} & \frac{c}{m_1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ k_2 \end{bmatrix} [y]$$

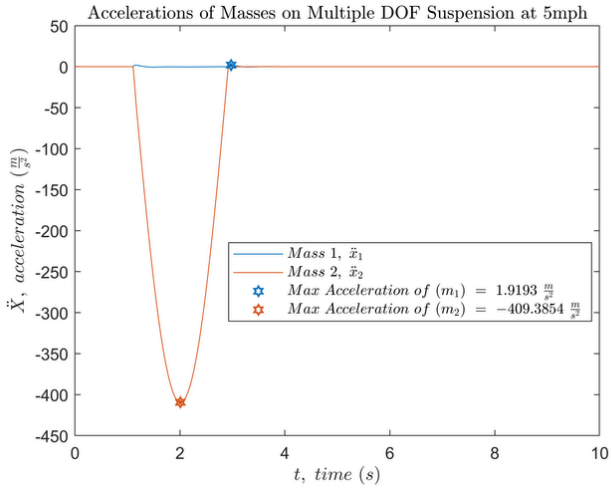
- $m_1 = 320kg$
- $m_2 = 44kg$
- $k_1 = 3.2 \times 10^4 \frac{N}{m}$
- $k_2 = 1.8 \times 10^5 \frac{N}{m}$
- $c_1 = 3430 \frac{N \cdot s}{m}$
- $L = 3.7m$
- $h = 0.1m.$
- $v = 5, 10,$ and $35mph$
- $d = v \cdot 1s$

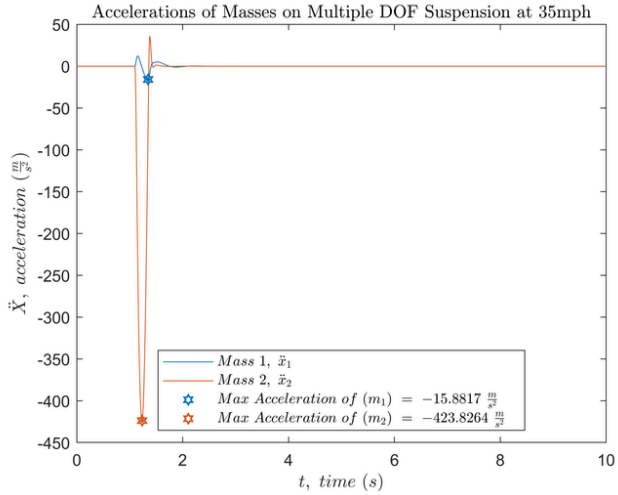
The road profile is given as

$$y(t) = \begin{cases} 0 & 0 \le t < \frac{d}{v} \\ h \sin\left(\frac{\pi v}{L}\left(t - \frac{d}{v}\right)\right) & \frac{d}{v} \le t \le \frac{d+L}{v} \\ 0 & t > \frac{d+L}{v} \end{cases}$$










```
%note, will need abs to get max negative, but will only return pos values
[maxAccel1,accIdx1] = max(abs(accel(:,1)));
[maxAccel2,accIdx2] = max(abs(accel(:,2)));
maxAccel1 = accel(accIdx1,1);
maxAccel2 = accel(accIdx2,2);
```

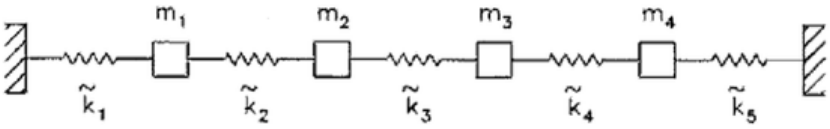
```
legendStuff = ["$Mass \ 1, \ \ddot{x}_{1}$","$Mass \ 2, \ \ddot{x}_{2}$"...
    ,strcat("$Max \ Acceleration \ of \ (m_{1}) \ = \ "...
    ,num2str(maxAccel1) ," \ \frac{m}{s^2} \ $"),...
    strcat("$Max \ Acceleration \ of \ (m_{2}) \ = \ ",num2str(maxAccel2) ," \ \frac{m}{s^2} \ $")];
```

```
[t , z]= ode45 ( @ (t , z ) multipleDOF_Car_Suspension_Function(t,z,y,yTime,m,c,k) , time , IC );
```

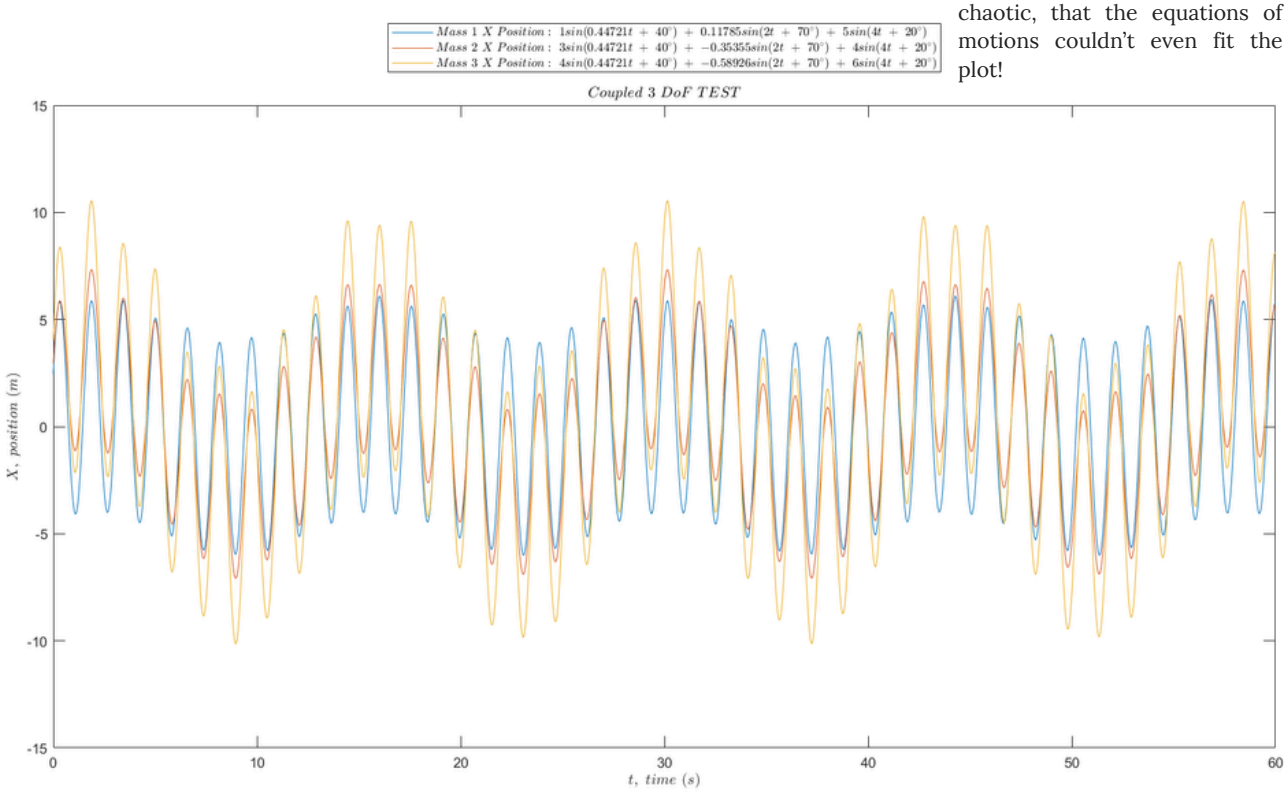
$$\begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} + \begin{bmatrix} 27 & -3 \\ -3 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

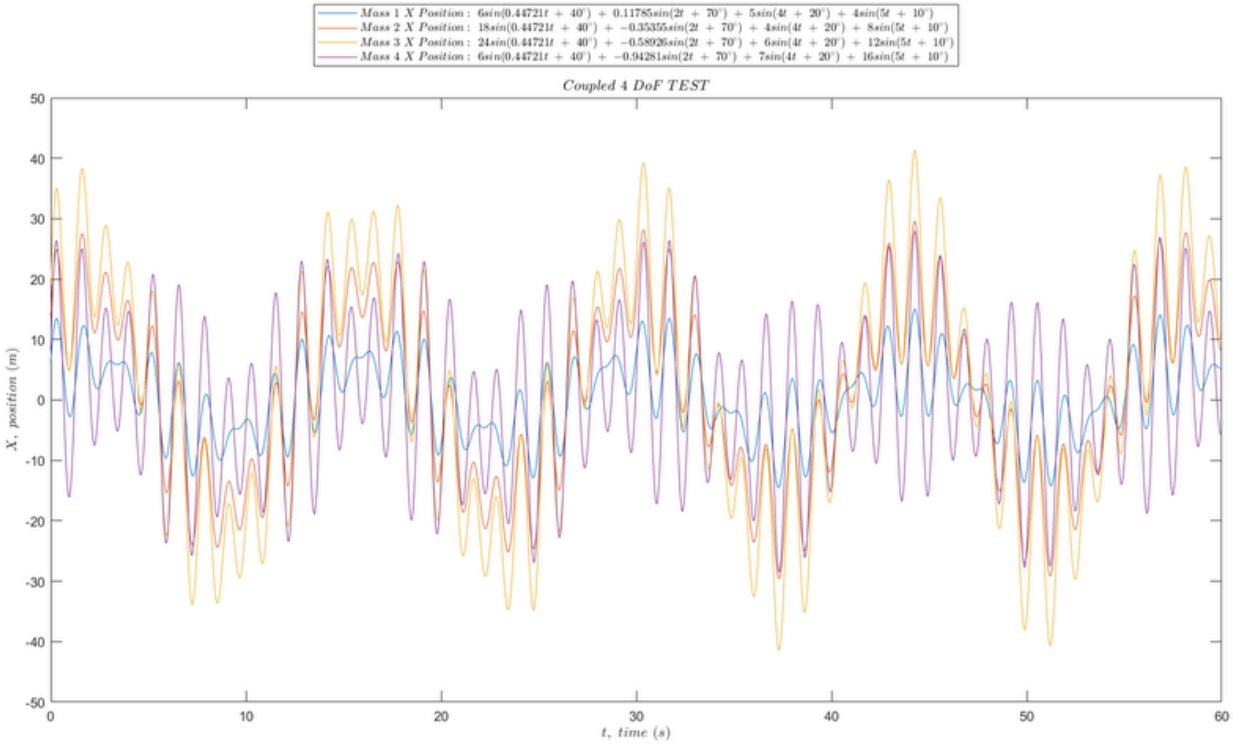
$$x(t) = A_1 \sin(\omega_1 t + \phi_1)\, u_1 + A_2 \sin(\omega_2 t + \phi_2)\, u_2$$

This project was born from me going overboard on a homework problem. Originally, I was supposed to plot the equations of motion for a series of 2 mass spring systems. They were given initial conditions, and were unforced. Realizing that the addition of terms was repetitive, and the u vector, the natural frequencies, and the phi angles were also repetitively added, I thought to make it n-DOF.

