

Dynamical Systems Homework 2

February 14, 2025

Gabe Morris

```
[1]: # toc
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sympy as sp
from scipy.integrate import cumulative_trapezoid
from my_rk4 import rk_solve

plt.style.use('../maroon_ipynb.mplstyle')
```

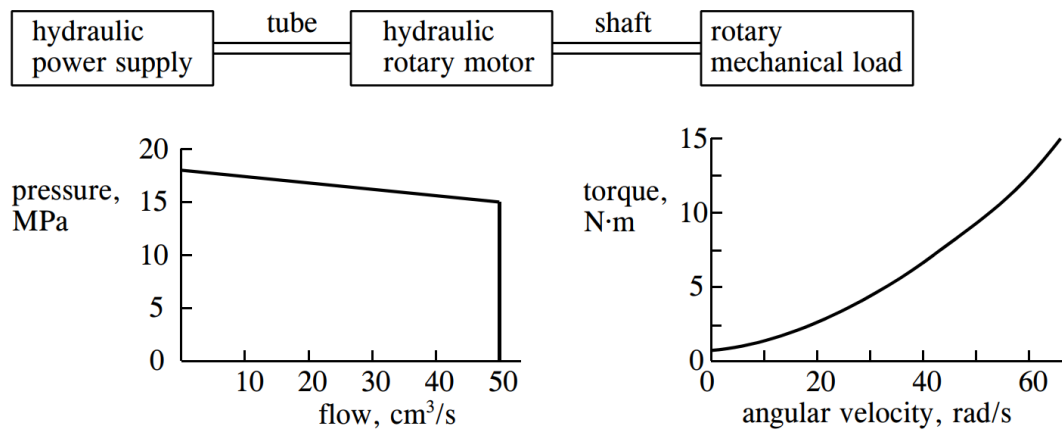
Contents

Problem 1	3
Given	3
Find	3
Solution	3
Part A	3
Part B	4
Part C	7
Verification	8
 Problem 2	 10
Given	10
Find	10
Solution	11
Part A	11
Part B	12
Part C	15
Verification	18
Part D	19

Problem 1

Given

A hydraulic power supply (comprising a motor-driven pump and a relief valve) has the pressure-flow characteristic plotted on the left below. It drives a positive displacement hydraulic motor which in turn rotates a shaft that drives some mechanical equipment. Frictional and leakage losses in the hydraulic motor may be neglected. The characteristics of the mechanical load are plotted on the right below.

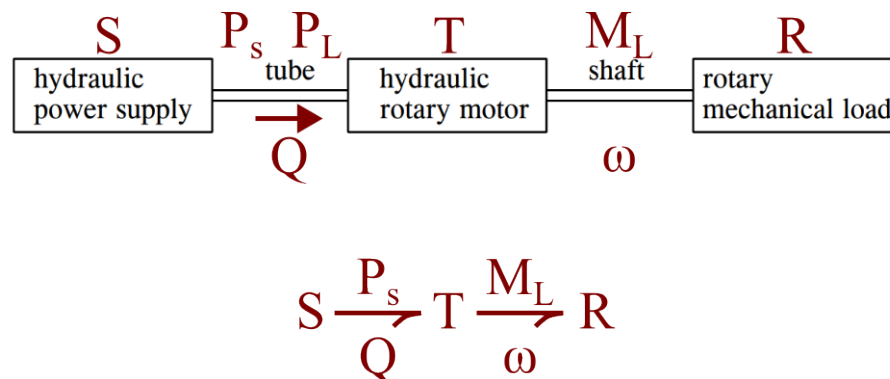


Find

- Define key variables and place them on both the drawing and a bond-graph model of the system.
- Find the maximum possible speed of the load.
- Determine the volumetric displacement per revolution of the hydraulic motor.

Solution

Part A



In the above image, the source, S , causes a pressure difference, P_s , in the hydraulic rotary motor (the transformer T). This pressure difference causes a flow rate, Q , in the motor. The motor then

induces a moment, M_L , on a shaft connected to the mechanical load, R . This in turn causes the load to rotate at a speed, ω . The load also produces a resistant pressure, P_L , on the source. This ideal machine yields the following relationships:

$$P_s = TM_L$$

$$\omega = TQ$$

Part B

We need to find the modulus that results in a maximum speed. This can be done by operating at the peak power of the source curve. First off, let's convert these graphs into functions. For the source curve, on inspection, you'll find that the relationship is linear:

$$P_s(Q) = 18 - 0.06Q \text{ MPa}$$

We need to ensure consistent units for the transformer modulus. This requires the pressure and flow rate to change to Pa and m^3/s , respectively. Making this change, we get:

$$P_s(Q) = 18 \cdot 10^6 - 6 \cdot 10^{10}Q \text{ Pa}$$

This ensures that the units for the modulus come out to be $\frac{1}{m^3}$. For the load curve, this is a little more complicated. The curve can be approximated by a quadratic function that has been fitted to the data.

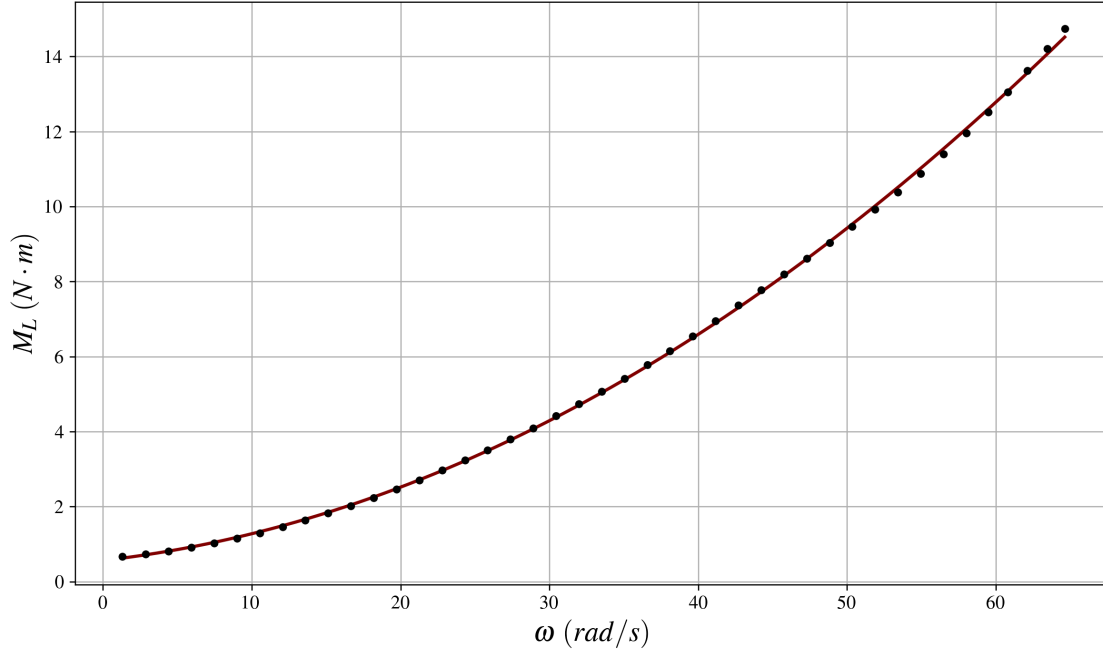
```
[2]: # Finding the best fit curve for the load data
data = pd.read_csv('p1_data.csv')
omegas = np.array(data['omega'])
torques = np.array(data['torque'])

A, B, C = np.polyfit(omegas, torques, 2)
print(f'A: {A}, B: {B}, C: {C}')

omega_array = np.linspace(omegas[0], omegas[-1], 300)
torque_array = A*omega_array**2 + B*omega_array + C

fig, ax = plt.subplots()
ax.scatter(omegas, torques, zorder=3, color='black', marker='.')
ax.plot(omega_array, torque_array)
ax.set_xlabel(r'$\omega$ (rad/s)')
ax.set_ylabel(r'$M_L$ (N·m)')
plt.show()
```

