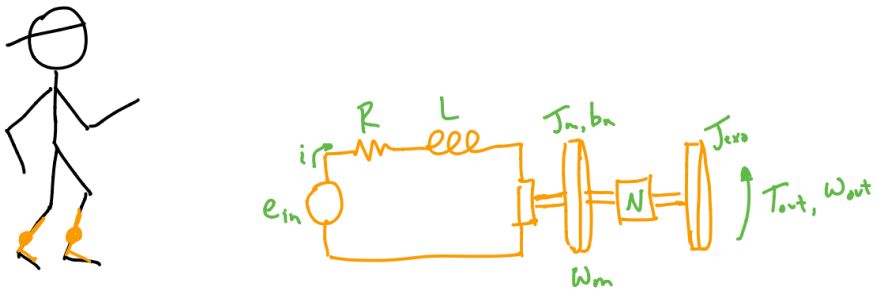
MATLAB Problem 5

A quick note before you start – you will answer all questions for this problem within this Word document. When you're finished, please save the document as a .pdf, which you will upload directly to Gradescope. **Please also upload any code you use to BruinLearn.** This can be done through the same BruinLearn "assignment" from which you downloaded the problem set files. You should turn in all of the .m files that you use in this problem.

We have arrived at the final MATLAB problem of the quarter. In this assignment, you're going to (drumroll please)…. design a robotic exoskeleton! Exoskeletons work in parallel with the human body to augment or restore movement ability. We're going to focus our efforts on augmentation, by considering the first autonomous exoskeleton to successfully reduce the metabolic cost of walking. This exoskeleton was first developed by Luke Mooney and Hugh Herr at MIT (check out the journal article PDFs for more) and is now being commercialized by a company called Dephy. They are currently claiming an average of 20% reduction in metabolic cost during walking, which means someone wearing the device could walk ten miles while only feeling like they had walked eight. Applications for such a device include any profession that involves walking for long periods, such as nursing, waiting tables, or military service.

5a: Build a mathematical model and analytically solve the resulting system of equations.

As always, our first step is to model the system. We can think of the exoskeleton as an electrical subsystem (the motor) working in series with a mechanical subsystem (the transmission and the exoskeleton's joint). Together, these two subsystems look something like this:



where the motor has terminal resistance R, terminal inductance L, current $i(t)$, input voltage $e_{in}(t)$, rotor inertia J_m, damping b_m, speed constant K_v, and torque constant K_T. The output of the motor is connected via a transmission of ratio N to the exoskeleton, which has inertia J_{exo}, and produces a net torque at the output $T_{out}(t)$. The motor shaft moves with velocity $\omega_m(t)$, and the output velocity of the whole exoskeleton is $\omega_{out}(t)$; these two values are related by the transmission ratio ($\omega_m = N\omega_{out}$).

Our first task is to find the system of differential equations that models this exoskeleton. Some of these concepts may be new, so I'll help you get started. The electrical equation for the motor (as we derived in the first few weeks of class) is:

$$e_{in} = Ri + L\frac{di}{dt} + \frac{1}{K_v}\omega_m$$

Take a moment to convince yourself that this equation makes sense before you move on.

The mechanical equation for the exoskeleton is:

$$K_T i = J_m\dot{\omega}_m + b_m\omega_m + \frac{1}{N}(J_{exo}\dot{\omega}_{out} + T_{out})$$

where the $K_T i$ term represents the torque produced by the motor (call this T_m). We can represent torque this way because motors produce torque proportional to the current through their windings ($T_m = K_T i$). Notice here

that the inertial term at the output $J_{exo}\dot{\omega}_{out}$ and the output torque T_{out} are both reduced by the transmission ratio N. Because the inertia of the exo J_{exo} is small and divided by a factor of N, the $J_{exo}\dot{\omega}_{out}$ term is negligible and our new equation becomes:

$$K_T i = J_m \dot{\omega}_m + b_m \omega_m + \frac{1}{N} T_{out}$$

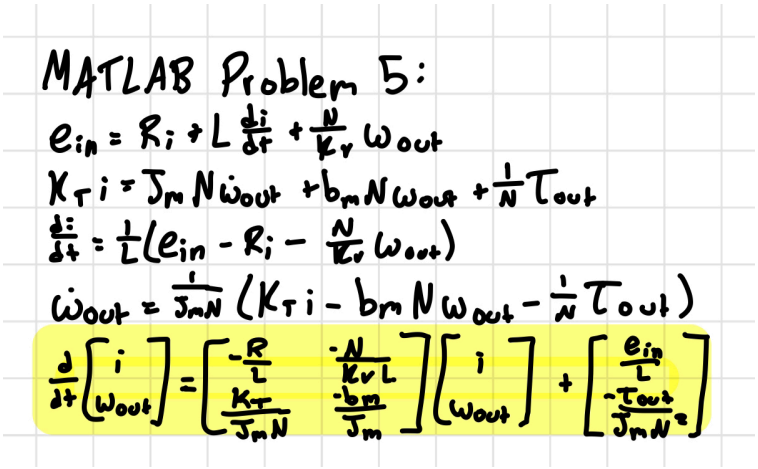
Again, take a moment to convince yourself that this equation makes sense before you move on.

One common way to design a motorized system is to calculate the current required to create a given torque and position trajectory. This essentially asks the question: "what does my motor have to do to produce the desired output?" To do this, it will first be helpful to transform your equation so that everything is in terms of the output velocity ω_{out} that you're trying to track, rather than the motor velocity ω_m. Note here that transmissions trade velocity for torque; in other words, to get the reduction $\frac{1}{N}$ applied to the output torque, we pay the price by having to move faster ($\omega_m = N\omega_{out}$). Don't worry if this is new – you'll learn all about transmissions in your mechanisms class, if you haven't already. For now, the key takeaway is that our exoskeleton system can be represented in full by the following pair of equations:

$$e_{in} = Ri + L\frac{di}{dt} + \frac{N}{K_v}\omega_{out}$$

$$K_T i = J_m N\dot{\omega}_{out} + b_m N\omega_{out} + \frac{1}{N} T_{out}$$

Put this system of equations into state space (matrix) form. You can write this out by hand, or using an equation editor. Paste your work here.