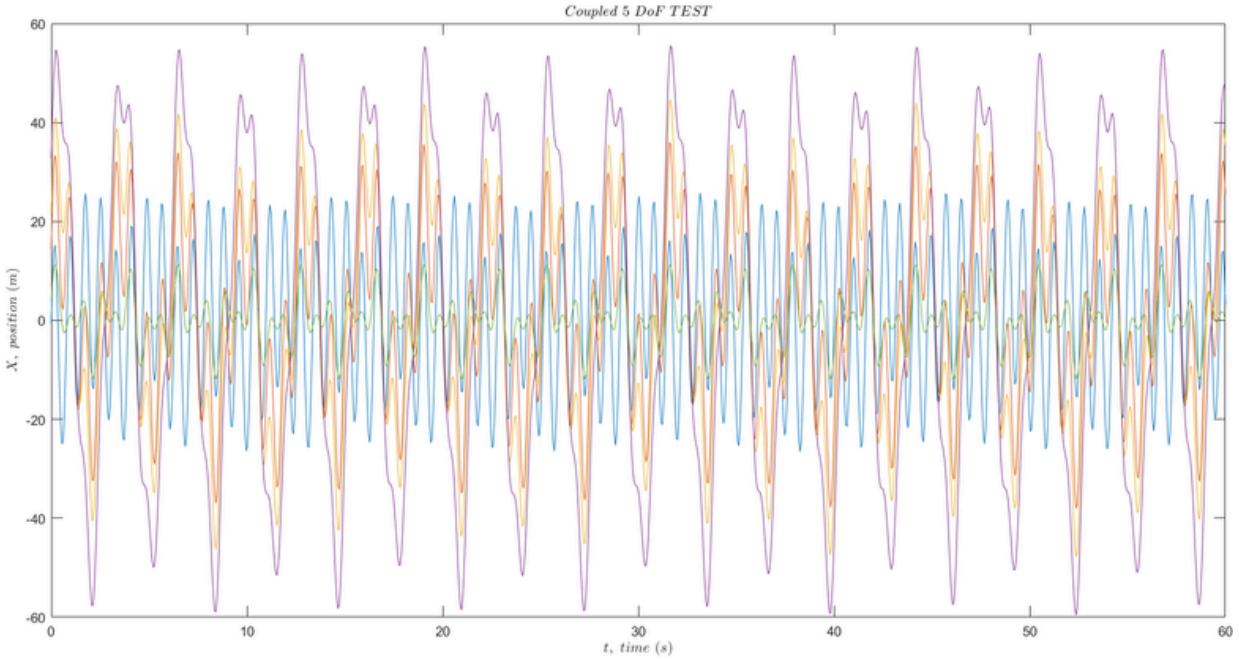


I first tried to just add 3 mass-springs in series. Then I went with 4. Then 5. I then went crazy, and tried a 21 DOF system (on the next page). It was so chaotic, that the equations of motions couldn't even fit the plot!

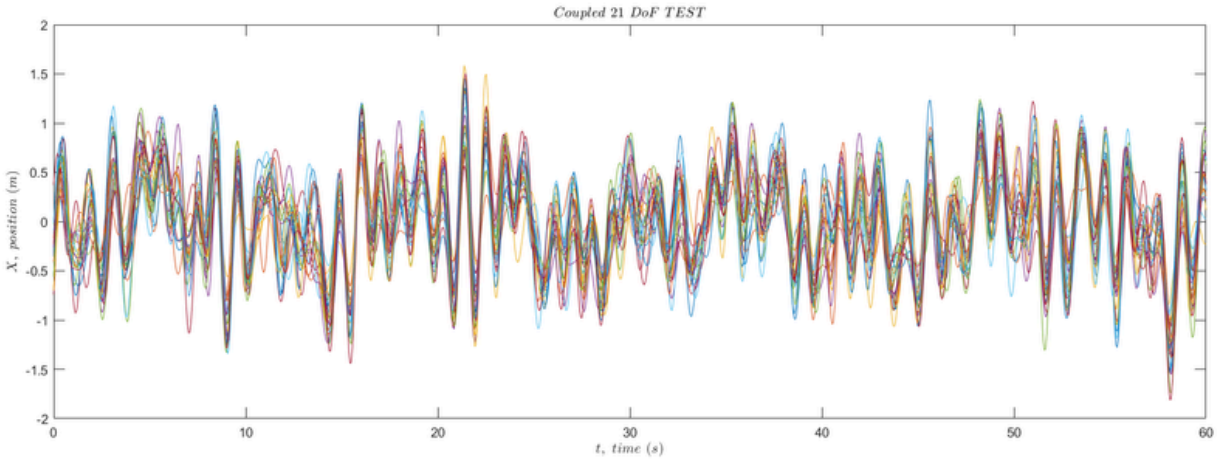




$$\begin{aligned}
&Mass\ 1\ X\ Position: \ 1sin(0.4472lt + 0^\circ) + -6sin(2t + 40^\circ) + -0.5892\delta sin(4t + 70^\circ) + 1sin(5t + 20^\circ) + 20sin(8t + 10^\circ)\\
&Mass\ 2\ X\ Position: \ 3sin(0.4472lt + 0^\circ) + 18sin(2t + 40^\circ) + -0.4714sin(4t + 70^\circ) + 2sin(5t + 20^\circ) + 16sin(8t + 10^\circ)\\
&Mass\ 3\ X\ Position: \ 4sin(0.4472lt + 0^\circ) + 30sin(2t + 40^\circ) + -0.7071sin(4t + 70^\circ) + 3sin(5t + 20^\circ) + 12sin(8t + 10^\circ)\\
&Mass\ 4\ X\ Position: \ 1sin(0.4472lt + 0^\circ) + 48sin(2t + 40^\circ) + -0.8249\delta sin(4t + 70^\circ) + 4sin(5t + 20^\circ) + 8sin(8t + 10^\circ)\\
&Mass\ 5\ X\ Position: \ 0sin(0.4472lt + 0^\circ) + 3sin(2t + 40^\circ) + -0.1666\gamma sin(4t + 70^\circ) + 5sin(5t + 20^\circ) + 4sin(8t + 10^\circ)
\end{aligned}$$



$$\begin{aligned}
&+ -0.072746sin(5.6236t + -175.3625^\circ) + -0.2215\delta sin(5.7243t + 81.6917^\circ) + 0.098986sin(5.885t + 59.2194^\circ) + 0.11602sin(5.9887t + 29.3323^\circ) + 0.011682sin(6.3405t + 266.3575^\circ) + -0.15347sin(6.7805t + -169.3591^\circ)\\
&+ -0.032235sin(5.6236t + -175.3625^\circ) + -0.12929sin(5.7243t + 81.6917^\circ) + 0.22565sin(5.885t + 59.2194^\circ) + 0.070998sin(5.9887t + 29.3323^\circ) + 0.043775sin(6.3405t + 266.3575^\circ) + -0.049501sin(6.7805t + -169.3591^\circ)\\
&-0.075857sin(5.6236t + -175.3625^\circ) + -0.0656sin(5.7243t + 81.6917^\circ) + 0.008547sin(5.885t + 59.2194^\circ) + 0.023065sin(5.9887t + 29.3323^\circ) + 0.038492sin(6.3405t + 266.3575^\circ) + -0.11057sin(6.7805t + -169.3591^\circ)\\
&-0.023262sin(5.6236t + -175.3625^\circ) + -0.14715sin(5.7243t + 81.6917^\circ) + 0.16145sin(5.885t + 59.2194^\circ) + 0.13341sin(5.9887t + 29.3323^\circ) + 0.1147\delta sin(6.3405t + 266.3575^\circ) + -0.19687sin(6.7805t + -169.3591^\circ)\\
&-0.0541sin(5.6236t + -175.3625^\circ) + -0.16562sin(5.7243t + 81.6917^\circ) + 0.11872sin(5.885t + 59.2194^\circ) + 0.17006sin(5.9887t + 29.3323^\circ) + 0.14507sin(6.3405t + 266.3575^\circ) + -0.06215sin(6.7805t + -169.3591^\circ)\\
&) + -0.051418sin(5.6236t + -175.3625^\circ) + -0.041442sin(5.7243t + 81.6917^\circ) + 0.061308sin(5.885t + 59.2194^\circ) + 0.092465sin(5.9887t + 29.3323^\circ) + 0.059181sin(6.3405t + 266.3575^\circ) + -0.23934sin(6.7805t + -169.3591^\circ)\\
&01^\circ) + -0.041601sin(5.6236t + -175.3625^\circ) + -0.20255sin(5.7243t + 81.6917^\circ) + 0.067869sin(5.885t + 59.2194^\circ) + 0.019832sin(5.9887t + 29.3323^\circ) + 0.1341\gamma sin(6.3405t + 266.3575^\circ) + -0.07874sin(6.7805t + -169.3591^\circ)\\
&-0.053869sin(5.6236t + -175.3625^\circ) + -0.077717sin(5.7243t + 81.6917^\circ) + 0.16055sin(5.885t + 59.2194^\circ) + 0.058186sin(5.9887t + 29.3323^\circ) + 0.11603sin(6.3405t + 266.3575^\circ) + -0.29736sin(6.7805t + -169.3591^\circ)\\
&01^\circ) + -0.051432sin(5.6236t + -175.3625^\circ) + -0.13215sin(5.7243t + 81.6917^\circ) + 0.035775sin(5.885t + 59.2194^\circ) + 0.033552sin(5.9887t + 29.3323^\circ) + 0.0011537sin(6.3405t + 266.3575^\circ) + -0.08721sin(6.7805t + -169.3591^\circ)\\
&+ -0.013745sin(5.6236t + -175.3625^\circ) + -0.20031sin(5.7243t + 81.6917^\circ) + 0.22723sin(5.885t + 59.2194^\circ) + 0.06136sin(5.9887t + 29.3323^\circ) + 0.10346sin(6.3405t + 266.3575^\circ) + -0.24812sin(6.7805t + -169.3591^\circ)\\
&') + -0.0098782sin(5.6236t + -175.3625^\circ) + -0.086519sin(5.7243t + 81.6917^\circ) + 0.17879sin(5.885t + 59.2194^\circ) + 0.096097sin(5.9887t + 29.3323^\circ) + 0.06645sin(6.3405t + 266.3575^\circ) + -0.06115sin(6.7805t + -169.3591^\circ)\\
&+ -0.077093sin(5.6236t + -175.3625^\circ) + -0.09979\delta sin(5.7243t + 81.6917^\circ) + 0.26422sin(5.885t + 59.2194^\circ) + 0.11511sin(5.9887t + 29.3323^\circ) + 0.15737sin(6.3405t + 266.3575^\circ) + -0.093186sin(6.7805t + -169.3591^\circ)\\
&-0.013204sin(5.6236t + -175.3625^\circ) + -0.21203sin(5.7243t + 81.6917^\circ) + 0.26572sin(5.885t + 59.2194^\circ) + 0.099883sin(5.9887t + 29.3323^\circ) + 0.00019776sin(6.3405t + 266.3575^\circ) + -0.029532sin(6.7805t + -169.3591^\circ)\\
&) + -0.0025156sin(5.6236t + -175.3625^\circ) + -0.091149sin(5.7243t + 81.6917^\circ) + 0.24242sin(5.885t + 59.2194^\circ) + 0.19115sin(5.9887t + 29.3323^\circ) + 0.079453sin(6.3405t + 266.3575^\circ) + -0.18676sin(6.7805t + -169.3591^\circ)\\
&-0.043305sin(5.6236t + -175.3625^\circ) + -0.28325sin(5.7243t + 81.6917^\circ) + 0.012758sin(5.885t + 59.2194^\circ) + 0.11312sin(5.9887t + 29.3323^\circ) + 0.072907sin(6.3405t + 266.3575^\circ) + -0.2215sin(6.7805t + -169.3591^\circ)\\
&-0.068049sin(5.6236t + -175.3625^\circ) + -0.33764sin(5.7243t + 81.6917^\circ) + 0.026122sin(5.885t + 59.2194^\circ) + 0.20605sin(5.9887t + 29.3323^\circ) + 0.079189sin(6.3405t + 266.3575^\circ) + -0.17716sin(6.7805t + -169.3591^\circ)\\
&-0.051637sin(5.6236t + -175.3625^\circ) + -0.25091sin(5.7243t + 81.6917^\circ) + 0.12134sin(5.885t + 59.2194^\circ) + 0.13925sin(5.9887t + 29.3323^\circ) + 0.13232sin(6.3405t + 266.3575^\circ) + -0.13799sin(6.7805t + -169.3591^\circ)\\
&1^\circ) + -0.014695sin(5.6236t + -175.3625^\circ) + -0.11815sin(5.7243t + 81.6917^\circ) + 0.20154sin(5.885t + 59.2194^\circ) + 0.20912sin(5.9887t + 29.3323^\circ) + 0.055403sin(6.3405t + 266.3575^\circ) + -0.20888sin(6.7805t + -169.3591^\circ)\\
&') + -0.028467sin(5.6236t + -175.3625^\circ) + -0.20068sin(5.7243t + 81.6917^\circ) + 0.24846sin(5.885t + 59.2194^\circ) + 0.052561sin(5.9887t + 29.3323^\circ) + 0.13482sin(6.3405t + 266.3575^\circ) + -0.20991sin(6.7805t + -169.3591^\circ)\\
&') + -0.035552sin(5.6236t + -175.3625^\circ) + -0.037029sin(5.7243t + 81.6917^\circ) + 0.15475sin(5.885t + 59.2194^\circ) + 0.14764sin(5.9887t + 29.3323^\circ) + 0.080983sin(6.3405t + 266.3575^\circ) + -0.22009sin(6.7805t + -169.3591^\circ)\\
&-0.075747sin(5.6236t + -175.3625^\circ) + -0.3114sin(5.7243t + 81.6917^\circ) + 0.31131sin(5.885t + 59.2194^\circ) + 0.06312sin(5.9887t + 29.3323^\circ) + 0.0061443sin(6.3405t + 266.3575^\circ) + -0.20607sin(6.7805t + -169.3591^\circ)
\end{aligned}$$