A: 0.002646604774363997, B: 0.04479742996739807, C: 0.5613276032319153



This yields the following equation:

$$M_L(\omega) = 0.00265\omega^2 + 0.0448\omega + 0.561 \text{ N} \cdot \text{m}$$

Now we can find the peak power of the source curve where power is

$$\dot{E} = P_s \cdot Q$$

```
[3]: # Finding the peak power
Q = sp.Symbol('Q')
Ps = int(18e6) - int(6e10)*Q
E_dot = Ps*Q
E_dot
```

```
[3]: Q (18000000 - 60000000000Q)
```

```
[4]: E_dot_diff = E_dot.diff()
Q_max = sp.solve(E_dot_diff, Q)[0]
Q_max.n() # m^3/s
```

```
[4]: 0.00015
```

We have calculated that the ideal flow rate would be $0.00015 \text{ m}^3/\text{s}$, but this is not within the bounds of source operating conditions. Therefore, we will have to operate at the maximum possible flow rate, which is $0.00005 \text{ m}^3/\text{s}$, since this is the peak power of the source.

```
[5]: Q_max_power = 0.00005 # m^3/s
# Q_max_power = Q_max # m^3/s

# Find the value of T at this flow rate
omega, T = sp.symbols('omega T')
M_L = A*omega**2 + B*omega + C
omega_ = T*Q
M_L_ = M_L.subs(omega, omega_)
eq = sp.Eq(Ps, T*M_L_)
eq
```

```
[5]: 18000000-60000000000Q = T (0.002646604774364Q^2T^2 + 0.0447974299673981QT + 0.561327603231915)
```

```
[6]: eq = eq.subs(Q, Q_max_power)
eq.simplify()
```

```
[6]: 6.61651193590999 · 10-12T3 + 2.2398714983699 · 10-6T2 + 0.561327603231915T = 15000000.0
```

```
[7]: sol = sp.solve(eq, T)
sol
```

```
[7]: [1190301.78775134,
      -764414.663069025 - 1149032.59001054*I,
      -764414.663069025 + 1149032.59001054*I]
```

```
[8]: # Choose only the real solution
T_best = [T_val for T_val in sol if sp.im(T_val) == 0][0]
T_best # 1/m^3
```

```
[8]: 1190301.78775134
```

The transformer modulus of $T = 1,190,302 \frac{1}{m^3}$ produces the maximum power output that the source can provide. The corresponding load speed is

```
[9]: omega_max = T_best*Q_max_power
omega_max # rad/s
```

```
[9]: 59.5150893875669
```

If we plot the source curve with the load curve, we should see an equilibrium point at $Q_{max} = 0.00005$. However, this load needs to be the load that acts on the source, P_L .

$$P_L = T \cdot M_L(\omega) = T \cdot M_L(T \cdot Q)$$

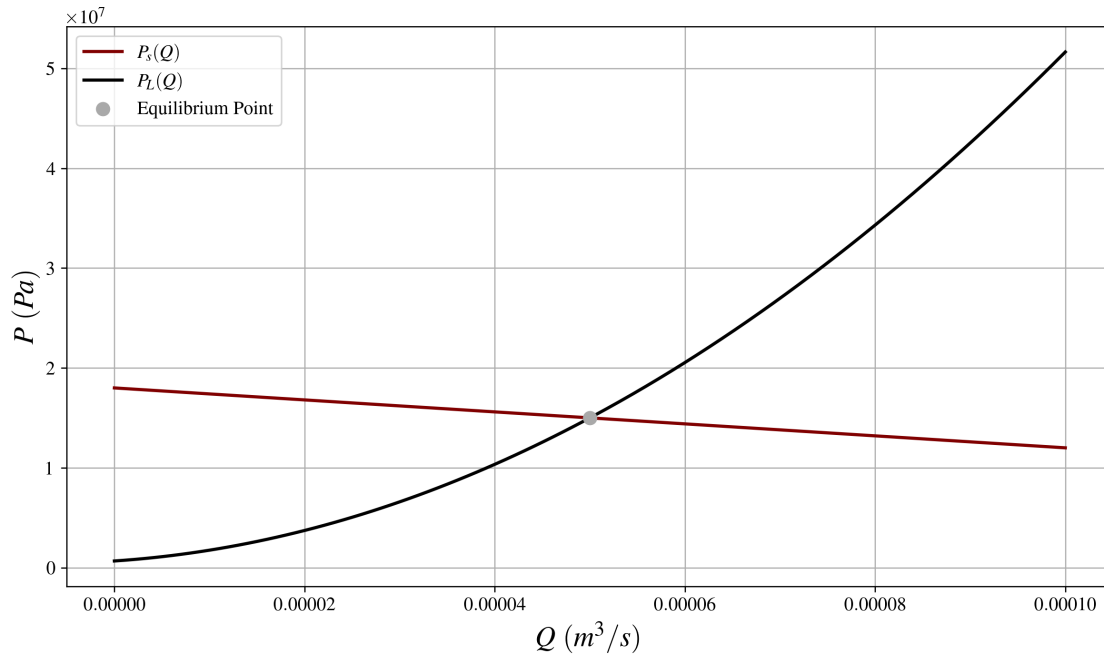
```
[10]: P_L = T_best*M_L.subs(omega, T_best*Q)
P_L.simplify()
```

```
[10]: 4.46334388758242 · 1015Q2 + 63469820628.0566Q + 668149.249641123
```

```
[11]: # Plotting
P_L_lamb = sp.lambdify(Q, P_L, modules='numpy')
Ps_lamb = sp.lambdify(Q, Ps, modules='numpy')

Q_array = np.linspace(0, float(Q_max_power)*2, 300)

fig, ax = plt.subplots()
ax.plot(Q_array, Ps_lamb(Q_array), label=r'$P_s(Q)$')
ax.plot(Q_array, P_L_lamb(Q_array), label=r'$P_L(Q)$')
ax.scatter(Q_max_power, Ps_lamb(Q_max_power), color='darkgrey', zorder=3,
↪label='Equilibrium Point')
ax.set_xlabel('$Q$ $(m^3/s)$')
ax.set_ylabel('$P$ $(Pa)$')
ax.legend()
plt.show()
```



Part C

The volumetric displacement per revolution would be

$$D = \frac{2\pi}{T}$$

which is just the reciprocal of the modulus. The 2π is the conversion factor from radians to revolutions.

```
[12]: D = 2*sp.pi/T_best
      D.n(5) # m^3/rev
```

```
[12]: 5.2786 · 10-6
```