Next, you'll want to plug in real values for the model's parameters. Pull the necessary constants from the motorSpecs.pdf file that you downloaded with the problem set. These are real values from a real motor, which are very similar to the one used in the original Dephy Exo Boot. Pretty cool! You won't find a damping coefficient b_m on the spec sheet; this is not uncommon, as these values can be difficult to measure.

Throughout this problem, please use $b_m = 0.001\frac{Nm*s}{rad}$. For this part, you can also set $N = 1$; this implies that there is no transmission – in other words, the motor is directly connected to the output. **Hint:** you'll need to do some unit conversion to get everything into SI units (either base units or derived units). I recommend double- and triple-checking your unit conversions – if these are off, the rest of the problem set will be very difficult.

Now that you have real values in place, find the homogeneous solution to the resulting system of equations. To make this homogeneous, we are essentially saying that the motor leads are shorted together ($e_{in}(t) = 0\,V$), and that no torque is applied at the output ($T_{out}(t) = 0\,Nm$). Assume that the output starts with an initial

velocity $\omega_{out} = 10\frac{rad}{s}$ and there is no initial current in the motor windings. **Do this by hand, then take a picture and insert it into this document. Feel free to convert to decimal approximations, where convenient. Tip:** for this step, it is totally acceptable to ask for MATLAB's help in finding the eigenvalues and eigenvectors – check out the eig() function for more on this.

$b_m = 0.001 \frac{Nms}{rad}$, $N = 1$, $R = 1.03\,\Omega$, $L = 0.204\,mH$ $K_T = 44.8\,mNm/A$, $K_V = 213\,rpm/V$
$J_m = 101\,gcm^2$, $e_{in}(t) = 0\,V$, $t_{out}(t) = 0\,Nm$, $\omega_{out}(0) = 10\,rad/s$, $i(0) = 0\,A$

Unit Conversions

$L = 2.04 \times 10^{-4}\,H$

$K_T = 4.48 \times 10^{-2}\,Nm/A$

$K_V = 7.1\pi\,\frac{rad}{Vs}$

$J_m = 1.01 \times 10^{-5}\,kg/m^2$

$$\frac{d}{dt}\begin{bmatrix} i \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{-1.03}{2.04\times10^{-4}} & \frac{-1}{7.1\pi \cdot 2.04\times10^{-4}} \\ \frac{4.48\times10^{-2}}{1.01\times10^{-5}\cdot 1} & \frac{-0.001}{1.01\times10^{-5}\cdot 1} \end{bmatrix}\begin{bmatrix} i \\ \omega \end{bmatrix}$$

$$\frac{d}{dt}\begin{bmatrix} i \\ \omega \end{bmatrix} = \begin{bmatrix} -5049.02 & -219.77 \\ 4435.64 & -99.01 \end{bmatrix}\begin{bmatrix} i \\ \omega \end{bmatrix}$$

$$\begin{bmatrix} i \\ \omega \end{bmatrix} = \vec{s}e^{rt} \quad \frac{d}{dt}\begin{bmatrix} i \\ \omega \end{bmatrix} = r\vec{s}e^{rt}$$

$$(A - rI)\vec{s}e^{rt} = 0$$

$$\det(A - rI) = 0$$

$$\det\left(\begin{bmatrix} -5049.02 - r & -219.77 \\ 4435.64 & -99.01 - r \end{bmatrix}\right) = 0$$

$$(-5049.02 - r)(-99.01 - r) - (-219.77)(4435.65) = 0$$

$$r^2 + 5148.03r + 1474709.06 = 0$$

$r_1 = -4843.6 \qquad r_2 = -304.5$

$$\vec{s}^{(1)} = \begin{bmatrix} -0.7305 \\ 0.6829 \end{bmatrix} \qquad \vec{s}^{(2)} = \begin{bmatrix} 0.0463 \\ -0.9989 \end{bmatrix}$$

$$\begin{bmatrix} i \\ \omega \end{bmatrix} = C_1\begin{bmatrix} -0.7305 \\ 0.6829 \end{bmatrix}e^{-4843.6t} + C_2\begin{bmatrix} 0.0463 \\ -0.9989 \end{bmatrix}e^{-304.5t}$$

$$\begin{bmatrix} 0 \\ 10 \end{bmatrix} = C_1\begin{bmatrix} -0.7305 \\ 0.6829 \end{bmatrix} + C_2\begin{bmatrix} 0.0463 \\ -0.9989 \end{bmatrix}$$

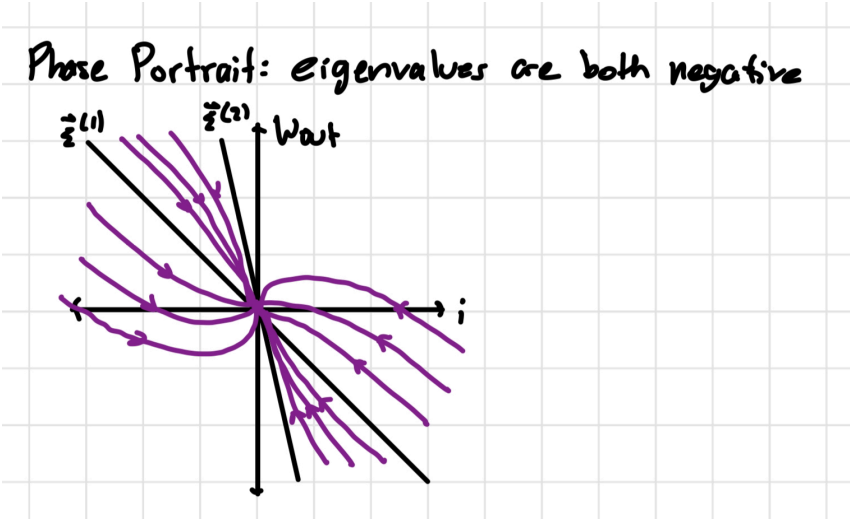
$$\begin{bmatrix} 0 \\ 10 \end{bmatrix} = \begin{bmatrix} -0.7305 & 0.0463 \\ 0.6829 & -0.9989 \end{bmatrix}\begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$$

$C_1 = -0.6628$

$C_2 = -10.4628$

$$\begin{bmatrix} i \\ \omega \end{bmatrix} = -0.6628\begin{bmatrix} -0.7305 \\ 0.6829 \end{bmatrix}e^{-4843.6t} - 10.4628\begin{bmatrix} 0.0463 \\ -0.9989 \end{bmatrix}e^{-304.5t}$$

Based on the eigenvalues and eigenvectors of your solution, sketch a rough phase portrait of your solution, take a picture, and insert it here.