Shown below is the algorithm for creating the equations of motion. There are n equations, and each equation has n different terms, each for how every other coordinate system affects a specific mass. This is why there are two nested loops.

$$\vec{x(t)} = A_1 sin(\omega_1 t + \phi_1)\vec{u_1} + A_2 sin(\omega_2 t + \phi_2)\vec{u_2} + A_3 sin(\omega_3 t + \phi_3)\vec{u_3} + ...$$

$$\Downarrow$$

$$\vec{x(t)} = A_1 sin(\omega_1 t + \phi_1)\begin{bmatrix} u_{1,1} \\ u_{1,2} \\ u_{1,3} \\ \vdots \end{bmatrix} + A_2 sin(\omega_2 t + \phi_2)\begin{bmatrix} u_{2,1} \\ u_{2,2} \\ u_{2,3} \\ \vdots \end{bmatrix} + A_3 sin(\omega_3 t + \phi_3)\begin{bmatrix} u_{3,1} \\ u_{3,2} \\ u_{3,3} \\ \vdots \end{bmatrix} + ...$$

$$\Downarrow$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} A_1 sin(\omega_1 t + \phi_1)u_{1,1} + A_2 sin(\omega_2 t + \phi_2)u_{2,1} + A_3 sin(\omega_3 t + \phi_3)u_{3,1} + ... \\ A_1 sin(\omega_1 t + \phi_1)u_{1,2} + A_2 sin(\omega_2 t + \phi_2)u_{2,2} + A_3 sin(\omega_3 t + \phi_3)u_{3,2} + ... \\ A_1 sin(\omega_1 t + \phi_1)u_{1,3} + A_2 sin(\omega_2 t + \phi_2)u_{2,3} + A_3 sin(\omega_3 t + \phi_3)u_{3,3} + ... \\ \vdots + \vdots + \vdots + ... \end{bmatrix}$$

$$\Downarrow$$

```
function [coupledDoFValues] = coupledDoFPlotter(A_Values, phi_Values,natFreqs,u_Vectors,t_vals,titled)
    X = zeros([length(A_Values),length(t_vals)]);
    for i = 1:length(A_Values) %gets X mass1, X mass2, ... and selects which u vector to use
        %X(i,:) = X(i,:) + A_Values(1)*sin(natFreqs(1)*t_vals + phi_Values(1)*180/pi)*u_Vectors(1,i);
        for ii = 1:length(A_Values) %within X mass1, goes thru A_1, A2..., phi_1, phi_2..., selects
            %which value in the specified u vector to use
            X(i,:) = X(i,:) + A_Values(ii)*sin(natFreqs(ii)*t_vals + phi_Values(ii)*pi/180)*u_Vectors(ii,i);
        end
    end
    coupledDoFValues = X;
    figure;
    for i = 1:length(A_Values)
        plot(t_vals,X(i,:))
        hold on;
    end
```

Shown below is how I made the legend automatically write the equation of motion for me. It would have been too tedious to write the equations of motion for the 21-DOF system myself

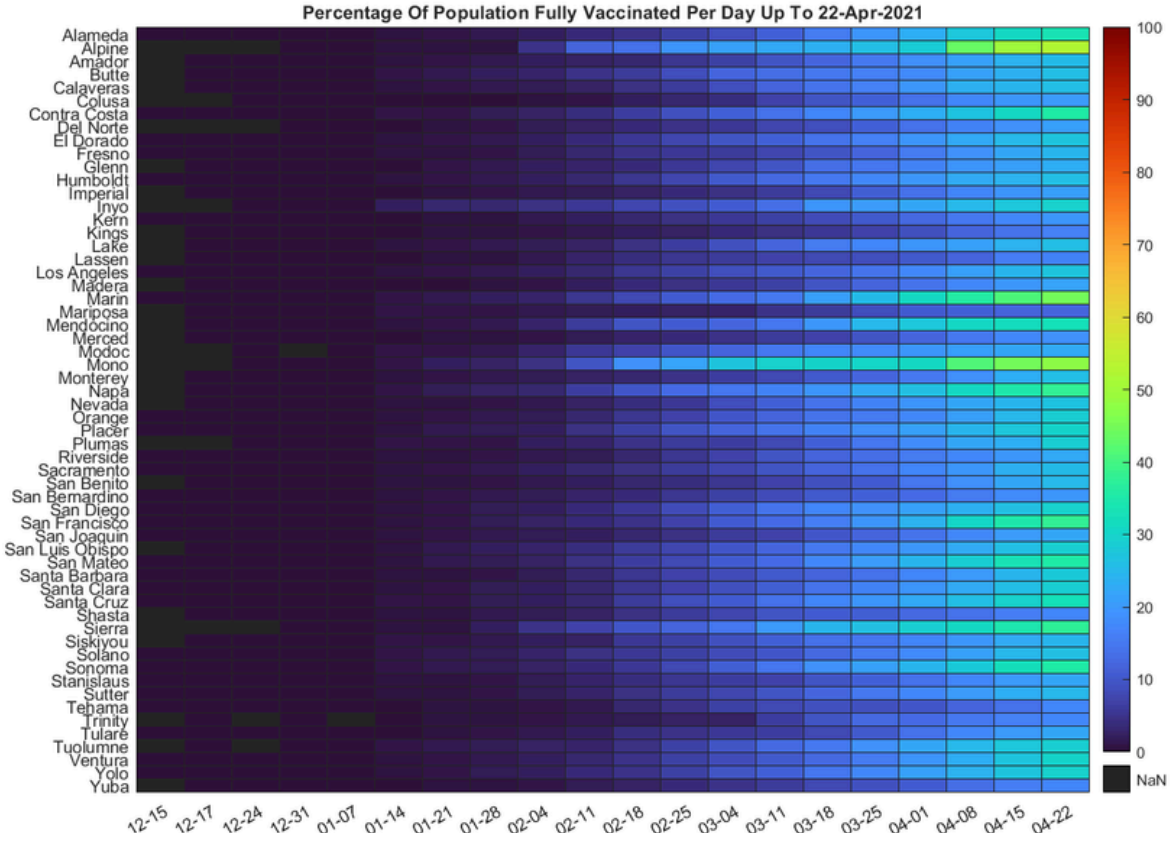
```
for i = 1:length(A_Values)
    a = strcat("$Mass \ ", num2str(i)," \ X \ Position : \ ");
    b = "";
    for ii = 1:length(A_Values)
        if ii > 1
          b = strcat(b, " \ + \ ") ;
        end
        b = strcat(b, num2str(u_Vectors(ii,i)*A_Values(ii)),"sin(",num2str(natFreqs(ii)),"t \ + \ ", num2str(phi_Values(ii)), "^{\circ}  )");
    end
    c = strcat(a,b," $");
    legendStuff(i) = c;
end
```

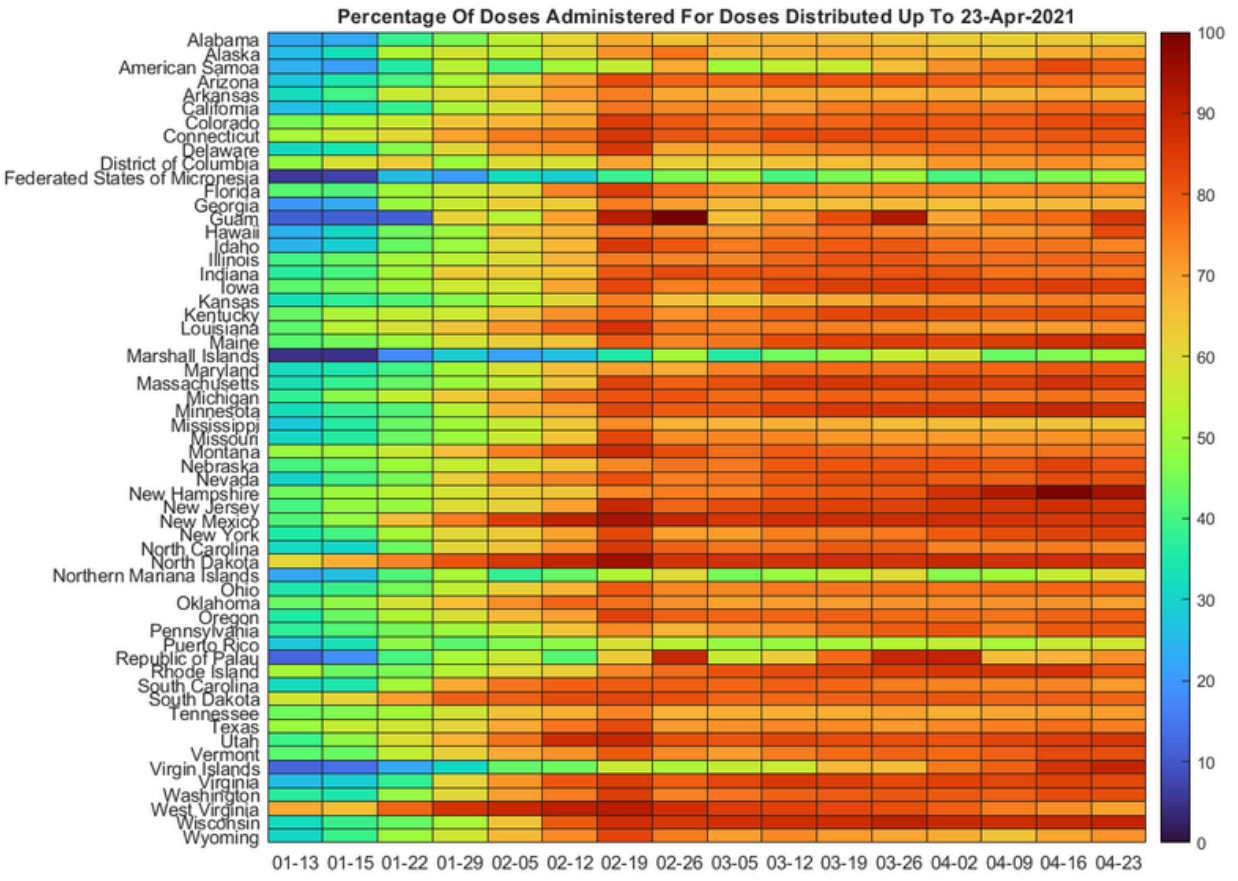
REPRESENTING COVID-19 VACCINATIONS WITH HEAT MAPS — MAY 2021