Verification

We can verify that this modulus results in the maximum speed by considering the speed, ω , as a function of T .

```
[13]: eq1 = sp.Eq(Ps, T*M_L)
      eq2 = sp.Eq(omega, T*Q)
      display(eq1, eq2)
```

$$18000000 - 60000000000Q = T(0.002646604774364\omega^2 + 0.0447974299673981\omega + 0.561327603231915)$$

$$\omega = QT$$

```
[14]: Q_of_T_omega = sp.solve(eq1, Q)[0]
      eq3 = eq2.subs(Q, Q_of_T_omega)
      eq3
```

```
[14]:  $\omega = T(-4.41100795727333 \cdot 10^{-14}T\omega^2 - 7.46623832789968 \cdot 10^{-13}T\omega - 9.35546005386525 \cdot 10^{-12}T + 0.0003)$ 
```

```
[15]: omega_of_T = sp.solve(eq3, omega, dict=True)
      for sol in omega_of_T:
          for key, value in sol.items():
              display(sp.Eq(key, value).n(5))
```

$$\omega = \frac{2.2671 \cdot 10^{-14} \left(-3.7331 \cdot 10^{14}T^2 - 5.0 \cdot 10^{26} \left(-1.0932 \cdot 10^{-24}T^4 + 5.2932 \cdot 10^{-17}T^3 + 1.4932 \cdot 10^{-12}T^2 + 1.0 \right)^0 \right)}{T^2}$$

$$\omega = \frac{2.2671 \cdot 10^{-14} \left(-3.7331 \cdot 10^{14}T^2 + 5.0 \cdot 10^{26} \left(-1.0932 \cdot 10^{-24}T^4 + 5.2932 \cdot 10^{-17}T^3 + 1.4932 \cdot 10^{-12}T^2 + 1.0 \right)^0 \right)}{T^2}$$

Only one of these solutions should result in positive values of ω for positive values of T , and it's the second solution.

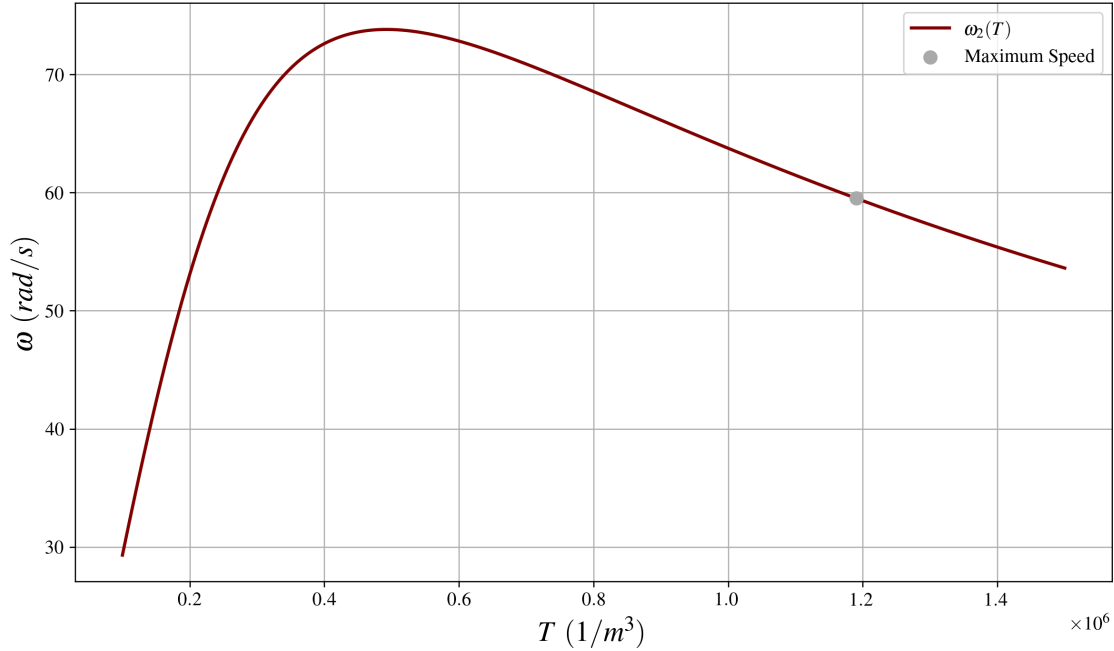
```
[16]: omega1, omega2 = omega_of_T[0][omega], omega_of_T[1][omega]
      omega1_lamb, omega2_lamb = sp.lambdify(T, omega1, modules='numpy'), sp.
      ↪lambdify(T, omega2, modules='numpy')

      T_array = np.linspace(100_000, 1.5e6, 5000)
      omega1_array, omega2_array = omega1_lamb(T_array), omega2_lamb(T_array)

      fig, ax = plt.subplots()
      # ax.plot(T_array, omega1_array, label=r'$\omega_1(T)$')
      ax.plot(T_array, omega2_array, label=r'$\omega_2(T)$')
      ax.set_xlabel(r'$T$ $(1/m^3)$')
      ax.set_ylabel(r'$\omega$ $(rad/s)$')
```



```
ax.scatter(float(T_best), float(omega_max), color='darkgrey', zorder=3,
          ↪label='Maximum Speed')
ax.legend()
plt.show()
```

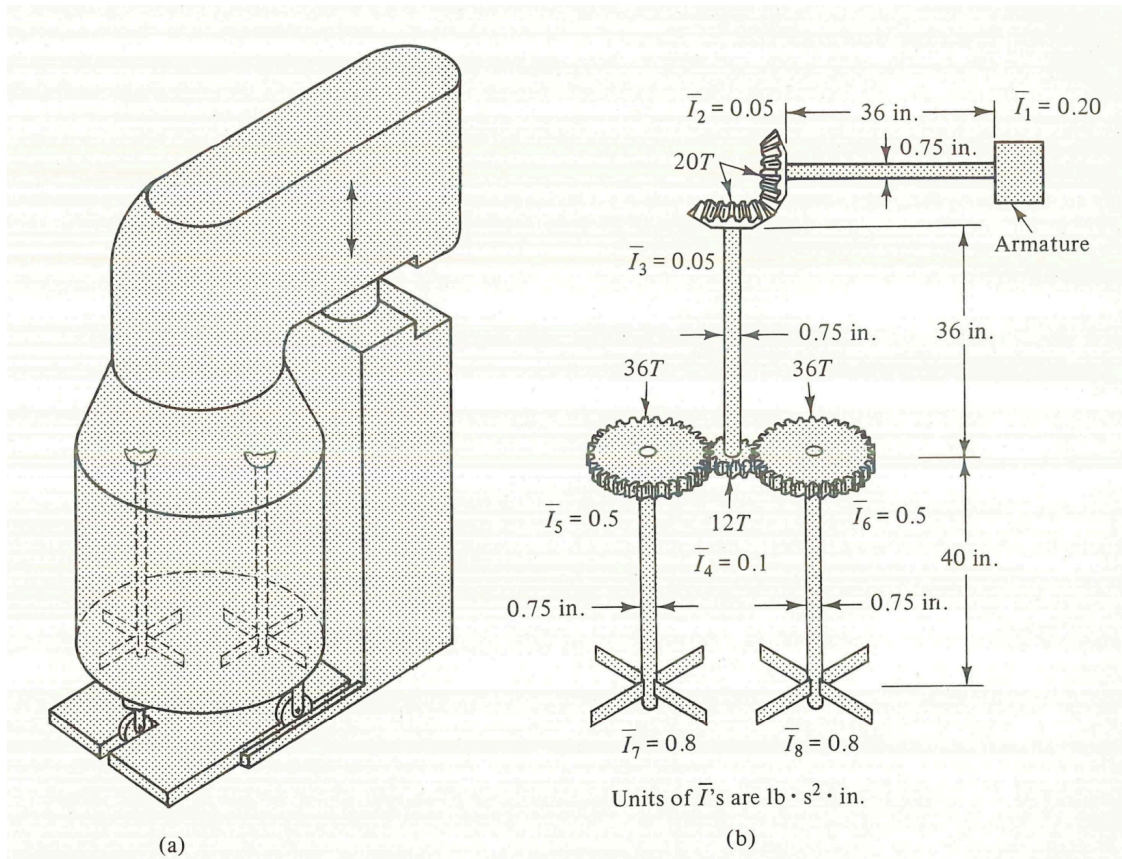


If there was not a limitation on the source, then the best transformer modulus would be $T = 492,104 \frac{1}{m^3}$, resulting in a speed of 73.8 rad/s . However, this occurs at a flow rate of $Q_{max} = 0.00015$, which is not within the operating conditions of the source. If you wanted to see the results without this limitation, then you can uncomment the line `Q_max_power = Q_max # m³/s` in the code cell 5.