What happens to the current in the motor windings as $t \to \infty$? What about the output velocity?

As t approaches infinity, the current in the motor windings shrinks to 0 and the output velocity also shrinks to 0.

5b: Update your RK4 to handle systems of differential equations, and use it to solve the homogeneous equation.

With a slight modification to your RK4 solver from MATLAB Problem 4, you'll be able to use it to solve any system of first-order differential equations. The key is to update your `yout` array so that it has `n` rows rather than a single row, where `n` is the number of first-order equations in your system. A simple way to do this in code is to use the length of your initial conditions vector `y0` to figure out how many equations you have.

Follow these steps to make this change:

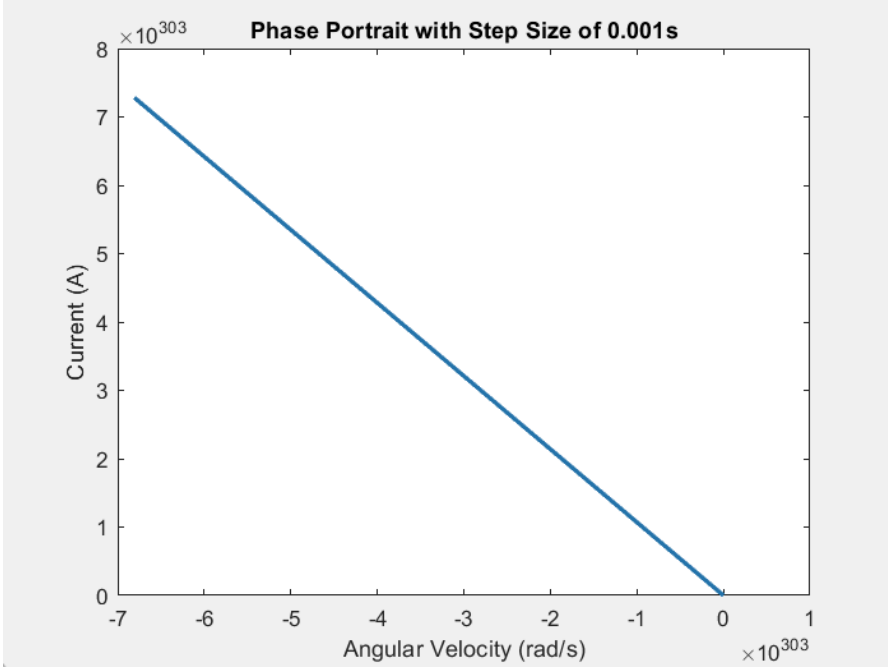
1. In the line where you allocate an array for `yout`, modify your code to create an array of `n` rows, where `n` is the length of the initial conditions vector. The number of columns should be the same as before.
2. In the line where you add your initial condition as the first entry in the `yout` array, you need to modify your code so it can add an arbitrary length initial conditions vector. This line should now look something like this: `yout(:,1) = y0`, which reads in plain English as "set `yout` at every row of the first column equal to the column vector `y0`".
3. `y_n` is now a vector of length `n`, which means that in the line where you set `y_n` equal to the most recent value in the `yout` array, you'll need to grab a whole `nx1` vector rather than just a single value.
4. `y_nplus1` is also a vector of length `n`, which means that in the line where you add `y_nplus1` to the `yout` array, you now need to add the whole `y_nplus1` vector rather than just a single value.

Et voila! If you made these four changes correctly, your RK4 solver code will now be able to handle any system of equations of arbitrary length.

Now that your solver is up to the task, let's use it to learn more about our exoskeleton. We should first do a sanity check by solving the homogeneous case that we tackled by hand in 5a. Put the system of equations from 5a into an ODE function called exoBootsODEfun.m. For now, you only need to code up the homogeneous equation (i.e. both inputs are zero). We won't be passing anything in or out of this ODE function, so all your constants can live inside the ODE function, and should be set to the values from 5a.

Use your RK4solver to solve your ODE function from $0 \leq t \leq 1$, using the same initial conditions as in 5a. Start with a step size of **0.001 s**.

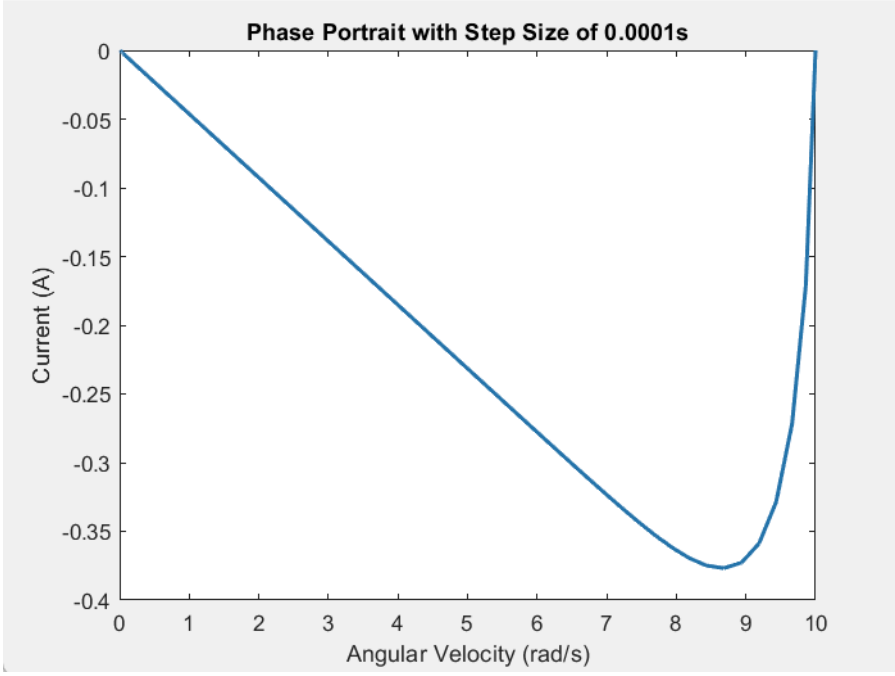
Plot a phase portrait of your result, with angular velocity on the x-axis and current on the y-axis. Label your axes, and apply our standard formatting. As always, save as a .png and insert the image here.