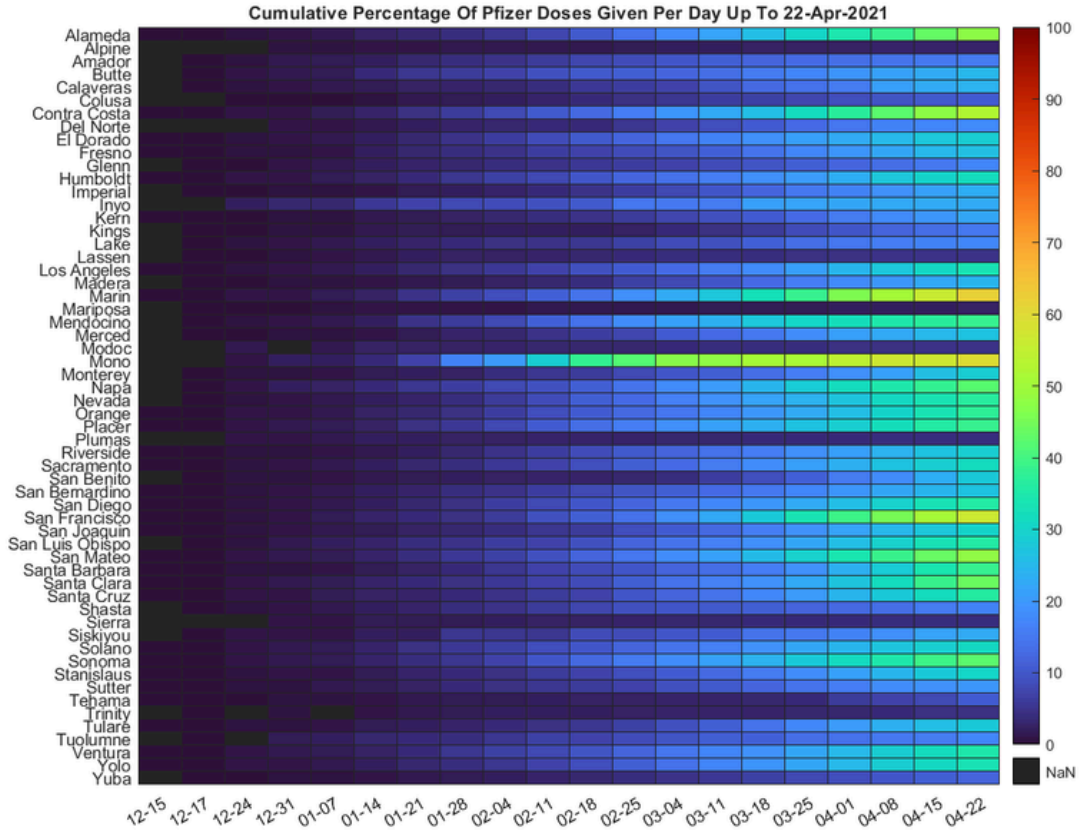
I was in my freshman year of college in the second year of the pandemic, and the vaccines had just started to be rolled out. We were tasked with representing vaccination data using heat maps - each place would be tracked every couple of days, and the more vaccines it received, the "hotter" it appeared on the map.

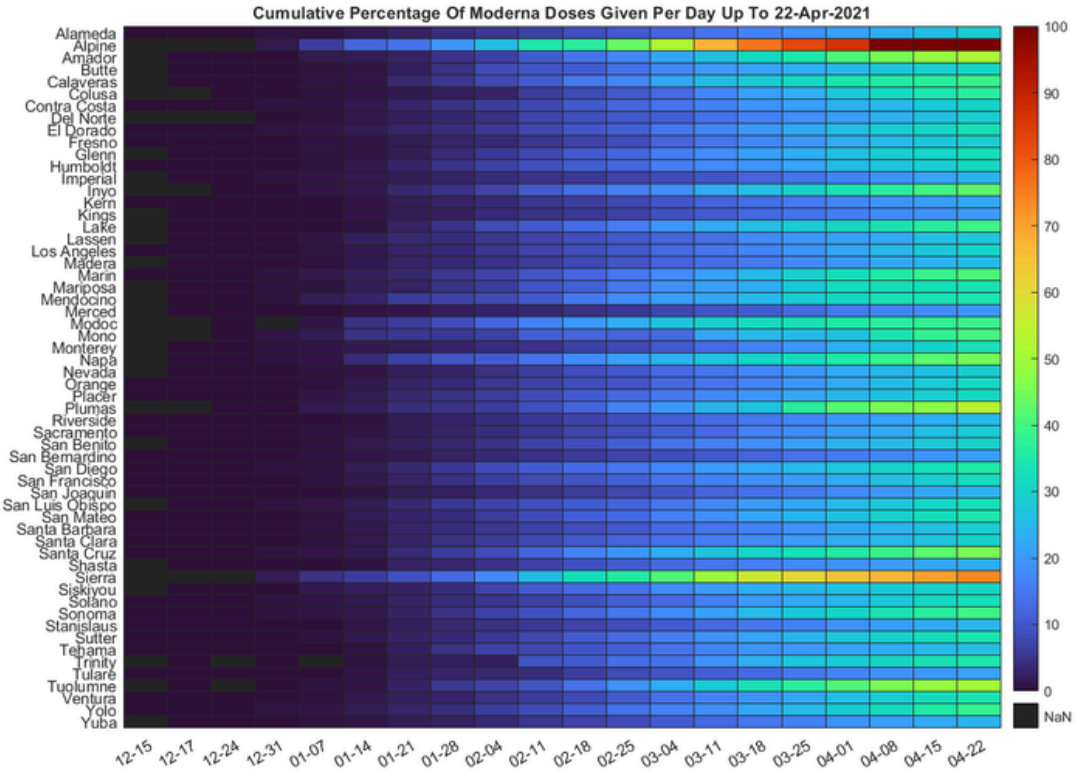
.

This was sorted by county and manufacturer.