You should also recognize that in order to get that dot to move to the left, you need to keep increasing the flow rate Q . Looking at the graph, you can see that the output shaft speed decreases with increasing modulus, so we can definitely say that this is the maximum speed.

Problem 2

Given



For the figure shown above assume the armature has a moment-speed curve of $M_a(\omega_1) = -315\omega_1 + 2000$ when the motor is on and acts as a damper when the motor is off with $M_a(\omega_1) = -14.4\omega_1$. In both cases the armature speed is in rad/s and the moment is in $\text{lb} \cdot \text{in.}$

Assume that all shafts are made of steel ($G = 1.15 \cdot 10^7 \text{ psi}$) and that the diameter of the bevel gears is 2 inches. Model each shaft as a rotational spring ($k = \frac{\pi r^4 G}{2L}$). Model the resistance felt by each of the paddles (inertias 7 and 8 shown at the bottom of the industrial mixer) with rotational dampers of a coefficient of $B = 2880 \text{ lb} \cdot \text{in} \cdot \text{s}$. The number of teeth of each gear (20T, 12T, and 36T) is given in the figure.

Find

Obtain the following:

- A detailed free body diagram including your choice of reference coordinates.
- A complete set of differential equations in state variable form. Notice that there are 8 inertias and 4 springs, but not all inertias are independent.
- Solve the state variable equations using a 4th order Runge-Kutta method for a time period of 20 seconds. Assume that all initial states are zero and that the motor armature is on for

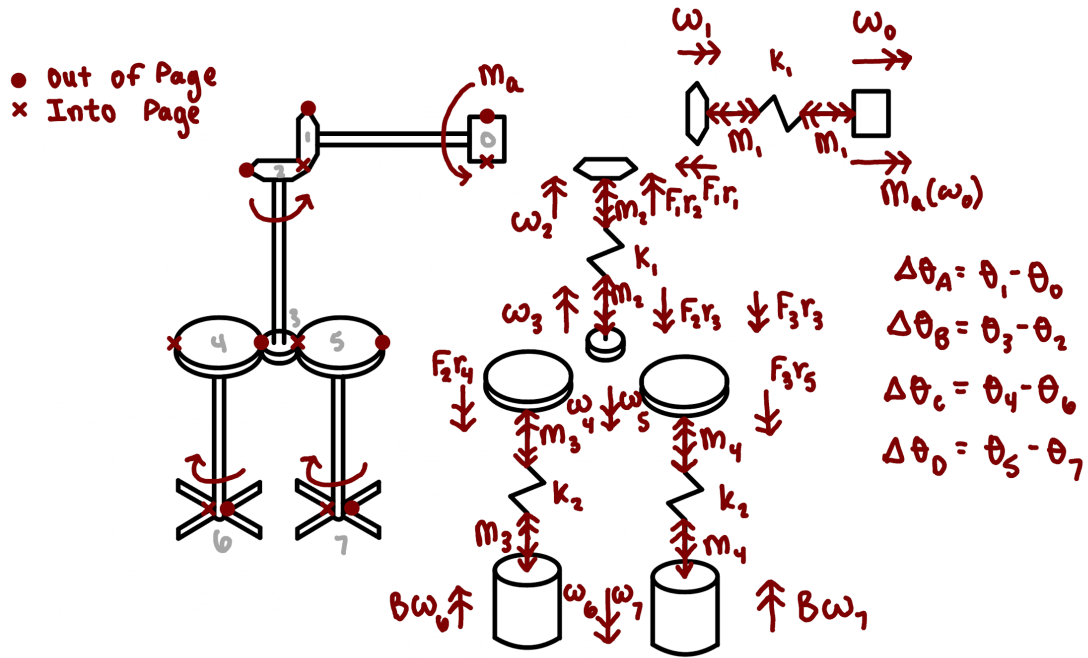
the first 10 seconds and off for the last 10 seconds. It is recommended that a very small time step is used. Try $\Delta t = 0.0005s$.

d. Verify using energy conservation and include a discussion on the results with the following points:

- When the motor is turned on and at steady-state (ω_1 is constant), does it operate at its maximum power?
- Also, consider the mechanical failure modes that might occur with the plotted speed results.

Solution

Part A



With this free body diagram, we can see that inertias 1 and 2 can be combined as well as inertias 3, 4, and 5. Additionally, we have the following transformer relationships:

$$\begin{aligned}\omega_2 &= T_1 \omega_1 \\ M_1 &= T_1 M_2 \\ \omega_4 &= T_2 \omega_3 \\ M_2 &= T_2 M_3 \\ \omega_5 &= T_3 \omega_3 \\ M_2 &= T_3 M_4\end{aligned}$$

with $T_1 = \frac{r_1}{r_2} = \frac{N_1}{N_2}$, $T_2 = \frac{r_3}{r_4} = \frac{N_3}{N_4}$, and $T_3 = \frac{r_3}{r_5} = \frac{N_3}{N_5}$.

```
[17]: # Define symbols
t = sp.Symbol('t')
k1, k2 = sp.symbols('k1 k2')
Ma = sp.Symbol('M_a')
T1, T2, T3 = sp.symbols('T1 T2 T3')
I0, I1, I2, I3, I4, I5, I6, I7 = sp.symbols('I0 I1 I2 I3 I4 I5 I6 I7')
B = sp.Symbol('B')

# Define functions
del_A, del_B, del_C, del_D = [sp.Function(rf'\Delta\theta_{i}')(t) for i in range(4)]
omega0, omega1, omega2, omega3, omega4, omega5, omega6, omega7 = [sp.Function(rf'\omega_{i}')(t) for i in range(8)]
```

Part B

Let's make the first lump of mass as if it were seen by I_2 . The second lump of mass is going to be seen by I_3 .

```
[18]: half = sp.Rational(1, 2)
KE1 = half*I1*omega1**2 + half*I2*omega2**2
KE1
```

$$[18]: \frac{I_1 \omega_1^2(t)}{2} + \frac{I_2 \omega_2^2(t)}{2}$$

```
[19]: KE1 = KE1.subs(omega1, omega2/T1)
KE1 = sp.collect(KE1, half*omega2**2)
Ie1 = KE1.coeff(half*omega2**2)
KE1
```

$$[19]: \frac{\left(\frac{I_1}{T_1^2} + I_2\right) \omega_2^2(t)}{2}$$

```
[20]: Ie1
```

$$[20]: \frac{I_1}{T_1^2} + I_2$$

```
[21]: # Second lump
KE2 = half*I3*omega3**2 + half*I4*omega4**2 + half*I5*omega5**2
KE2
```

$$[21]: \frac{I_3 \omega_3^2(t)}{2} + \frac{I_4 \omega_4^2(t)}{2} + \frac{I_5 \omega_5^2(t)}{2}$$

```
[22]: KE2 = KE2.subs([
    (omega4, omega3*T2),
    (omega5, omega3*T3)
])
```

```
KE2 = sp.collect(KE2, half*omega3**2)
Ie2 = KE2.coef(half*omega3**2)
KE2
```