Describe your results. What do you think is happening here?

The phase portrait shows a straight line. There is a linear relationship between the angular velocity and the current. As the angular velocity increases, the current decreases. The step size we chose is too large and we are unable to see the phase portrait behavior/shape. We need to decrease the step size so it does not show a straight line.

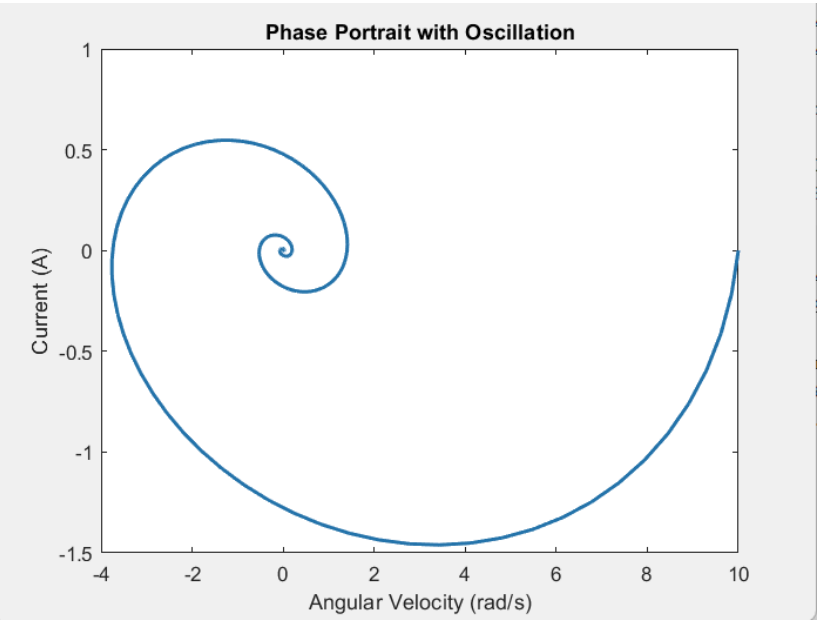
Reduce the step size by a factor of 10, and try again. **Plot a phase portrait of the result, with angular velocity on the x-axis and current on the y-axis. Label your axes, and apply our standard formatting. As always, save as a .png and insert the image here.**



From this phase portrait, does there appear to be any oscillation in the system?

There does not appear to be any oscillation in the system.

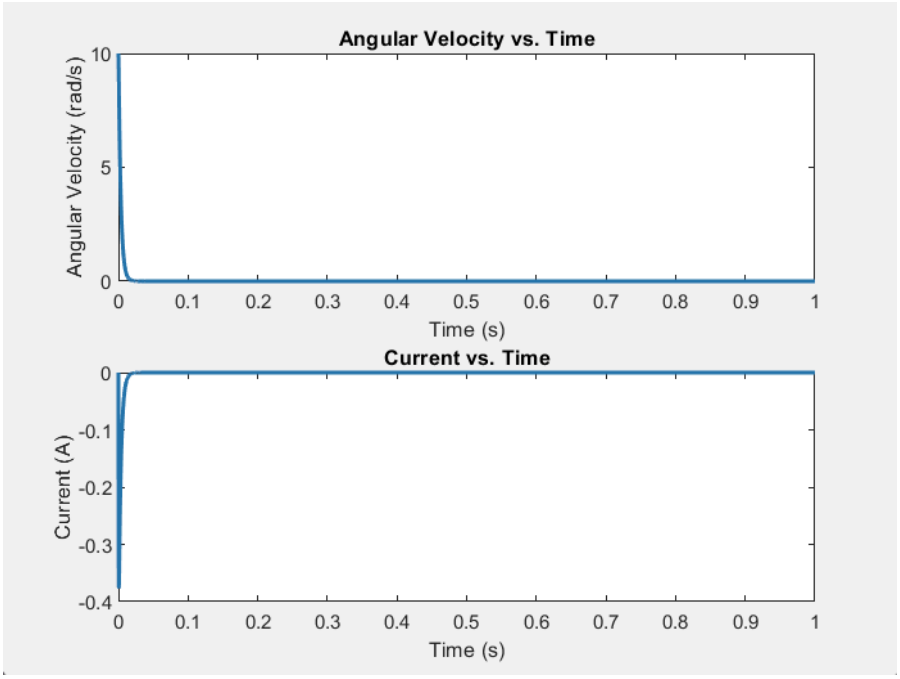
Think for a moment about which parameters you expect to have the greatest impact on whether the system oscillates. Play around with some of the parameters until you find one that you can change to create oscillation in your phase portrait. **Drop the phase portrait showing oscillation here.**



Which parameter did you change to make your system oscillate? Why did you choose this one?

I decreased the terminal resistance of the motor by a factor of 10 to make the system oscillate. I chose this parameter because when I solved the discriminant by hand, I noticed that in order to make the discriminant a negative value, one possibility is to make the terminal resistance a small decimal value. I then changed the terminal resistance value and replotted the phase portrait. A negative discriminant will result in complex conjugate eigenvalues, and thus, oscillation occurs.

Next, set all of your parameters back to their original values, so we can take a look how quickly the system responds when the leads are shorted. In a single figure with 2x1 subplots, plot angular velocity vs. time above current vs. time. **Label your axes, apply our standard formatting, and drop the figure here.**



How quickly does the motor stop moving? What does this imply about what happens when we short a motor's leads together?

The motor quickly stops moving in less than 0.1 seconds (when t=0.0401 seconds to be more precise). This implies that the motor will immediately stop moving when we short a motor's leads together.

5C: Solve the non-homogeneous equation while tracking an output trajectory.

This is where things start to get really fun. Our objective now is to evaluate our exoskeleton's performance while tracking a desired output position and producing a given torque trajectory. The following directions will guide you through this process.