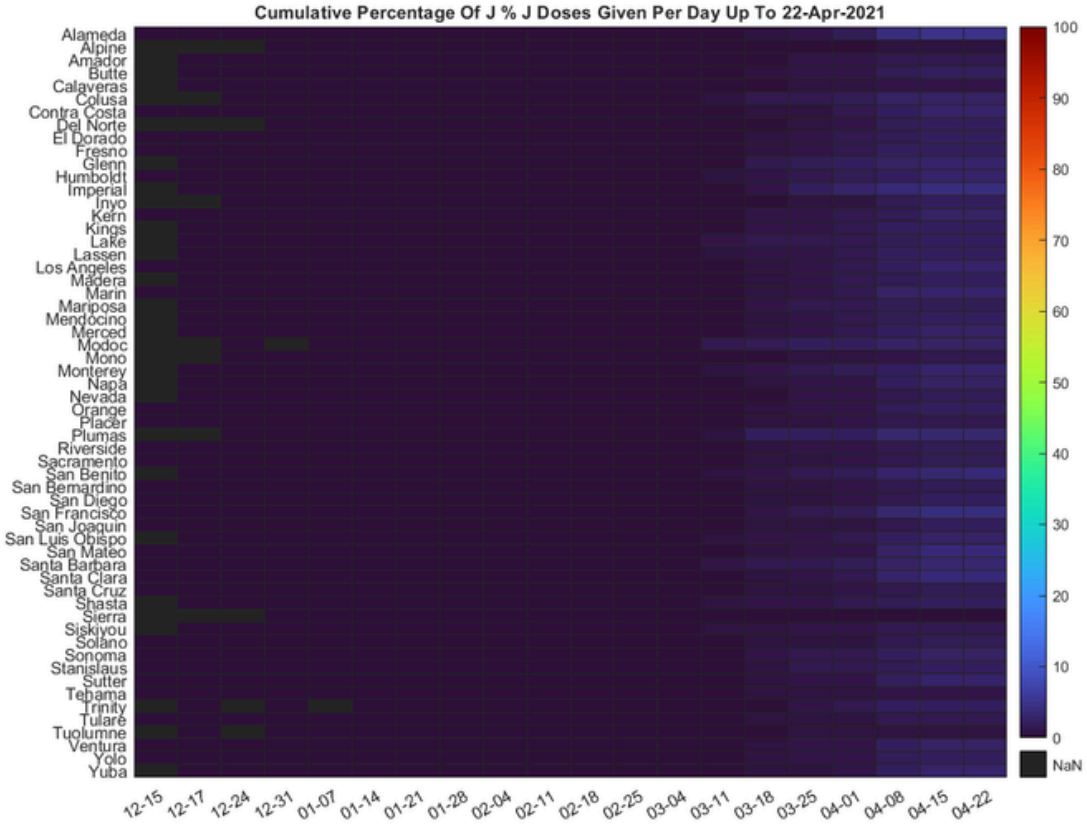


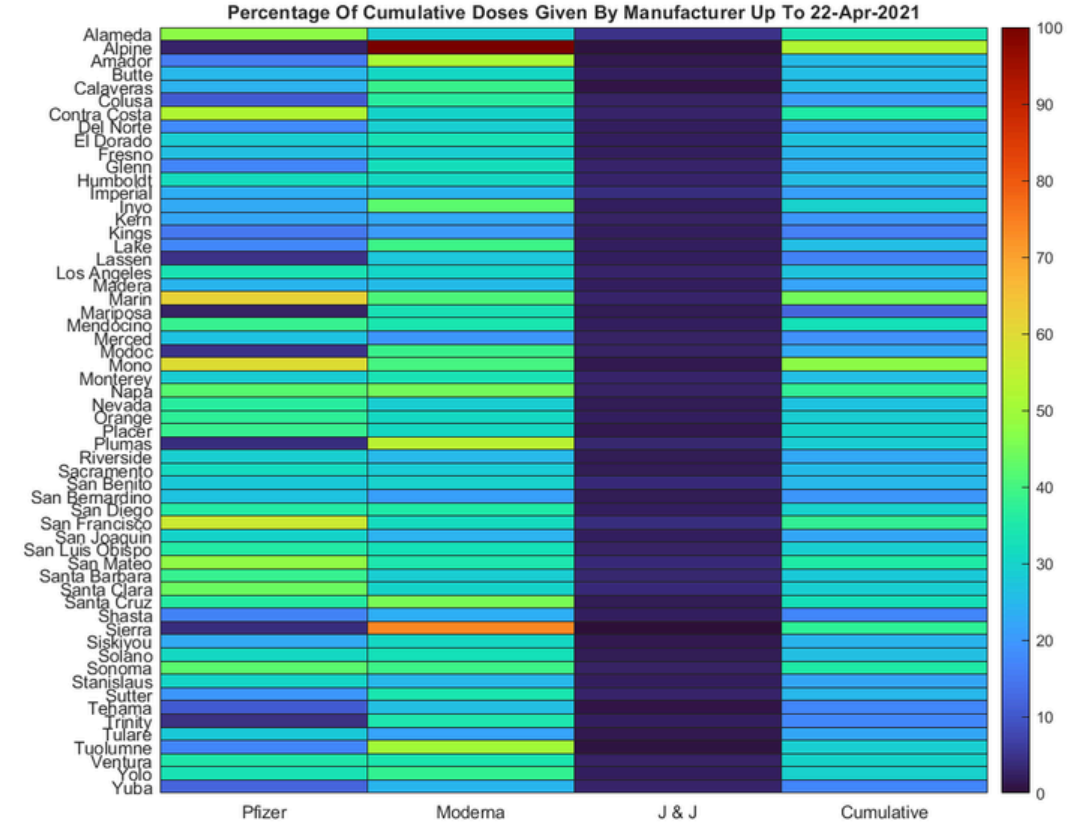


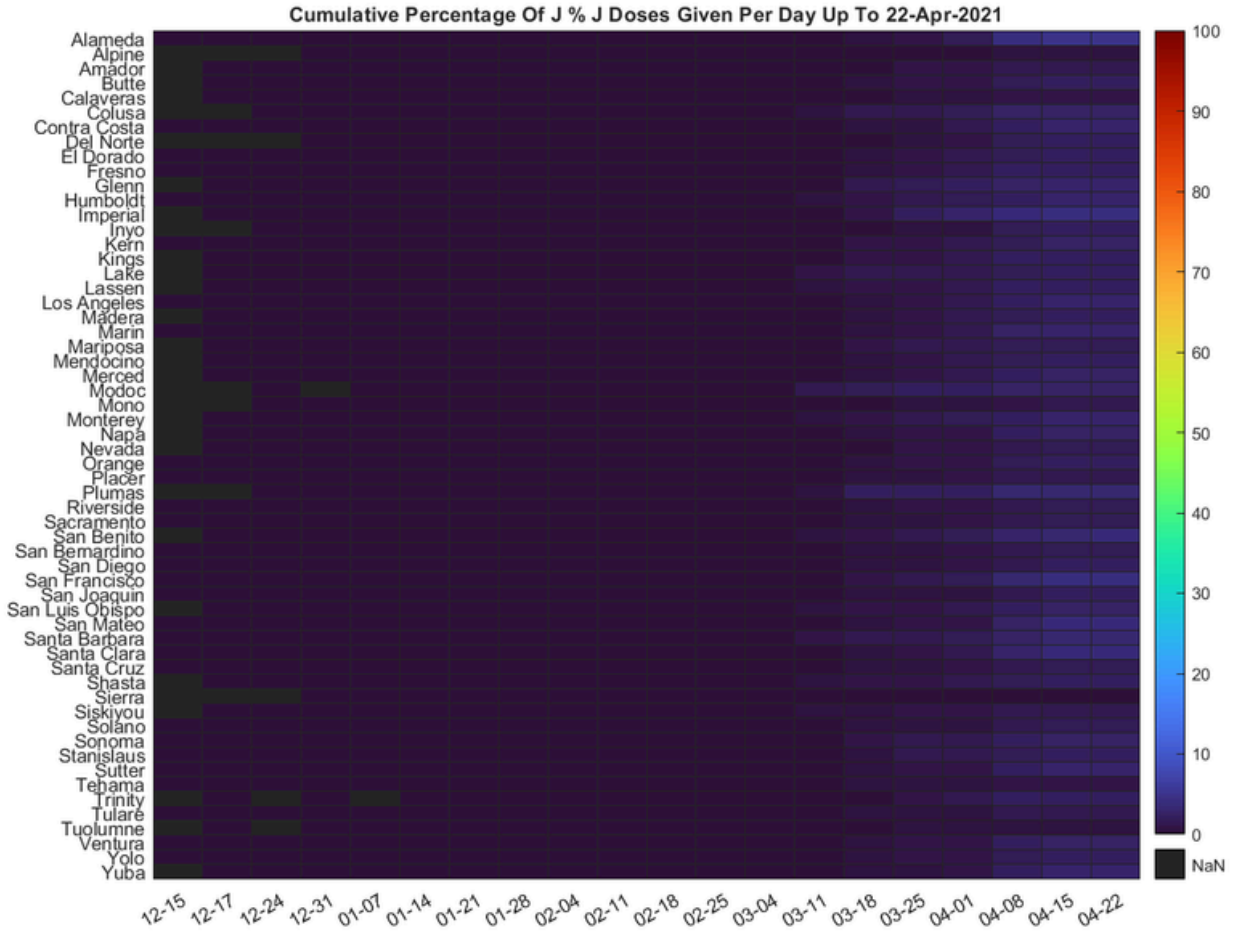

Cumulative Percentage Of Pfizer Doses Given Per Day Up To 22-Apr-2021
Alameda
Alpine
Amador
Butte
Calaveras
Colusa
Contra Costa
Del Norte
El Dorado
Fresno
Glenn
Humboldt
Imperial
Inyo
Kern
Kings
Lake
Lassen
Los Angeles
Madera
Marin
Mariposa
Mendocino
Merced
Modoc
Mono
Monterey
Napa
Nevada
Orange
Placer
Plumas
Riverside
Sacramento
San Benito
San Bernardino
San Diego
San Francisco
San Joaquin
San Luis Obispo
San Mateo
Santa Barbara
Santa Clara
Santa Cruz
Shasta
Sierra
Siskiyou
Solano
Sonoma
Stanislaus
Sutter
Tehama
Trinity
Tulare
Tuolumne
Ventura
Yolo
Yuba
12-15
12-17
12-24
12-31
01-07
01-14
01-21
01-28
02-04
02-11
02-18
02-25
03-04
03-11
03-18
03-25
04-01
04-08
04-15
04-22
100
90
80
70
60
50
40
30
20
10
0
NaN

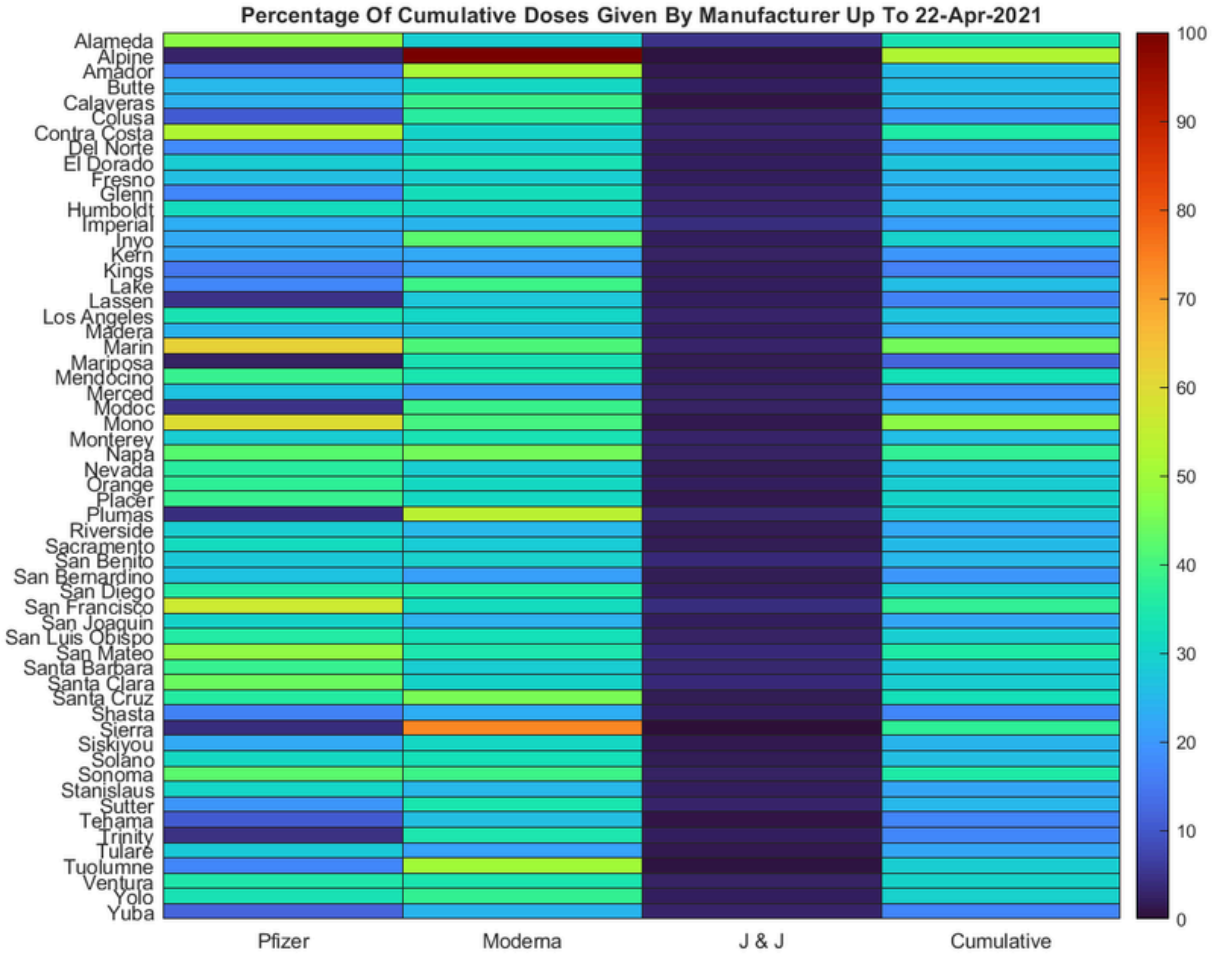

Cumulative Percentage Of Moderna Doses Given Per Day Up To 22-Apr-2021
Alameda
Alpine
Amador
Butte
Calaveras
Colusa
Contra Costa
Del Norte
El Dorado
Fresno
Glenn
Humboldt
Imperial
Inyo
Kern
Kings
Lake
Lassen
Los Angeles
Madera
Marin
Mariposa
Mendocino
Merced
Modoc
Mono
Monterey
Napa
Nevada
Orange
Placer
Plumas
Riverside
Sacramento
San Benito
San Bernardino
San Diego
San Francisco
San Joaquin
San Luis Obispo
San Mateo
Santa Barbara
Santa Clara
Santa Cruz
Shasta
Sierra
Siskiyou
Solano
Sonoma
Stanislaus
Sutter
Tehama
Trinity
Tulare
Tuolumne
Ventura
Yolo
Yuba
12-15
12-17
12-24
12-31
01-07
01-14
01-21
01-28
02-04
02-11
02-18
02-25
03-04
03-11
03-18
03-25
04-01
04-08
04-15
04-22
100
90
80
70
60
50
40
30
20
10
0
NaN

From June 22 to August 12, I worked on a gearshift. This project really was the culmination of all my knowledge and experience, using everything I learned from the beginning of my journey. I extensively used variables and patterns - something I learned at the start of my






Cumulative Percentage Of J % J Doses Given Per Day Up To 22-Apr-2021
Alameda
Alpine
Amador
Butte
Calaveras
Colusa
Contra Costa
Del Norte
El Dorado
Fresno
Glenn
Humboldt
Imperial
Inyo
Kern
Kings
Lake
Lassen
Los Angeles
Madera
Marin
Mariposa
Mendocino
Merced
Modoc
Mono
Monterey
Napa
Nevada
Orange
Placer
Plumas
Riverside
Sacramento
San Benito
San Bernardino
San Diego
San Francisco
San Joaquin
San Luis Obispo
San Mateo
Santa Barbara
Santa Clara
Santa Cruz
Shasta
Sierra
Siskiyou
Solano
Sonoma
Stanislaus
Sutter
Tehama
Trinity
Tulare
Tuolumne
Ventura
Yolo
Yuba
12-15
12-17
12-24
12-31
01-07
01-14
01-21
01-28
02-04
02-11
02-18
02-25
03-04
03-11
03-18
03-25
04-01
04-08
04-15
04-22
100
90
80
70
60
50
40
30
20
10
0
NaN


Percentage Of Cumulative Doses Given By Manufacturer Up To 22-Apr-2021
Alameda
Alpine
Amador
Butte
Calaveras
Colusa
Contra Costa
Del Norte
El Dorado
Fresno
Glenn
Humboldt
Imperial
Inyo
Kern
Kings
Lake
Lassen
Los Angeles
Madera
Marin
Mariposa
Mendocino
Merced
Modoc
Mono
Monterey
Napa
Nevada
Orange
Placer
Plumas
Riverside
Sacramento
San Benito
San Bernardino
San Diego
San Francisco
San Joaquin
San Luis Obispo
San Mateo
Santa Barbara
Santa Clara
Santa Cruz
Shasta
Sierra
Siskiyou
Solano
Sonoma
Stanislaus
Sutter
Tehama
Trinity
Tulare
Tuolumne
Ventura
Yolo
Yuba
Pfizer
Moderna
J & J
Cumulative
100
90
80
70
60
50
40
30
20
10
0

The data often needed to go through a lot of sorting and management before being plotted properly. Often, if the counties weren't sorted alphabetically, they were sorted by population.

```
%%Sorting
    function [sortedNames] = nameSorting(table,catergory,number)
        top = sortrows(table, catergory, "descend");
        top([number+1:height(top)],:) = [];
        topNames = top.County;
        sortedNames = categorical(topNames);
    end
```

```
%Puts the most [insert variable] counties first
%Everything after the top few counties is erased
%Gathers a list of names of those counties.
% The previous line gathered ALL information about the county.
%makes the array usable elsewhere
```

When recording vaccine data by date, the same counties or places are bound to be logged in the same data sheet more than once. This function finds the indecies of the same county, and keeps track of them when grabbing data to log for it.

```
%%Indexing
    function [sortedIndex,indexOnlyTable] = indexSorting(table1,table2,counties,catergory,number,mostRecent)
        indecies = [];
        if mostRecent == "yes"
            for i = 1:number
                indecies(i) = max(find(counties == table2(i)));
            end
        end
        newTable = table1;
        if height(table1)>number
            removedIndecies = 1:1:height(table1);
            removedIndecies(indecies) = [];
            newTable(removedIndecies,:) = [];
        end
        newTable = sortrows(newTable, catergory, "descend");
        sortedIndex = indecies;
        indexOnlyTable = newTable;
    end
```

```
%Creating an array of the indexes of the counties

%Finds the most recent index of the county in the vaccine table
```

When finished using one table, often times, certain columns aren't needed anymore. Some data tables have ALMOST similar columns, with just a couple in the way so they don't line up perfectly. This function removes those columns, and shifts everything accordingly. That way, the tables do align their columns.

```
%%Date Remover
    function [newTable] = dateRemover(table,dateVar,dates)
        indecies = [];
        for i = 1:height(table)
            if ismember(table{i,dateVar},dates) == false
                indecies = [indecies,i];
            end
        end
        table(indecies,:) = [];
        newTable = table;
    end
```

Finally, with all counties tracked by their indecies on each table, and with table columns aligned, each county can finally be plotted along a heat map, per date.