[22]:
$$\frac{(I_3 + I_4 T_2^2 + I_5 T_3^2) \omega_3^2(t)}{2}$$

[23]: Ie2

[23]:
$$I_3 + I_4 T_2^2 + I_5 T_3^2$$

The equations of motions are as follows:

```
[24]: # Transformer substitutions
trans_sub = [
    (omega1, omega2/T1),
    (omega4, omega3*T2),
    (omega5, omega3*T3)
]

eq1 = sp.Eq(del_A.diff(), omega1 - omega0).subs(trans_sub)
eq2 = sp.Eq(del_B.diff(), omega3 - omega2).subs(trans_sub)
eq3 = sp.Eq(del_C.diff(), omega4 - omega6).subs(trans_sub)
eq4 = sp.Eq(del_D.diff(), omega5 - omega7).subs(trans_sub)
eq5 = sp.Eq(I0*omega0.diff(), Ma + k1*del_A)
eq6 = sp.Eq(Ie1*omega2.diff(), k1*del_B - k1/T1*del_A)
eq7 = sp.Eq(Ie2*omega3.diff(), -k1*del_B - T2*k2*del_C - T3*k2*del_D)
eq8 = sp.Eq(I6*omega6.diff(), k2*del_C - B*omega6)
eq9 = sp.Eq(I7*omega7.diff(), k2*del_D - B*omega7)
eqs = [eq1, eq2, eq3, eq4, eq5, eq6, eq7, eq8, eq9]
display(*eqs)
```

$$\frac{d}{dt} \Delta \theta_A(t) = -\omega_0(t) + \frac{\omega_2(t)}{T_1}$$

$$\frac{d}{dt} \Delta \theta_B(t) = -\omega_2(t) + \omega_3(t)$$

$$\frac{d}{dt} \Delta \theta_C(t) = T_2 \omega_3(t) - \omega_6(t)$$

$$\frac{d}{dt} \Delta \theta_D(t) = T_3 \omega_3(t) - \omega_7(t)$$

$$I_0 \frac{d}{dt} \omega_0(t) = M_a + k_1 \Delta \theta_A(t)$$

$$\left(\frac{I_1}{T_1^2} + I_2 \right) \frac{d}{dt} \omega_2(t) = k_1 \Delta \theta_B(t) - \frac{k_1 \Delta \theta_A(t)}{T_1}$$

$$(I_3 + I_4 T_2^2 + I_5 T_3^2) \frac{d}{dt} \omega_3(t) = -T_2 k_2 \Delta \theta_C(t) - T_3 k_2 \Delta \theta_D(t) - k_1 \Delta \theta_B(t)$$

$$I_6 \frac{d}{dt} \omega_6(t) = -B \omega_6(t) + k_2 \Delta \theta_C(t)$$

$$I_7 \frac{d}{dt} \omega_7(t) = -B\omega_7(t) + k_2 \Delta\theta_D(t)$$

I have defined M_a as a symbol instead of a function of ω_0 , but this is to allow `sympy` to convert this expression to the matrix form later. If I defined it as a function, then `sympy` would recognize a non-linearity in the state variable solution.

```
[25]: # Solving for the first order derivatives
state_sol = sp.solve(
    eqs,
    (del_A.diff(), del_B.diff(), del_C.diff(), del_D.diff(), omega0.diff(),
    ↪ omega2.diff(), omega3.diff(), omega6.diff(), omega7.diff()),
    dict=True
)[0]

for key, value in state_sol.items():
    display(sp.Eq(key, value.simplify()))
```

$$\frac{d}{dt} \Delta\theta_A(t) = -\omega_0(t) + \frac{\omega_2(t)}{T_1}$$

$$\frac{d}{dt} \Delta\theta_B(t) = -\omega_2(t) + \omega_3(t)$$

$$\frac{d}{dt} \Delta\theta_C(t) = T_2\omega_3(t) - \omega_6(t)$$

$$\frac{d}{dt} \Delta\theta_D(t) = T_3\omega_3(t) - \omega_7(t)$$

$$\frac{d}{dt} \omega_0(t) = \frac{M_a + k_1 \Delta\theta_A(t)}{I_0}$$

$$\frac{d}{dt} \omega_2(t) = \frac{T_1 k_1 (T_1 \Delta\theta_B(t) - \Delta\theta_A(t))}{I_1 + I_2 T_1^2}$$

$$\frac{d}{dt} \omega_3(t) = \frac{-T_2 k_2 \Delta\theta_C(t) - T_3 k_2 \Delta\theta_D(t) - k_1 \Delta\theta_B(t)}{I_3 + I_4 T_2^2 + I_5 T_3^2}$$

$$\frac{d}{dt} \omega_6(t) = \frac{-B\omega_6(t) + k_2 \Delta\theta_C(t)}{I_6}$$

$$\frac{d}{dt} \omega_7(t) = \frac{-B\omega_7(t) + k_2 \Delta\theta_D(t)}{I_7}$$

```
[26]: # Convert to matrix
x = [del_A, del_B, del_C, del_D, omega0, omega2, omega3, omega6, omega7]
A, b = sp.linear_eq_to_matrix(list(state_sol.values()), x)
mat_eq = sp.Eq(sp.Matrix(list(state_sol.keys())), sp.Add(sp.MatMul(A, sp.
    ↪ Matrix(x)), -b))
mat_eq
```

[26]:

$$\begin{bmatrix} \frac{d}{dt} \Delta\theta_A(t) \\ \frac{d}{dt} \Delta\theta_B(t) \\ \frac{d}{dt} \Delta\theta_C(t) \\ \frac{d}{dt} \Delta\theta_D(t) \\ \frac{d}{dt} \omega_0(t) \\ \frac{d}{dt} \omega_2(t) \\ \frac{d}{dt} \omega_3(t) \\ \frac{d}{dt} \omega_6(t) \\ \frac{d}{dt} \omega_7(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{M_a}{I_0} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & -1 & \frac{1}{T_1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & T_2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & T_3 & 0 & -1 \\ \frac{k_1}{I_0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{T_1 k_1}{I_1 + I_2 T_1^2} & \frac{T_1^2 k_1}{I_1 + I_2 T_1^2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{k_1}{I_3 + I_4 T_2^2 + I_5 T_3^2} & -\frac{T_2 k_2}{I_3 + I_4 T_2^2 + I_5 T_3^2} & -\frac{T_3 k_2}{I_3 + I_4 T_2^2 + I_5 T_3^2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{k_2}{I_6} & 0 & 0 & 0 & 0 & -\frac{B}{I_6} & 0 \\ 0 & 0 & 0 & \frac{k_2}{I_7} & 0 & 0 & 0 & 0 & -\frac{B}{I_7} \end{bmatrix} \begin{bmatrix} \Delta\theta_A(t) \\ \Delta\theta_B(t) \\ \Delta\theta_C(t) \\ \Delta\theta_D(t) \\ \omega_0(t) \\ \omega_2(t) \\ \omega_3(t) \\ \omega_6(t) \\ \omega_7(t) \end{bmatrix}$$