First, you'll want to download the exoBootOutput.mat file, and load in the variables. The torque trajectory contained in the `exoTorque` variable is taken from the optimal trajectories found in a recent well-known exoskeleton manuscript (Zhang et. al., in the attached pdfs.). Note that both the `exoAngles` and `exoTorques` vectors are synced to the same `exoTime` vector.

Now that you've loaded these trajectories, it's time to create your non-homogeneous ODE function:

1. Create a new ODE function called exoBootNHODEfun.m. Code up your full non-homogeneous system of equations from part a, with one minor modification – add a third equation to the system for θ_{out}, defined by the relationship $\frac{d\theta_{out}}{dt} = \omega_{out}$. When this is complete, you should have a system of three equations, with three state variables ($\theta_{out}, \omega_{out}, i$). We do this to make it easy to track the exo's position.
2. Set up exoBootNHODEfun.m so that N can be changed from your main code and passed in. The rest of the parameters can be hard-coded within the ODE function, as in 5b. Note that you will also need to pass in your `exoTorque`, `exoAngle`, and `exoTime` trajectories, as we did in MATLAB Problem 1.
3. Interpolate into `exoTorque` to find the present value of T_{exo}. Do the same with the `exoAngle`.

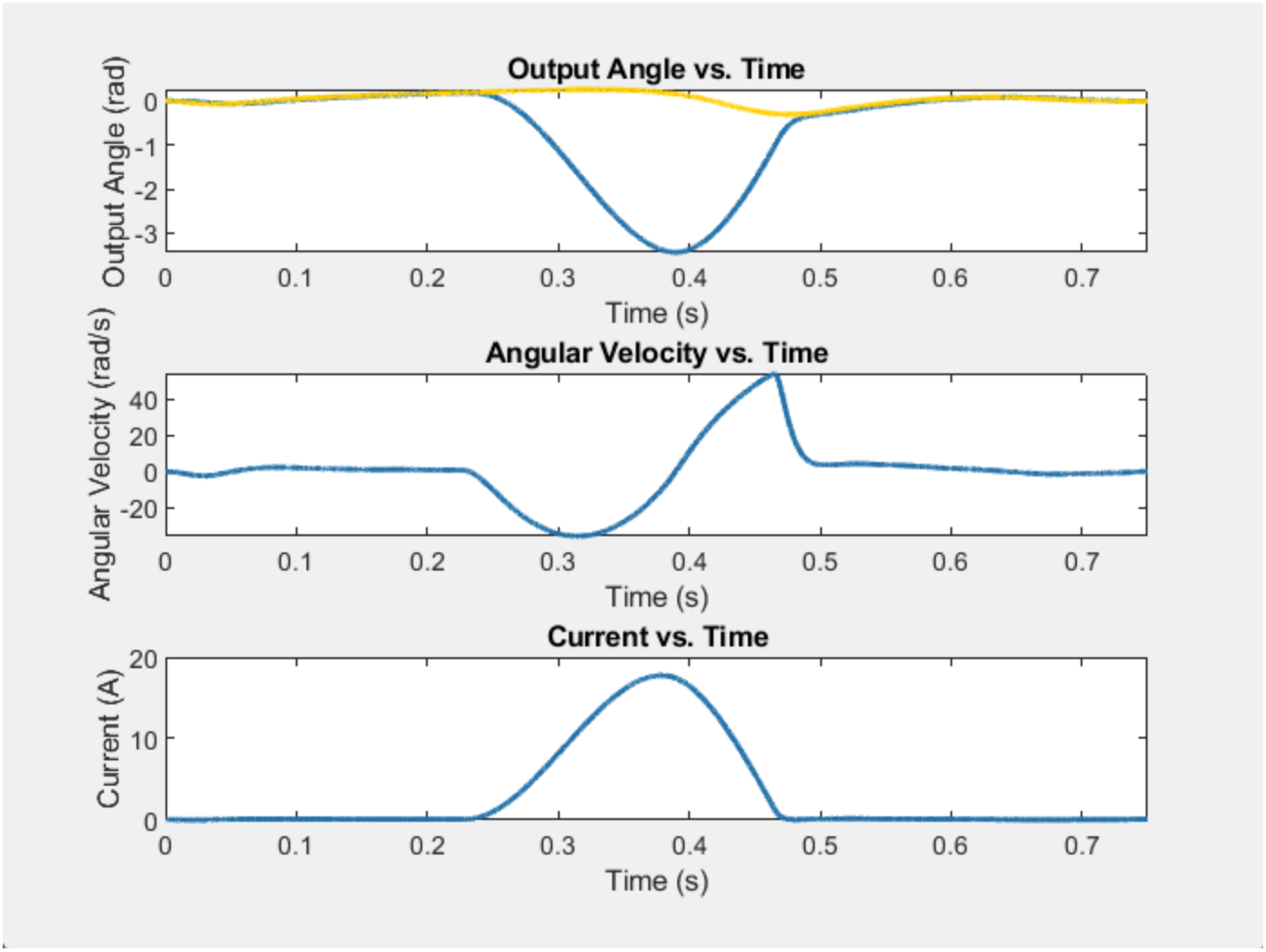
   Important note: this interpolated angle value represents our *desired* angle θ_{des}, and not our actual output angle θ_{out}. The actual output angle is one of our state variables (see step 1).
4. Set e_{in} based on the difference between the actual angle θ_{out} and the desired angle θ_{des}. This is an implementation of a very simple control strategy called proportional control. You'll learn all about this in your upcoming controls classes. For now, because this is probably new for most of you, I'll give you the code for it.

```
e_in = 5*(theta_des-theta_out)*N;
if e_in > 48
    e_in = 48;
elseif e_in < -48
    e_in = -48;
end
```

   These lines of code tell the simulated control system to apply a voltage across the motor inputs that is 5x the error between where you want the exo to be and where it actually is, all scaled by the transmission ratio. The "if" statements limit the voltage applied across the motor windings to the battery voltage, to make the system behave realistically – we can't easily ask for more voltage than our battery can provide.

Now that you have your new ODE function in place, you're ready to simulate some results. First, set the transmission ratio to 1, and all three initial conditions to zero. Use your RK4 solver with a step size of 0.0001s to solve the system of equations for $0 \leq t \leq 0.75$.