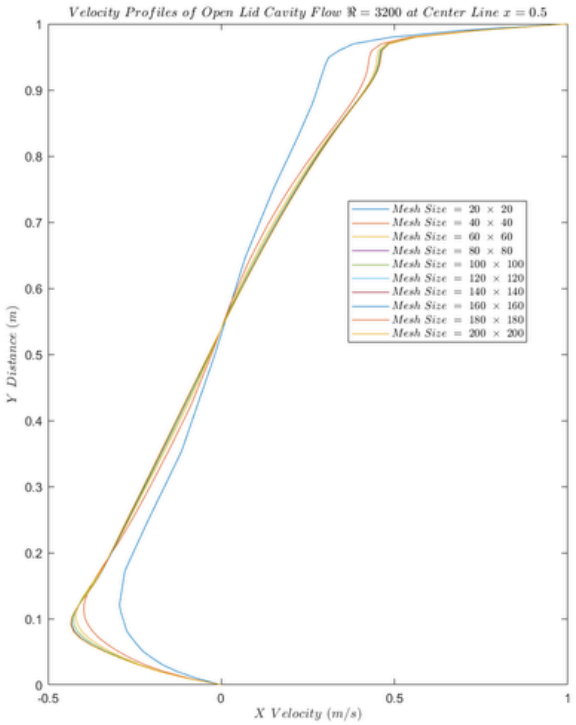
```
%%Heatmap Along Dates
    function [matrixHeat] = matrixHeatmapDates(popTable,dates,counties,names,vaxTable,tableDates,variable,partTitle)
        fully_vac_day = NaN(height(popTable),length(dates));
        for i = 1:height(popTable)
            countyIndecies = find(counties == names(i));
            factorIndecies = [];
            for ii = 1:length(countyIndecies)
                for iii = 1:length(dates)
                    if vaxTable{countyIndecies(ii),tableDates} == dates(iii)
                        factorIndecies(iii) = iii;
                    end
                end
            end
            factorIndecies = factorIndecies(factorIndecies~=0);
            for ii = 1:length(factorIndecies)
                fully_vac_day(i,factorIndecies(ii)) = vaxTable{countyIndecies(ii),variable};
            end
        end
        heatmap(dates,names,fully_vac_day,"Colormap",turbo);
        caxis([0 100]);
        title(partTitle + " Up To " + datestr(max(dates)))
    end
```

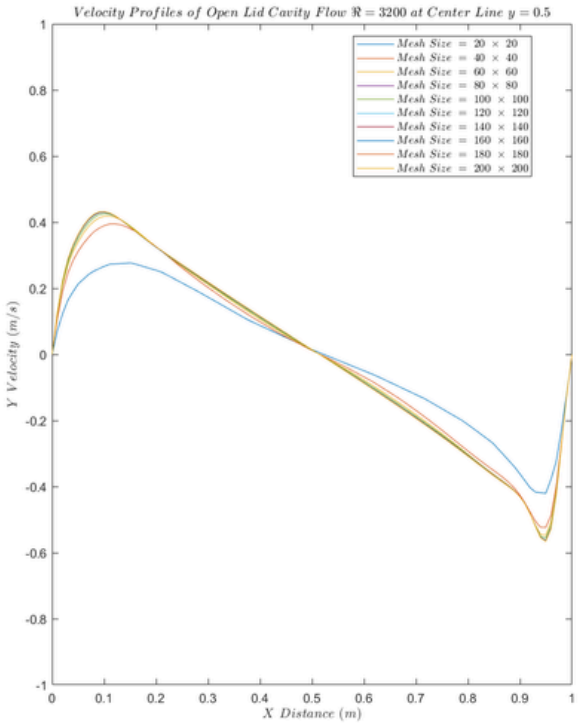
CFD LID DRIVEN CAVITY FLOW POST PROCESSING — APRIL 2024

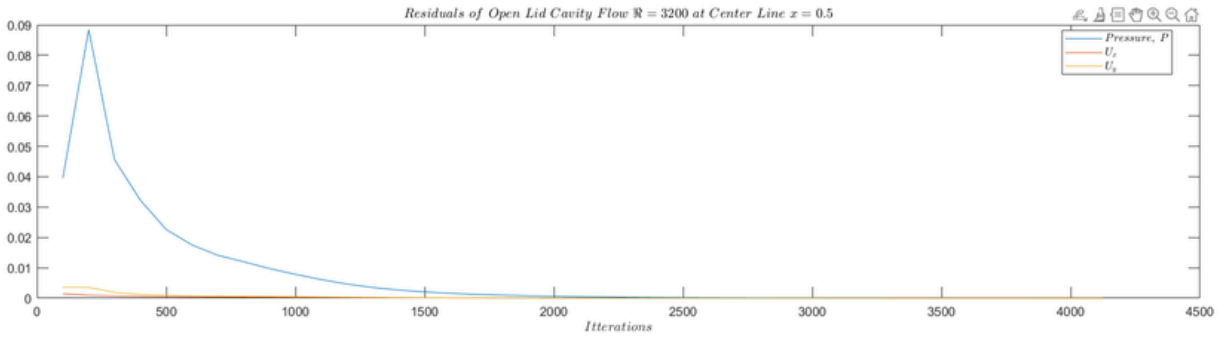
This was my first time using a real CFD software, that wasn't just something in a CAD package like Solidworks. In my CFD class, we used OpenFOAM, a Linux based solver that allowed serious customization and settings. The class was very derivation intensive, before we were allowed to actually apply software to the problems.

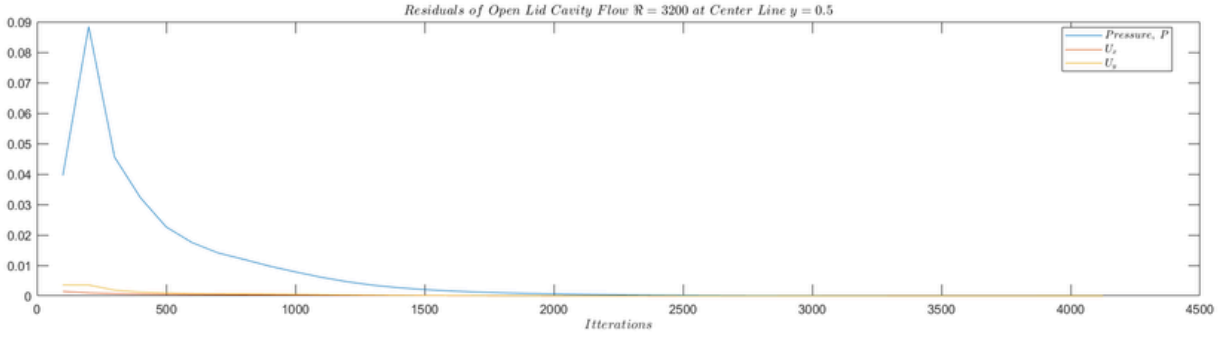
We ran a convection flow over a cavity with an open lid in OpenFOAM, playing with different settings, such as the mesh size and solver (Crank-Nicolson, Euler, Steady State), and would sample different places, ie the centerline, or the entire field. OpenFOAM would return tables of data, often in .txt or .dat files, and it would sort them in a predictable, repeated structure of folders

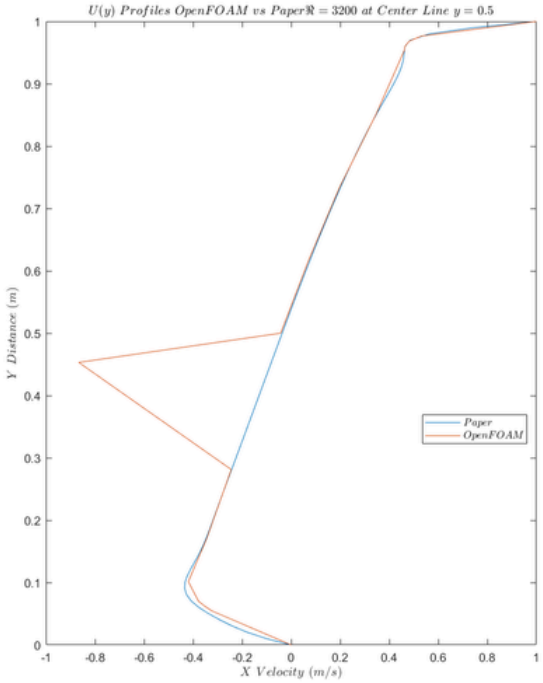
Shown below is how MATLAB represented the data that OpenFOAM spat out. Again, I had to make MATLAB automatically find the files for me, since it would have been too tedious to do by hand, given the number of test cases ran. This was also my first time bash scripting to make OpenFOAM automated.