Part C

We just need to write a function that will return the right hand side of the equations above. You can convert the `sympy` results above to the numerical world with the `lambdify` function. This ensures no errors when copying the results.

```
[27]: # Define the numerical constants
# Inertias in lbf*in*s^2
I0_ = 0.2
I1_, I2_ = 0.05, 0.05
I3_ = 0.1
I4_, I5_ = 0.5, 0.5
I6_, I7_ = 0.8, 0.8

# lengths and diameters in inches
L1, D1 = 36, 0.75
L2, D2 = 40, 0.75

G = 1.15e7 # psi
k1_ = (np.pi*(D1/2)**4*G)/(2*L1)
k2_ = (np.pi*(D2/2)**4*G)/(2*L2)

B_ = 2880 # lbf*in*s

# Transformer moduli
T1_ = 20/20
T2_ = 12/36
T3_ = 12/36

# Defining the input function
def Ma(w0_, t_):
    return -315*w0_ + 2000 if t_ < 10 else -14.4*w0_

A_lamb = sp.lambdify(
    (I0, I1, I2, I3, I4, I5, I6, I7, T1, T2, T3, k1, k2, B),
    A,
    modules='numpy'
```

```

)
A_ = A_lamb(I0_, I1_, I2_, I3_, I4_, I5_, I6_, I7_, T1_, T2_, T3_, k1_, k2_, B_)

def state_vars(states, t_):
    del_A_, del_B_, del_C_, del_D_, omega0_, omega2_, omega3_, omega6_, omega7_ =
    ↪ states
    Ma_ = Ma(omega0_, t_)
    b_ = np.array([0, 0, 0, 0, Ma_/I0_, 0, 0, 0, 0])
    return A_@states + b_

t_array = np.linspace(0, 20, 40_001) # results in h = 0.0005
sol = rk_solve(state_vars, [0]*9, t_array)