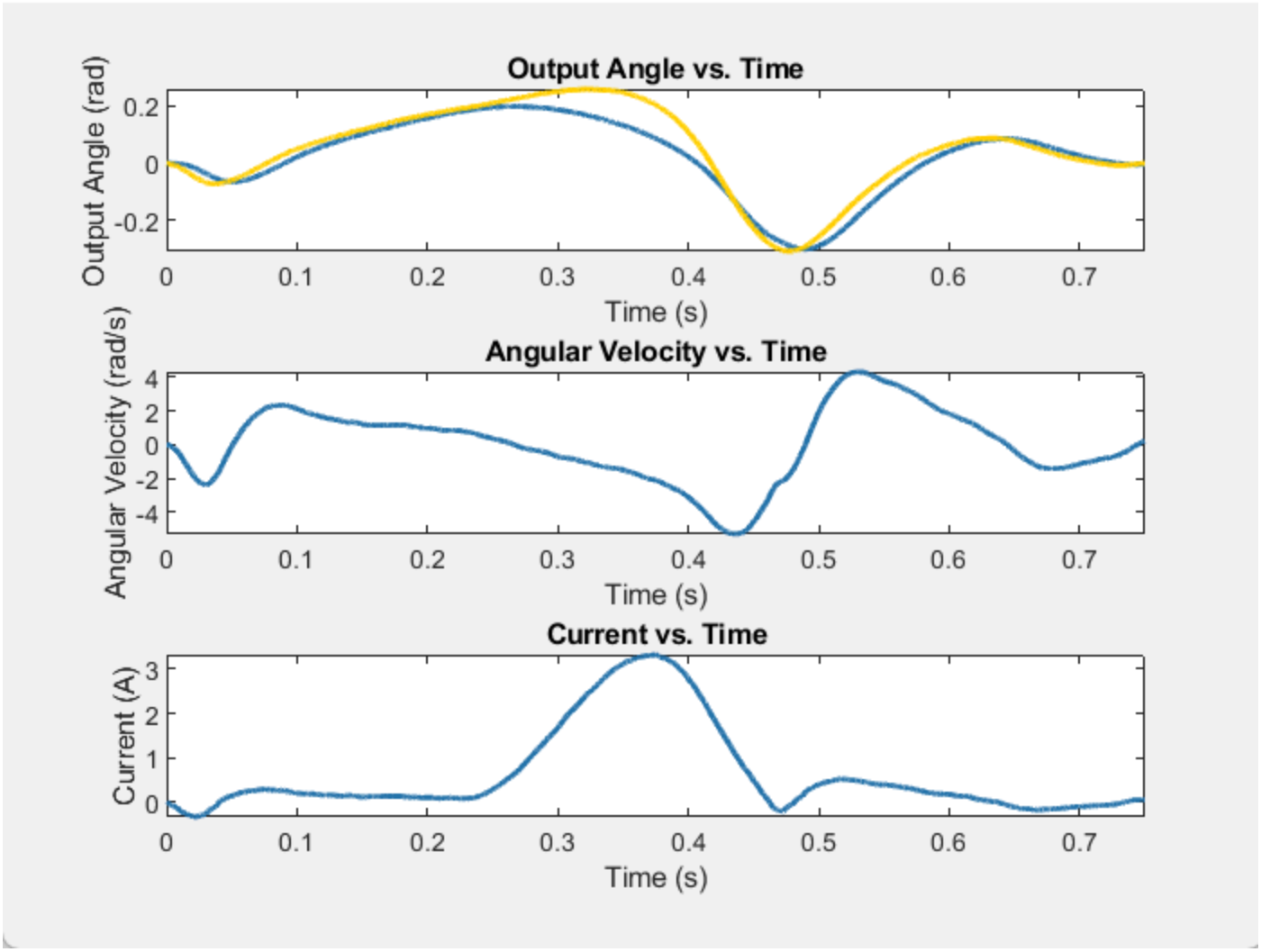
Plot each of the resultant three states ($\theta_{out}, \omega_{out}, i$) vs. time in a 3x1 set of subplots, with appropriate labels and our standard formatting. In the top subplot, also plot the desired angle trajectory on top of θ_{out}, in a different color. Drop your figure in here.



Describe what's happening here. Are we tracking our desired trajectory? Why do you think that might be the case?

We are not tracking our desired trajectory. This might be the case because the transmission ratio is not large enough to apply a large enough input voltage to reduce the error of the exo's position.

Next, let's turn the transmission ratio up to 5. **Re-run the simulation, make the same plots, and drop them here.**