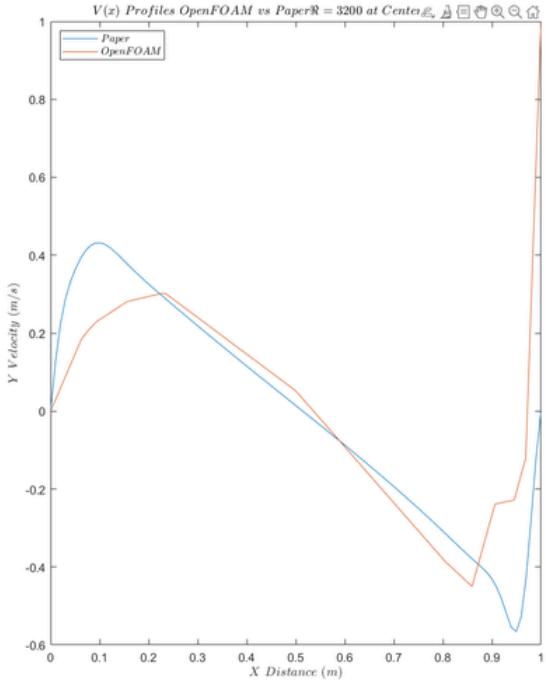






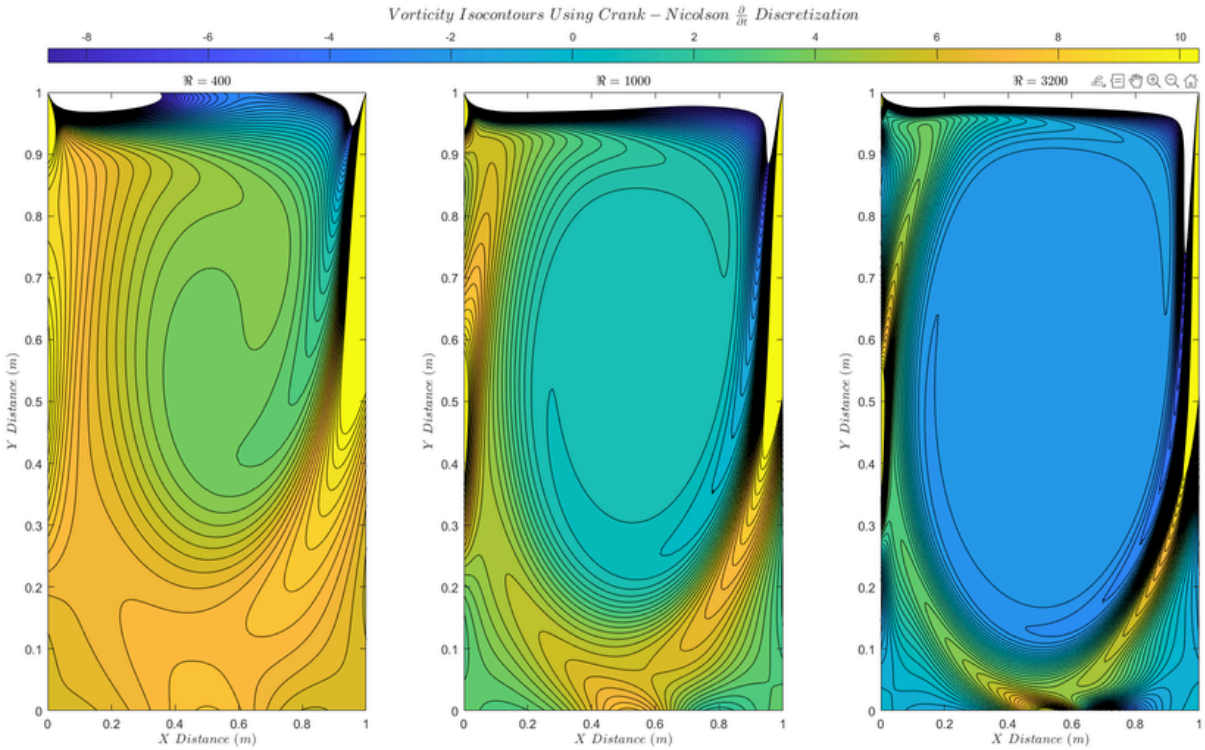






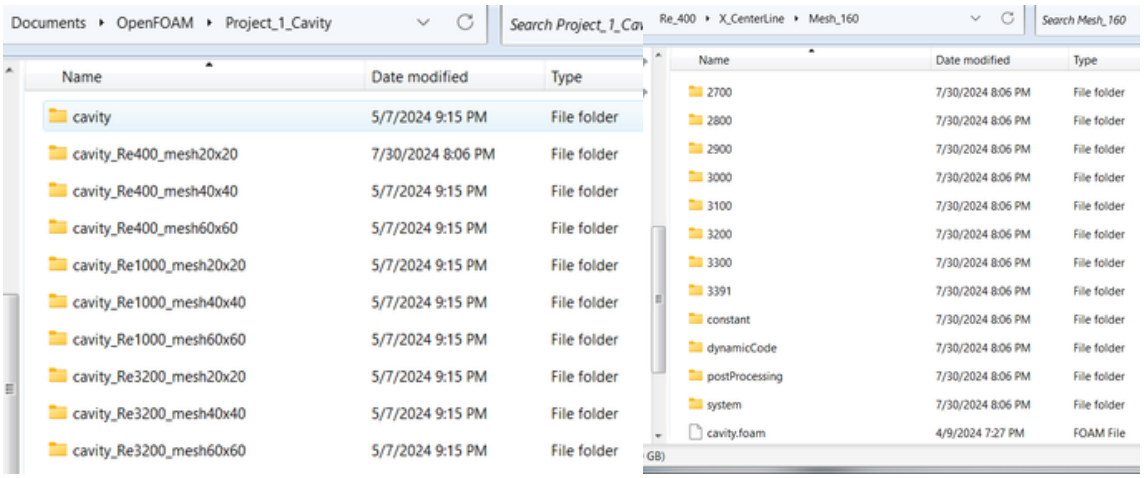
Shown below is the vorticity field of the cavity flow, per discretization scheme. Because the solution became grid independent due to the resolution of the mesh, the solver type was irrelevant, and as seen below, all produced very similar answers.

This was my first time using MATLAB's contour map to plot 2D data, and it looked very nice.



Shown below is the file structure that OpenFOAM generates. As seen, there are several test cases, and even within those test cases, lie folders per snapshot of time. It would have been impractical to manually type in all the file names myself, so I had MATLAB automate the file name inputs for me, using string concatenations.

```
fileNamesXY_crank{1,i} = strcat(['C:/Users/n8dsa/Documents/OpenFOAM/Project_1_Cavity/' ...
    'Re_', num2str( Re(i) ) ,'/X_CenterLine/Mesh_' , num2str(meshSizes(end)) , '-Crank_Nicolson/postProcessing/graphUniform/' ...
    num2str( crankNicConvergenceCases(i) ),'/line.xy']);
fileNamesXY_euler{1,i} = strcat(['C:/Users/n8dsa/Documents/OpenFOAM/Project_1_Cavity/' ...
    'Re_', num2str( Re(i) ) ,'/X_CenterLine/Mesh_' , num2str(meshSizes(end)) , '-Euler/postProcessing/graphUniform/' ...
    num2str( eulerConvergenceCases(i) ),'/line.xy']);
A_xy_otherCases{1,i} = readtable(fileNamesXY_crank{1,i},'filetype', 'text');
A_xy_otherCases{2,i} = readtable(fileNamesXY_euler{1,i},'filetype', 'text');

fileNamesXY_crank{2,i} = strcat(['C:/Users/n8dsa/Documents/OpenFOAM/Project_1_Cavity/' ...
    'Re_', num2str( Re(i) ) ,'/Y_CenterLine/Mesh_' , num2str(meshSizes(end)) , '-Crank_Nicolson/postProcessing/graphUniform/' ...
    num2str( crankNicConvergenceCases(i) ),'/line.xy']);
fileNamesXY_euler{2,i} = strcat(['C:/Users/n8dsa/Documents/OpenFOAM/Project_1_Cavity/' ...
    'Re_', num2str( Re(i) ) ,'/Y_CenterLine/Mesh_' , num2str(meshSizes(end)) , '-Euler/postProcessing/graphUniform/' ...
    num2str( eulerConvergenceCases(i) ),'/line.xy']);
B_xy_otherCases{1,i} = readtable(fileNamesXY_crank{2,i},'filetype', 'text');
B_xy_otherCases{2,i} = readtable(fileNamesXY_euler{2,i},'filetype', 'text');
```



C: > Users > n8dsa > Documents > OpenFOAM > Project_1_Cavity > Re_400 > X_CenterLine > Mesh_200-Crank_Nicolson > postProcessing > cutPlaneSurface > 4936 > cutPlane.xy

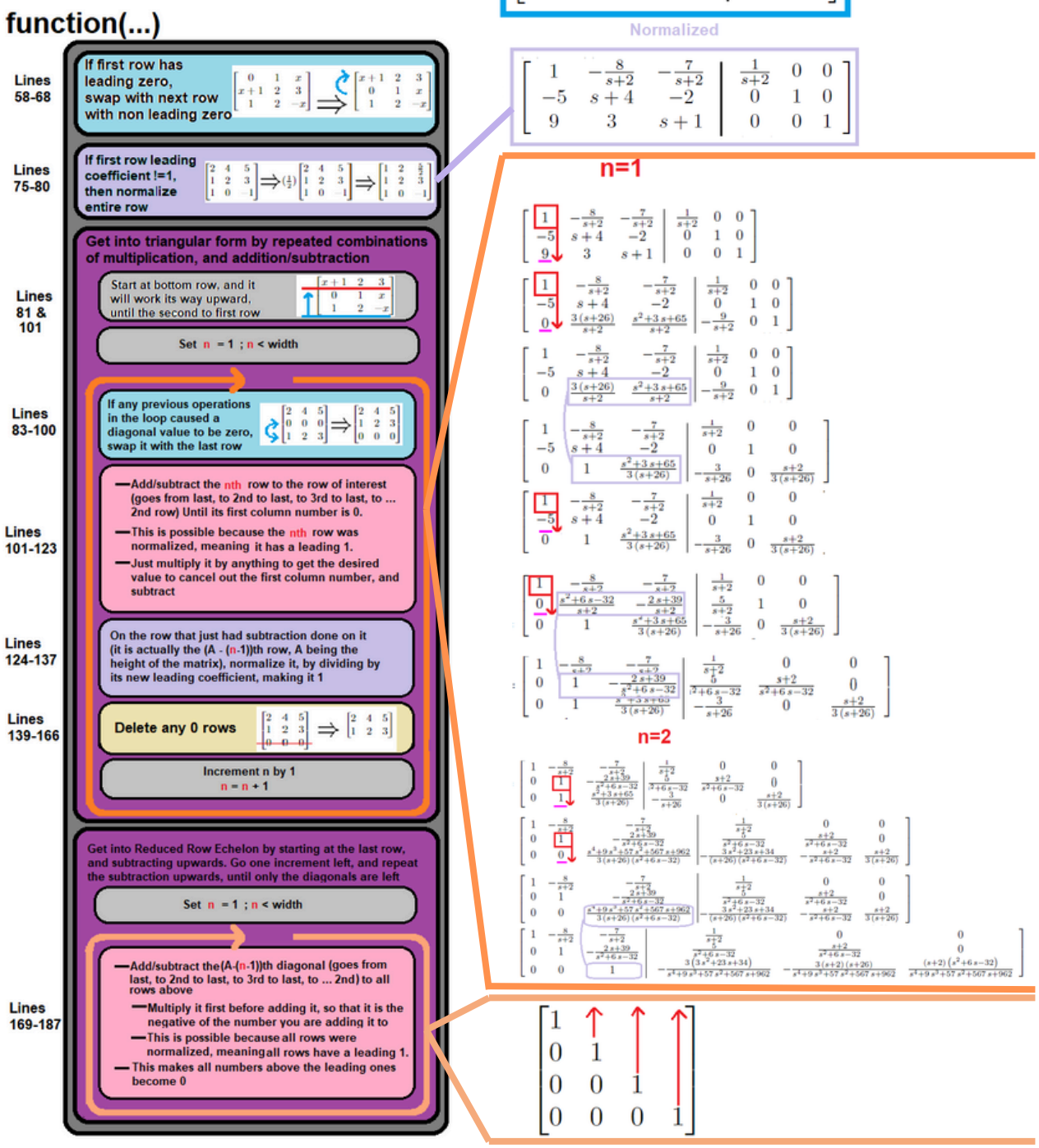
#	point_x	point_y	point_z	U_x	U_y	U_z	vorticity_x	vorticity_y	vorticity_z
2	0.00127397	0.000783158	0.01	-1.07781e-07	4.79364e-09	0	0	0	9.00521e-05
3	0	0.000783158	0.01	0	0	0	0	0	-2.26646e-06
4	0	0	0.01	0	0	0	0	0	6.8675e-05
5	0.00127397	0	0.01	0	0	0	0	0	0.000226419
6	0.00257792	0.000783158	0.01	-1.86714e-07	2.73301e-08	0	0	0	0.000143
7	0.00257792	0	0.01	0	0	0	0	0	0.000363921
8	0.00391255	0.000783158	0.01	-2.05989e-07	5.01064e-09	0	0	0	0.000107608
9	0.00391255	0	0.01	0	0	0	0	0	0.000408893
10	0.00527859	0.000783158	0.01	-1.33905e-07	-3.37434e-08	0	0	0	-2.85859e-05
11	0.00527859	0	0.01	0	0	0	0	0	0.000355877
12	0.00667677	0.000783158	0.01	6.83127e-08	-7.75346e-08	0	0	0	-0.000310904
13	0.00667677	0	0.01	0	0	0	0	0	0.000125601
14	0.00810785	0.000783158	0.01	4.14405e-07	-1.2226e-07	0	0	0	-0.000759771
15	0.00810785	0	0.01	0	0	0	0	0	-0.000301713
16	0.0095726	0.000783158	0.01	9.10232e-07	-1.65575e-07	0	0	0	-0.00138476
17	0.0095726	0	0.01	0	0	0	0	0	-0.000931972
18	0.0110718	0.000783158	0.01	1.55662e-06	-2.06243e-07	0	0	0	-0.00218897
19	0.0110718	0	0.01	0	0	0	0	0	-0.00176454
20	0.0126063	0.000783158	0.01	2.351e-06	-2.43617e-07	0	0	0	-0.00317095
21	0.0126063	0	0.01	0	0	0	0	0	-0.0027945
22	0.0141769	0.000783158	0.01	3.2885e-06	-2.77365e-07	0	0	0	-0.00432605
23	0.0141769	0	0.01	0	0	0	0	0	-0.00401418
24	0.0157845	0.000783158	0.01	4.36264e-06	-3.07338e-07	0	0	0	-0.00564733
25	0.0157845	0	0.01	0	0	0	0	0	-0.00541406
26	0.0174299	0.000783158	0.01	5.56569e-06	-3.33493e-07	0	0	0	-0.00712612
27	0.0174299	0	0.01	0	0	0	0	0	-0.00698327
28	0.019114	0.000783158	0.01	6.88898e-06	-3.55855e-07	0	0	0	-0.00875234
29	0.019114	0	0.01	0	0	0	0	0	-0.00870985

Although powerful and versatile, the Gauss-Jordan method is very tedious and repetitive. My professors always wanted me to show every single step of work when solving these, especially when dealing with transfer functions. Not wanting to do this repetitive task by hand, and not wanting to use the calculators online, I sought to write my own program that would not only solve the problem for me, but show each step itself.