```

I will now plot just the speeds of the inertias since this has more meaning than the displacements.

```

[28]: for i in range(4, 9):
        omega = sol[:, i]
        omega_label = sp.latex(x[i])

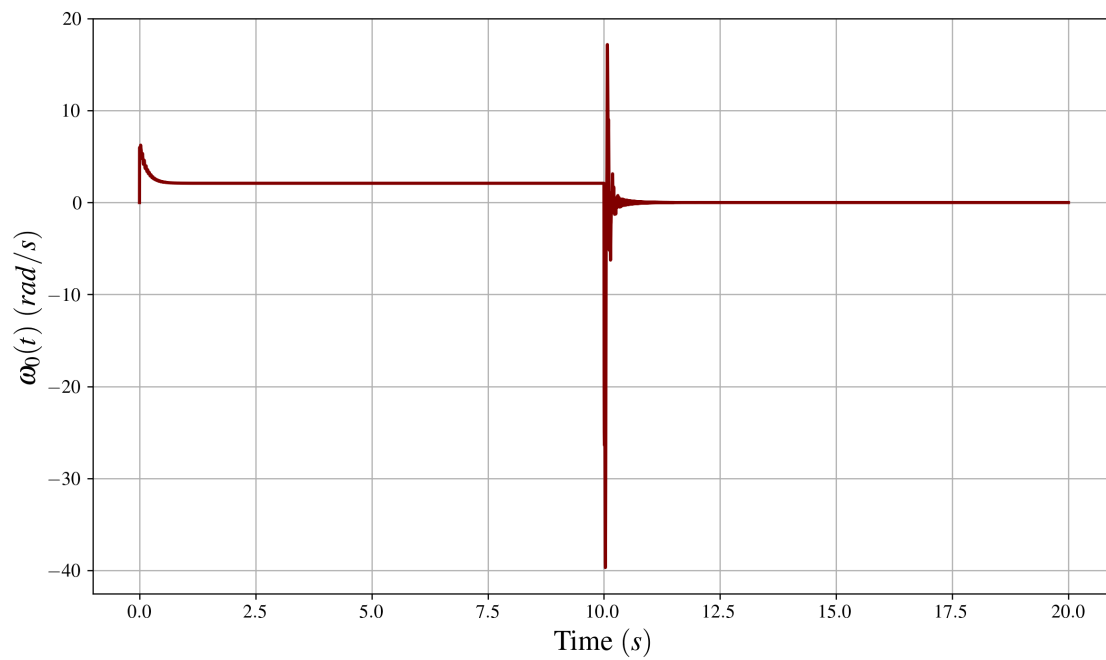
        fig, ax = plt.subplots()

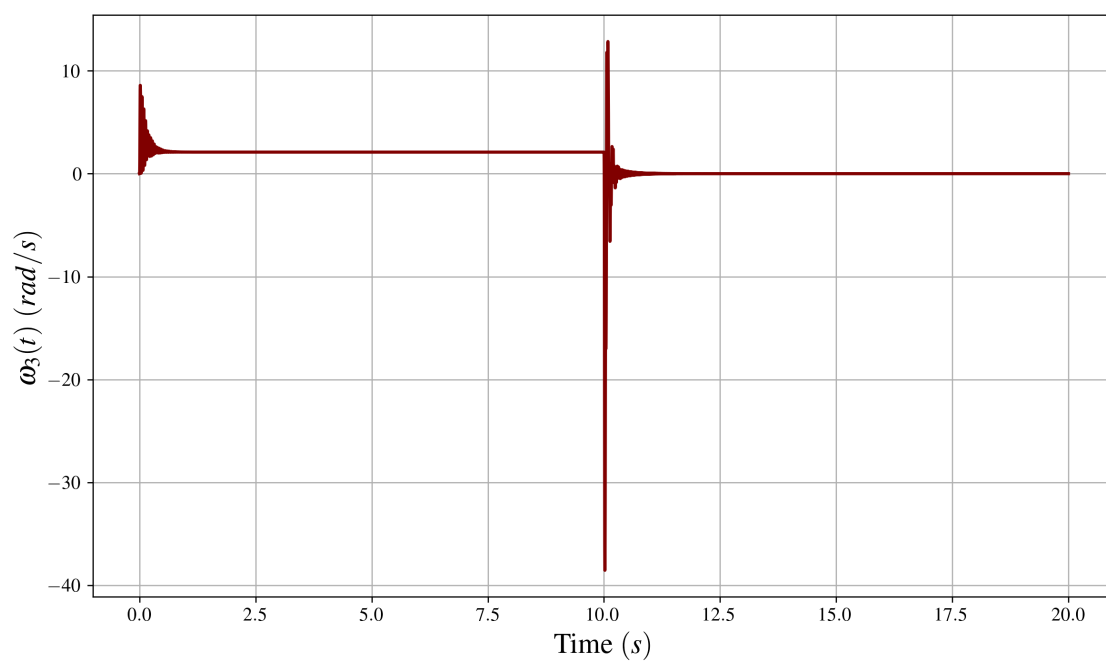
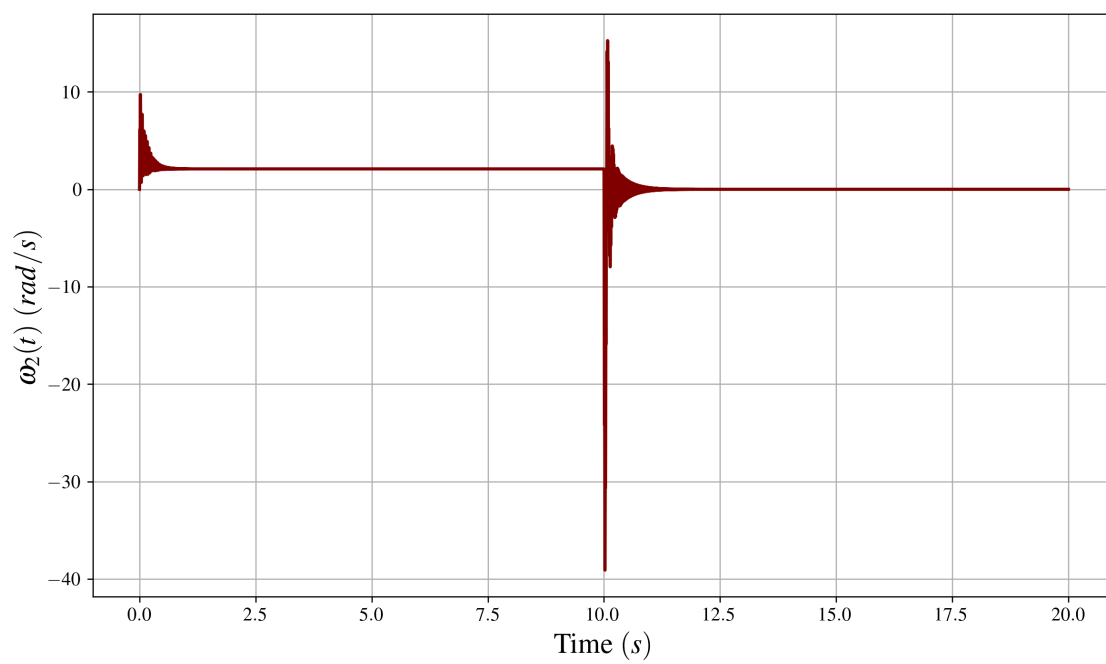
        ax.plot(t_array, omega)

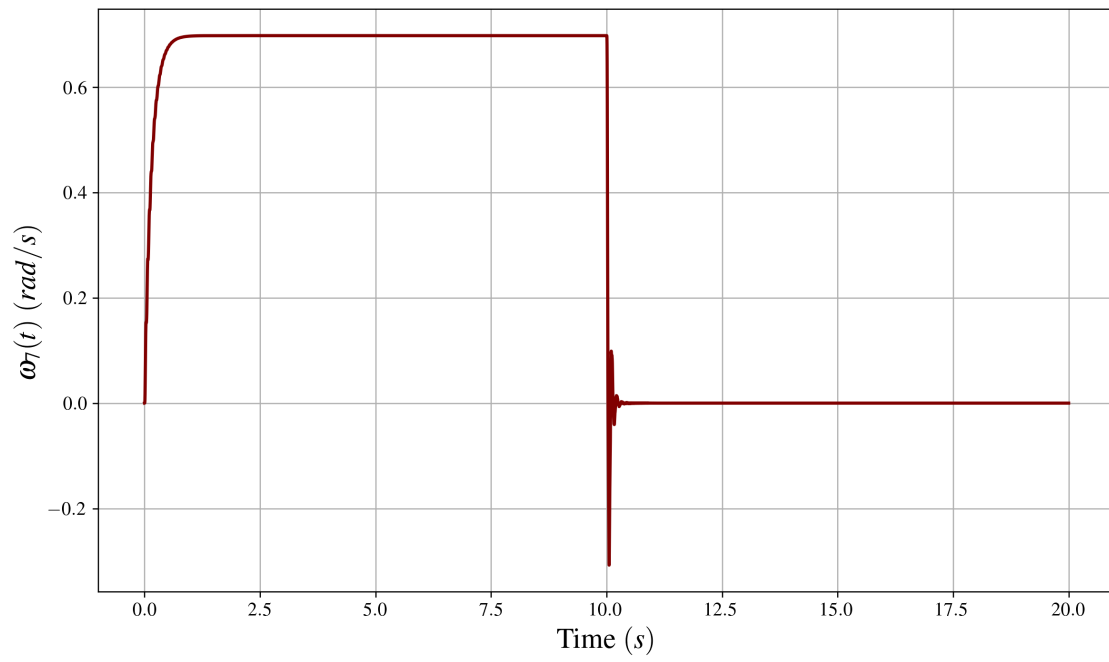
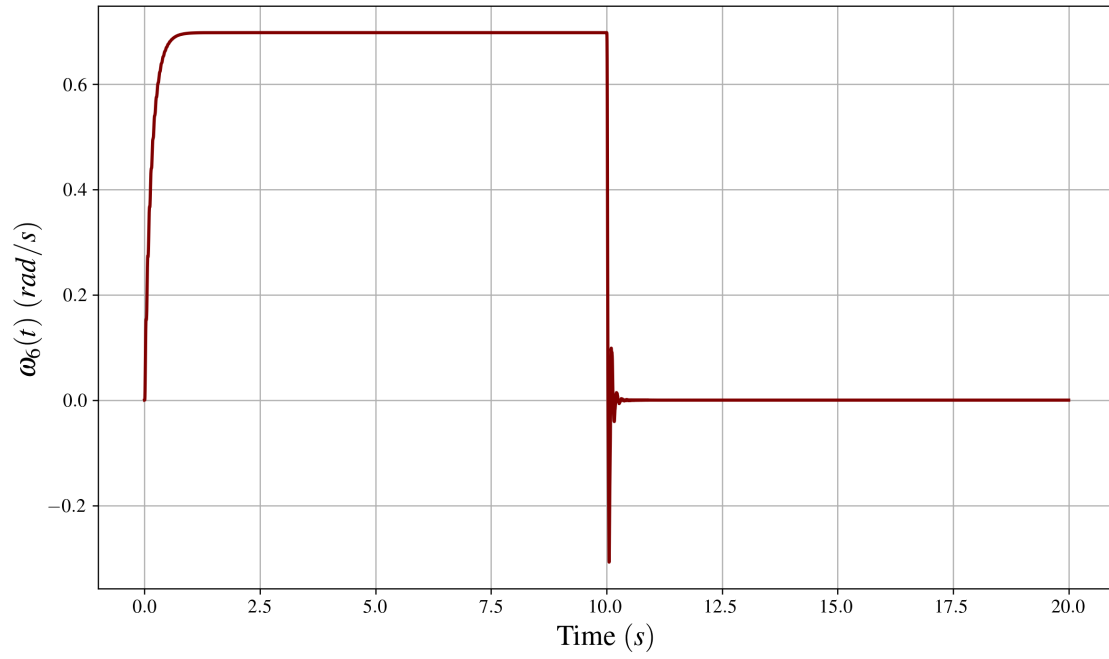
        ax.set_xlabel('Time $(s)$')
        ax.set_ylabel(fr'${omega_label}$ $(rad/s)$')

plt.show()