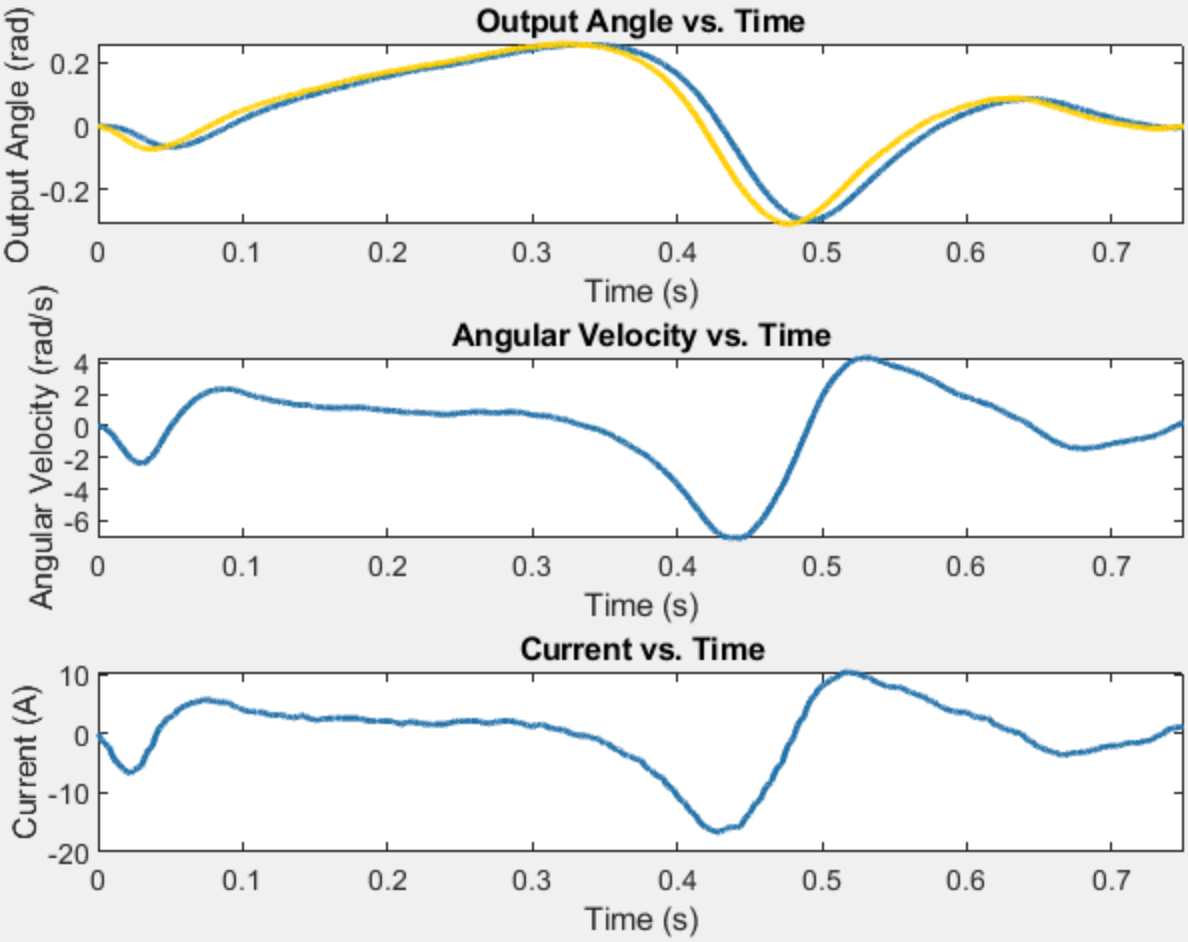
Now what's happening?

We are closer to tracking our desired trajectory since the desired angle trajectory graph is closer to the actual angle trajectory.

Why might changing the transmission ratio have this effect?

Changing the transmission ratio allows us to apply a large enough input voltage to reduce the error of the exo's position.

Lastly, let's turn the transmission ratio up to 100. **Again, re-run the simulation, make the same plots, and drop them here.**



Now what do you see? Look closely at the current.

The desired trajectory is closer to the desired angle trajectory graph. The current graph looks rough instead of smooth.

From this plot, it looks like there might be a cost associated with increasing the transmission ratio too much. Based on your initial system of equations, how might you explain this?

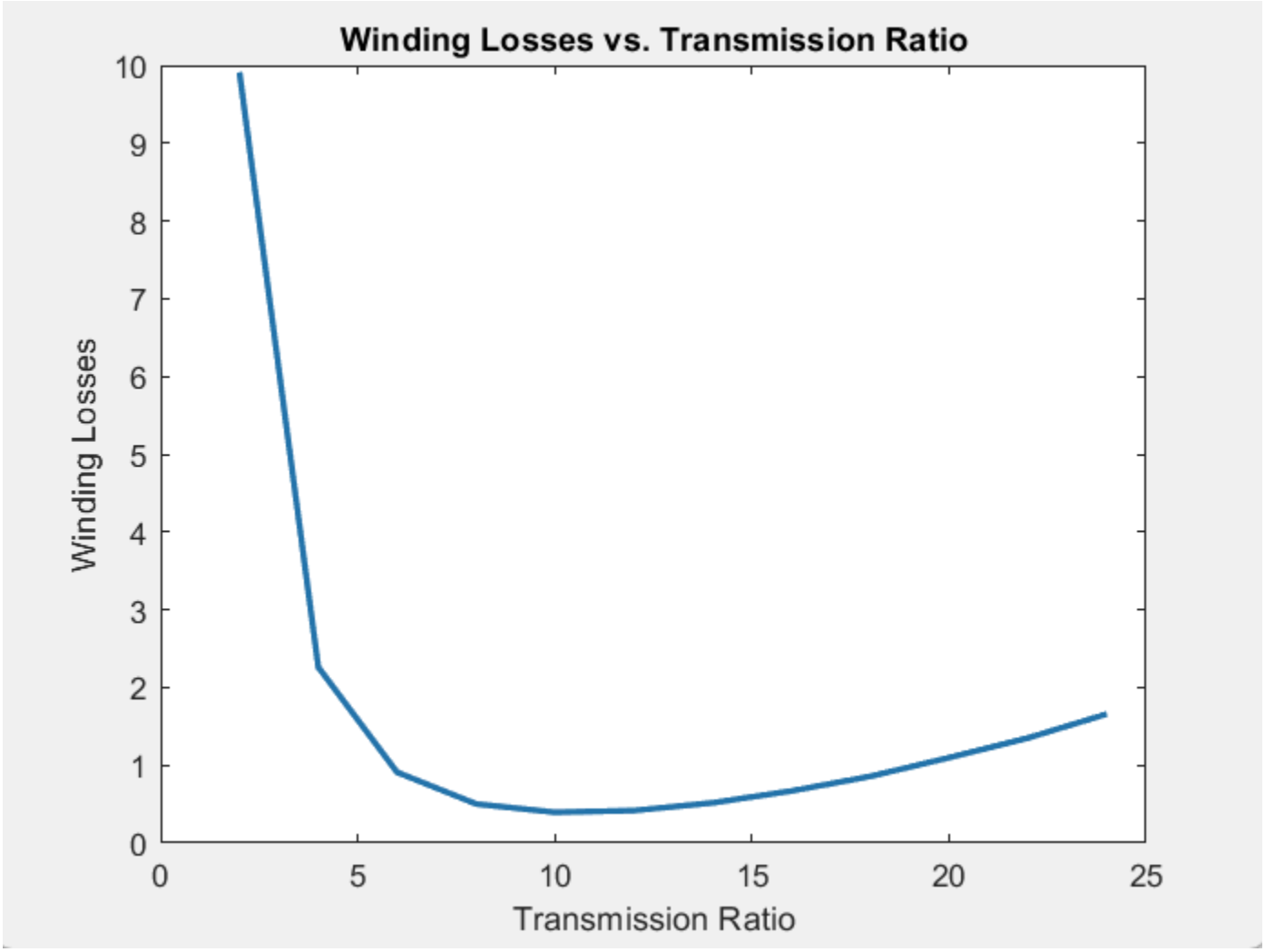
Based on the initial system of equations the torque output decreases when the transmission ratio increases. This decrease in torque output correlates to current, explaining the roughness of the current graph.