This allowed me to breeze through problems without any errors. The one in October was for purely numerical matricies, while the one in April allowed for variables. (It uses a floating point comparison in the numerical version to see if a value is close enough to 0)

I made a function in MATLAB, and just called it every time I needed a problem done. Below, is the algorithm, and on the next pages is a worked example

```
sI_A = sym(  [[s,-1,0];[0,s-1,-2];[5,4,s+3]]  );
Ident = sym(  [[1,0,0];[0,1,0];[0,0,1]]  );

%%function call and work
format rat
[~,sI_A_Inv] = gaussJordanN(sI_A,Ident,"Problem 3");
B = [0;1;0];
C = [1,0,1];
transferFunction = simplify( simplifyFraction( C * sI_A_Inv * B ) );
```



Below, is a fully worked example, it which I had to find a vector of transfer functions., in order to find the transfer function for a single input system.

Problem 3

$$\mathrm{L} = \begin{bmatrix} s+2 & -8 & -7 \\ -5 & s+4 & -2 \\ 9 & 3 & s+1 \end{bmatrix}$$

$$\mathrm{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\mathrm{M}_1(1/(s+2))...InMatlab : R(1,:) = (1/L(1,1)) * R(1,:); L(1,:) = 1/L(1,1)) * L(1,:)$

$$\mathrm{L} = \begin{bmatrix} 1 & -\frac{8}{s+2} & -\frac{7}{s+2} \\ -5 & s+4 & -2 \\ 9 & 3 & s+1 \end{bmatrix}$$

$$\mathrm{R} = \begin{bmatrix} \frac{1}{s+2} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\mathrm{A}_1 toA_3(-9)...InMatlab : R(3,:) = -1 * L(3,1) * R(1,:); L(3,:) = -1 * L(3,1) * L(1,:)$

$$\mathrm{L} = \begin{bmatrix} 1 & -\frac{8}{s+2} & -\frac{7}{s+2} \\ -5 & s+4 & -2 \\ 0 & \frac{3\,(s+26)}{s+2} & \frac{s^2+3\,s+65}{s+2} \end{bmatrix}$$

$$\mathrm{R} = \begin{bmatrix} \frac{1}{s+2} & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{9}{s+2} & 0 & 1 \end{bmatrix}$$

$\mathrm{M}_3(1/(3 * (s+26))/(s+2))...InMatlab : R(3,:) = (1/L(3,)) * R(3,:); L(3,:) = (1/L(3,)) * L(3,:)$

$$\mathrm{L} = \begin{bmatrix} 1 & -\frac{8}{s+2} & -\frac{7}{s+2} \\ -5 & s+4 & -2 \\ 0 & 1 & \frac{s^2+3\,s+65}{3\,(s+26)} \end{bmatrix}$$

$$\mathrm{R} = \begin{bmatrix} \frac{1}{s+2} & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{3}{s+26} & 0 & \frac{s+2}{3\,(s+26)} \end{bmatrix}$$

$\mathrm{A}_1 toA_2(5)...InMatlab : R(2,:) = -1 * L(2,1) * R(1,:); L(2,:) = -1 * L(2,1) * L(1,:)$

$$\mathrm{L} = \begin{bmatrix} 1 & -\frac{8}{s+2} & -\frac{7}{s+2} \\ 0 & \frac{s^2+6\,s-32}{s+2} & -\frac{2\,s+39}{s+2} \\ 0 & 1 & \frac{s^2+3\,s+65}{3\,(s+26)} \end{bmatrix}$$

$$\mathrm{R} = \begin{bmatrix} \frac{1}{s+2} & 0 & 0 \\ \frac{5}{s+2} & 1 & 0 \\ -\frac{3}{s+26} & 0 & \frac{s+2}{3\,(s+26)} \end{bmatrix}$$

$\mathrm{M}_2(1/(6 * s + s^2 - 32)/(s+2))...InMatlab : R(2,:) = (1/L(2,)) * R(2,:); L(2,:) = (1/L(2,)) * L(2,:)$

$$\mathrm{L} = \begin{bmatrix} 1 & -\frac{8}{s+2} & -\frac{7}{s+2} \\ 0 & 1 & -\frac{2\,s+39}{s^2+6\,s-32} \\ 0 & 1 & \frac{s^2+3\,s+65}{3\,(s+26)} \end{bmatrix}$$

$$\mathrm{R} = \begin{bmatrix} \frac{1}{s+2} & 0 & 0 \\ \frac{5}{s^2+6\,s-32} & \frac{s+2}{s^2+6\,s-32} & 0 \\ -\frac{3}{s+26} & 0 & \frac{s+2}{3\,(s+26)} \end{bmatrix}$$

$\mathrm{A}_2 toA_3(-1)...InMatlab : R(3,:) = -1 * L(3,2) * R(2,:); L(3,:) = -1 * L(3,2) * L(2,:)$

$$\mathrm{L} = \begin{bmatrix} 1 & -\frac{8}{s+2} & -\frac{7}{s+2} \\ 0 & 1 & -\frac{2\,s+39}{s^2+6\,s-32} \\ 0 & 0 & \frac{s^4+9\,s^3+57\,s^2+567\,s+962}{3\,(s+26)\,(s^2+6\,s-32)} \end{bmatrix}$$

$$\mathrm{R} = \begin{bmatrix} \frac{1}{s+2} & 0 & 0 \\ \frac{5}{s^2+6\,s-32} & \frac{s+2}{s^2+6\,s-32} & 0 \\ -\frac{3\,s^2+23\,s+34}{(s+26)\,(s^2+6\,s-32)} & -\frac{s+2}{s^2+6\,s-32} & \frac{s+2}{3\,(s+26)} \end{bmatrix}$$

$\mathrm{M}_3(1/(567 * s + 57 * s^2 + 9 * s^3 + s^4 + 962)/(3 * (s+26) * (6 * s + s^2 - 32)))...InMatlab : R(3,:) = (1/L(3,)) * R(3,:); L(3,:) = (1/L(3,)) * L(3,:)$

$$\mathrm{L} = \begin{bmatrix} 1 & -\frac{8}{s+2} & -\frac{7}{s+2} \\ 0 & 1 & -\frac{2\,s+39}{s^2+6\,s-32} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{R} = \begin{bmatrix} \frac{1}{s+2} & 0 & 0 \\ \frac{5}{s^2+6\,s-32} & \frac{s+2}{s^2+6\,s-32} & 0 \\ -\frac{3\left(3\,s^2+23\,s+34\right)}{s^4+9\,s^3+57\,s^2+567\,s+962} & -\frac{3\,(s+2)\,(s+26)}{s^4+9\,s^3+57\,s^2+567\,s+962} & \frac{(s+2)\left(s^2+6\,s-32\right)}{s^4+9\,s^3+57\,s^2+567\,s+962} \end{bmatrix}$$

$A_2 to A_1(8/(s+2))...InMatlab: R(1,:) = -1 * L(1,2)/L(2,2) * R(1,:); L(1,:) = -1 * L(1,2)/L(2,2) * L(1,:)$

$$\mathrm{L} = \begin{bmatrix} 1 & 0 & -\frac{7\,s+44}{s^2+6\,s-32} \\ 0 & 1 & -\frac{2\,s+39}{s^2+6\,s-32} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{R} = \begin{bmatrix} \frac{s+4}{s^2+6\,s-32} & \frac{8}{s^2+6\,s-32} & 0 \\ \frac{5}{s^2+6\,s-32} & \frac{s+2}{s^2+6\,s-32} & 0 \\ -\frac{3\,(3\,s+17)}{s^3+7\,s^2+43\,s+481} & -\frac{3\,(s+26)}{s^3+7\,s^2+43\,s+481} & \frac{s^2+6\,s-32}{s^3+7\,s^2+43\,s+481} \end{bmatrix}$$

$A_3 to A_1((7 * s + 44)/(6 * s + s^2 - 32))...InMatlab: R(1,:) = -1 * L(1,3)/L(3,3) * R(1,:); L(1,:) = -1 * L(1,3)/L(3,3) * L(1,:)$

$$\mathrm{L} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -\frac{2\,s+39}{s^2+6\,s-32} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{R} = \begin{bmatrix} \frac{s^2+5\,s+10}{s^3+7\,s^2+43\,s+481} & \frac{8\,s-13}{s^3+7\,s^2+43\,s+481} & \frac{7\,s+44}{s^3+7\,s^2+43\,s+481} \\ \frac{5}{s^2+6\,s-32} & \frac{s+2}{s^2+6\,s-32} & 0 \\ -\frac{3\,(3\,s+17)}{s^3+7\,s^2+43\,s+481} & -\frac{3\,(s+26)}{s^3+7\,s^2+43\,s+481} & \frac{s^2+6\,s-32}{s^3+7\,s^2+43\,s+481} \end{bmatrix}$$

$A_3 to A_2((2 * s + 39)/(6 * s + s^2 - 32))...InMatlab: R(2,:) = -1 * L(2,3)/L(3,3) * R(2,:); L(2,:) = -1 * L(2,3)/L(3,3) * L(2,:)$

$$\mathrm{L} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{R} = \begin{bmatrix} \frac{s^2+5\,s+10}{s^3+7\,s^2+43\,s+481} & \frac{8\,s-13}{s^3+7\,s^2+43\,s+481} & \frac{7\,s+44}{s^3+7\,s^2+43\,s+481} \\ \frac{5\,s-13}{s^3+7\,s^2+43\,s+481} & \frac{s^2+3\,s+65}{s^3+7\,s^2+43\,s+481} & \frac{2\,s+39}{s^3+7\,s^2+43\,s+481} \\ -\frac{3\,(3\,s+17)}{s^3+7\,s^2+43\,s+481} & -\frac{3\,(s+26)}{s^3+7\,s^2+43\,s+481} & \frac{s^2+6\,s-32}{s^3+7\,s^2+43\,s+481} \end{bmatrix}$$

$$[sI - A]^{-1} = \begin{bmatrix} \frac{s^2+5\,s+10}{s^3+7\,s^2+43\,s+481} & \frac{8\,s-13}{s^3+7\,s^2+43\,s+481} & \frac{7\,s+44}{s^3+7\,s^2+43\,s+481} \\ \frac{5\,s-13}{s^3+7\,s^2+43\,s+481} & \frac{s^2+3\,s+65}{s^3+7\,s^2+43\,s+481} & \frac{2\,s+39}{s^3+7\,s^2+43\,s+481} \\ -\frac{3\,(3\,s+17)}{s^3+7\,s^2+43\,s+481} & -\frac{3\,(s+26)}{s^3+7\,s^2+43\,s+481} & \frac{s^2+6\,s-32}{s^3+7\,s^2+43\,s+481} \end{bmatrix}$$

$$\frac{Y(s)}{U(s)} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{s^2+5\,s+10}{s^3+7\,s^2+43\,s+481} & \frac{8\,s-13}{s^3+7\,s^2+43\,s+481} & \frac{7\,s+44}{s^3+7\,s^2+43\,s+481} \\ \frac{5\,s-13}{s^3+7\,s^2+43\,s+481} & \frac{s^2+3\,s+65}{s^3+7\,s^2+43\,s+481} & \frac{2\,s+39}{s^3+7\,s^2+43\,s+481} \\ -\frac{3\,(3\,s+17)}{s^3+7\,s^2+43\,s+481} & -\frac{3\,(s+26)}{s^3+7\,s^2+43\,s+481} & \frac{s^2+6\,s-32}{s^3+7\,s^2+43\,s+481} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\frac{Y(s)}{U(s)} = \frac{5\,s-91}{s^3+7\,s^2+43\,s+481}$$