```







Verification

Part D

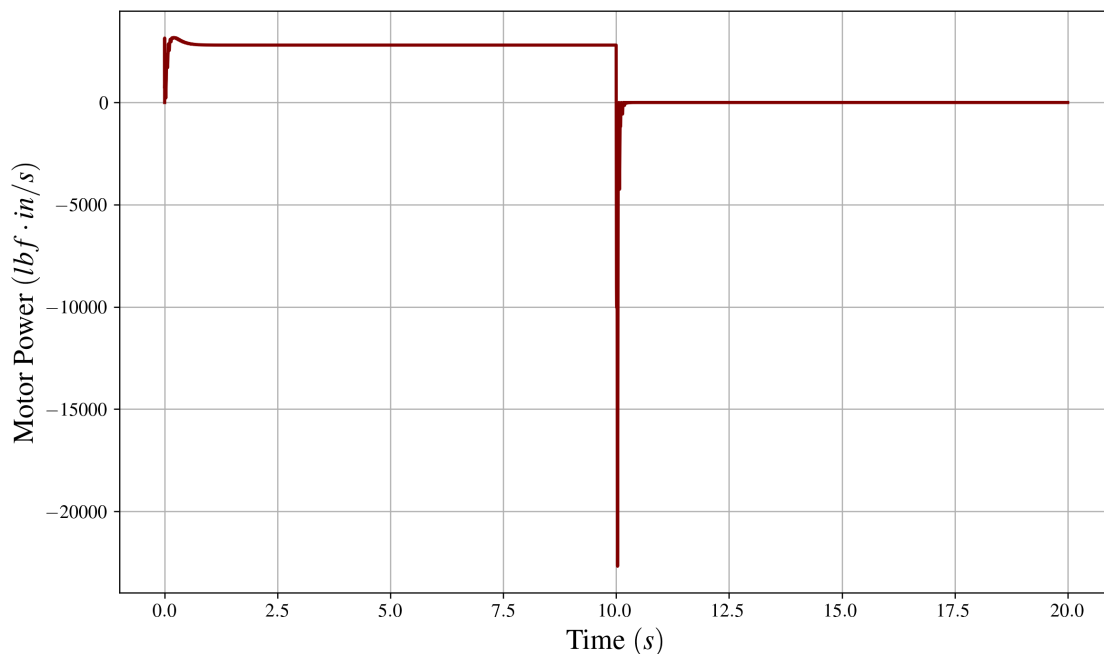
The results show that the motor velocity initially ramps up with an overshoot before arriving at a steady state value of 2.09424 rad/s . Upon the motor shutting off and turning into a damper, the system arrives at rest in about 0.25 s . There is a great deal of ringing in this model, which is not good because this means that the transient responses have high accelerations or jerks that tend to cause premature failure from fatigue or even monotonic fracture. The best approach at fixing this is to design the motor's control system to ramp down slowly when shutting off. The alternative would be to add more damping in the system, but this will increase energy loss.

Motor Power Power for a motor is defined as

$$P_{motor} = M_a(\omega_0)\omega_0$$

```
[29]: omega0 = sol[:, 4]
Ma_array = np.array([Ma(omega0_, t_) for omega0_, t_ in zip(omega0, t_array)])
P_motor = Ma_array*omega0

fig, ax = plt.subplots()
ax.plot(t_array, P_motor)
ax.set_xlabel('Time $(s)$')
ax.set_ylabel(r'Motor Power $(\text{lb}\cdot\text{in}/\text{s})$')
plt.show()
```



At steady state, the motor is not operating at its maximum possible power which occurs at $\omega_0 = 3.17 \text{ rad/s}$ with a value of $3175 \text{ lb}\cdot\text{in}/\text{s}$. Instead, it is operating at 2.09424 rad/s with a value of

2807 $\text{lbf} \cdot \text{in}/\text{s}$. The motor delivers just enough power to balance the load and overcome damping. This operating point is determined by the load's impedance and the inherent dynamics of the system, not by the maximum power available on the motor's characteristic curve.

Energy Conservation We can further verify the results by testing to see if the energy was conserved with the following relationship:

$$E_{\text{stored}}(t) = E_{\text{input}}(t) - E_{\text{output}}(t)$$

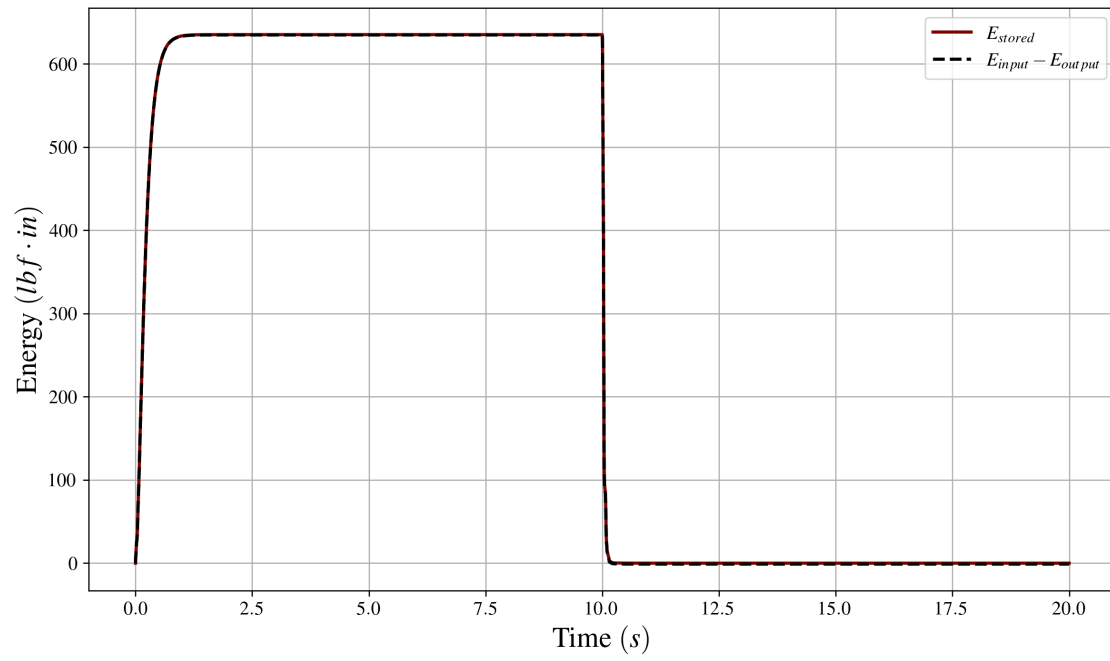
```
[30]: del_A, del_B, del_C, del_D, w0, w2, w3, w6, w7 = sol.T

KE = (
    1/2*I0_*w0**2 +
    1/2*(I2_ + I1_/T1_**2)*w2**2 +
    1/2*(I3_ + I4_*T2_**2 + I5_*T3_**2)*w3**2 +
    1/2*I6_*w6**2 +
    1/2*I7_*w7**2
)
PE = (
    1/2*k1_*del_A**2 +
    1/2*k1_*del_B**2 +
    1/2*k2_*del_C**2 +
    1/2*k2_*del_D**2
)
E_stored = KE + PE

E_input = cumulative_trapezoid(P_motor, t_array, initial=0)

P_damp = B_*(w6**2 + w7**2)
E_output = cumulative_trapezoid(P_damp, t_array, initial=0)

fig, ax = plt.subplots()
ax.plot(t_array, E_stored, label=r'$E_{\text{stored}}$')
ax.plot(t_array, E_input - E_output, label=r'$E_{\text{input}}-E_{\text{output}}$', ls='--')
ax.set_xlabel('Time $(s)$')
ax.set_ylabel(r'Energy $(\text{lbf}\cdot\text{in})$')
ax.legend()
plt.show()
```



The results clearly show that energy was conserved.