5D: Optimize transmission ratio.

We now want to use the code you've written to choose an optimal transmission ratio. Our goal will be to minimize the average "winding losses" in the system. Winding losses describe the energy we dissipate as heat due to the electrical resistance of the copper wires. Any energy lost as heat is not converted into mechanical work, and is therefore wasted. This is an extremely important concept in actuator design, and is often the deciding factor when choosing a motor. Winding losses can be calculated as $P_w = i^2 R$.

Write code to brute-force optimize the transmission ratio so that the average winding losses are minimized over the whole $0 \leq t \leq 0.75$ time window, which represents a single step. Try values of N starting at 2, and increasing by twos to 24. Use your non-homogeneous ODE function (exoBootNHODEfun.m). **Tip:** for this part, you can switch over to ODE45, rather than your RK4 solver, because it will run a bit faster.

Plot the resulting P_w vs. N, with appropriate labels and our standard formatting. Drop the plot in here.



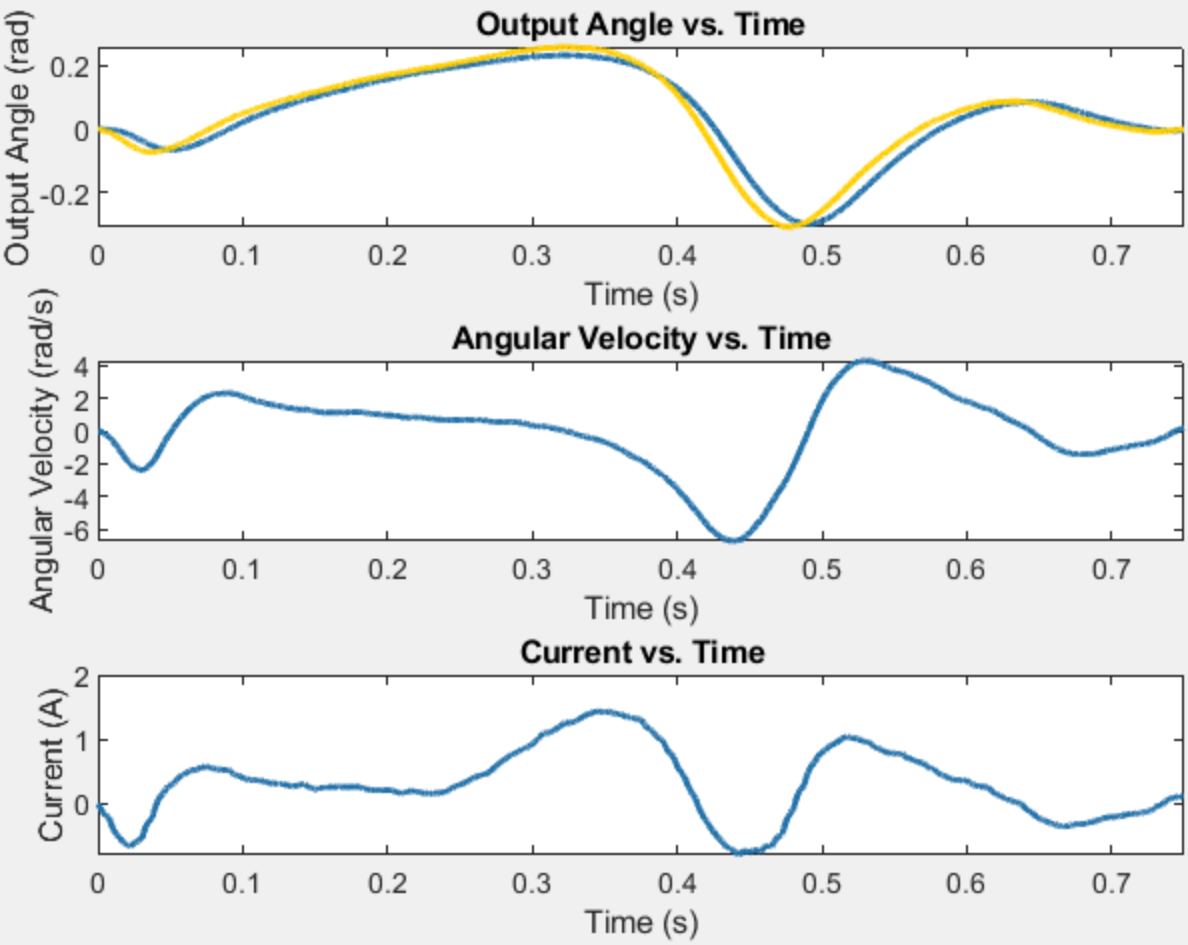
From this plot, does it look like there is an optimal transmission ratio?

Yes, there does seem to be an optimal transmission ratio (N=10).

We know that increasing the transmission ratio effectively decreases the effect of the output torque at the motor (this is the purpose of a transmission). We also know that motor torque is directly proportional to current through the motor windings. Given that winding loss is proportional to current squared, shouldn't we just be able to increase the transmission ratio infinitely and see our winding losses continue to drop? Explain why we aren't seeing this. Hint: this is similar to the answer to the last question from 5c.

We cannot increase the transmission ratio infinitely and see our winding losses continue to drop because the current also depends on the angular velocity, which changes as values of the transmission ratio changes. Current is not solely dependent on the motor torque.

Find the optimal transmission ratio from your results, and run your simulation one more time, using this optimized value. **Make the same type of 3x1 plot that you made in 5c, and drop it in here.**



We often face current limits in electronic motor drivers. From this plot, how much peak current would our driver need to be able to handle so it wouldn't fry under the expected conditions?

The driver would need to be able to handle 1.4442 amps so it wouldn't fry under the expected conditions.

Congratulations, you've designed an exoskeleton! Believe it or not, this is very close to the process that we use in the lab to choose motors and transmissions for our electromechanical systems. We always want to improve battery life, so power efficiency and winding losses are a huge concern in wearable robotics for the real world.