L8-3: Magnetic levitation example

CDS 110/ChE 105, Winter 2024
Richard M. Murray

This notebook contains the code used to create the magnetic levitation example in Lecture 8-1.

In [1]:
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

from math import pi

try:
    import control as ct
    print("python-control", ct.__version__)
except ImportError:
    !pip install control
    import control as ct
import control.optimal as opt
import control.flatsys as fs

python-control 0.10.0

The magnetic leviation system is governing by following equation:

\[
\ddot{z} = g - \frac{k_m k^2 A}{m} \frac{u^2}{z^2} - \frac{c}{m} \dot{z}
\]

and the system's output \(v_{ir}\) is the following:

\[
v_{ir} = k_T z + v_0
\]

In [2]:
# System dynamics
maglev_params = {
    'kT': 613.65,  # gain between position and voltage
    'v0': -16.18,  # voltage offset at zero position
    'm': 0.2,  # mass of ball, kg
    'g': 9.81,  # gravitational constant
    'kA': 1,  # electromagnet conductance
    'c': 1  # damping (added to improve visualization)
}

# gain on magnetic attractive force
maglev_params['km'] = 3.13e-3 * (maglev_params['m'] / 2) / maglev_params['kA']

def maglev_update(t, x, u, params):
    m, g, kA, km, c = map(params.get, ['m', 'g', 'kA', 'km', 'c'])
    return np.array([}
\[
x[1], \\
g = \text{km/m} \times (\text{kA} \times u[0])^2 / x[0]^2 - c \times x[1] 
\]

```python
def maglev_output(t, x, u, params):
    kT, v0 = map(params.get, ['kT', 'v0'])
    return np.array([kT * x[0] + v0])
```

```
maglev = ct.nlsys(
    maglev_update, maglev_output, params=maglev_params, name='maglev',
    inputs='Vu', outputs='Vy', states=['pos', 'vel']
)
```

```python
In [3]:
# Compute the equilibrium point that holds the ball at the origin
xeq, ueq = ct.find_eqpt(maglev, [0.02, 0], 0.2, y0=0)
print(f'\{{xeq=}}, \{ueq=}''', end='n----\n')

# Compute the linearization at that point
magP = ct.linearize(maglev, xeq, ueq)
print(magP, end='n----\n')

print('# Poles:', magP.poles())
print('# Zeros:', magP.zeros())
```

```
xeq=\text{array}(\{2.63668215e-02, 1.07930070e-25\}), \text{ueq=array}(\{2.08754147\})
----
<\text{StateSpace}>: \text{sys}[0]
Inputs (1): ['\text{u}[0]']
Outputs (1): ['\text{y}[0]']
States (2): ['\text{x}[0]', '\text{x}[1]']

A = [[ 0.             1. ],
     [744.07466586 -1.   ]]

B = [[ 0.               ],
     [-9.39861794        ]]

C = [[613.65  0.  ]]

D = [[0.]]

----
Poles: \{26.78231416+0.j -27.78231416+0.j\}
Zeros: []
```

```python
In [4]:
# Controller (analog circuit)
k1 = 0.5                        # gain set by gain pot
R1 = 22000                      # Internal resistor
R2 = 22000                      # Resistor plug-in
R = 2000; C = 1e-6              # RC plug-in

# Controller based on analog circuit
magC1 = -ct.tf([([R1 + R] * C, 1), [R * C, 1]) * k1 * R2/R1
magL1 = magP * magC1
```
Nyquist plot

```python
# Nyquist plot
plt.figure(figsize=[5, 3.75])
magP.name = "sys"
magL = magP * magC1
magL.name = "sys * ctrl"
c.t.nyquist_plot([magP, magL])
plt.gca().set_aspect('equal')
plt.suptitle("")
plt.title("Nyquist plot for maglev", fontsize=10)
plt.tight_layout()

# Bode plots
plt.figure(figsize=[5, 3.75])
magC1.name = "ctrl"
out = ct.bode_plot([magP, magL, magC1], np.logspace(0, 4), initial_phase=0)
axs = ct.get_plot_axes(out)
plt.suptitle("")
axs[0, 0].set_title("Bode plot for maglev", fontsize=10)
axs[0, 0].set_yscale('linear')
```

make some comment about how gain can be increased infinitely but there is a lower limit for stability

also plot/calculate gain/phase/stability margins

Out[6]: (0.06, 15.0)
# Default Bode plot (with no customization)

c.t.bode_plot([magP, magC1, magL])

In [7]:  # Default Bode plot (with no customization)
c.t.bode_plot([magP, magC1, magL])

Out[7]:  array([[list([<matplotlib.lines.Line2D object at 0x13051c770>, <matplotlib.
    lines.Line2D object at 0x13061ce00>, <matplotlib.lines.Line2D object at 0x1
    3061d520>])]
    ,
    list([<matplotlib.lines.Line2D object at 0x13061c950>, <matplotlib.
    lines.Line2D object at 0x13061d130>, <matplotlib.lines.Line2D object at 0x1
    3061d790>])],
    dtype=object)
In [8]: # Sensitivity function for closed loop system
   magS1 = ct.feedback(1, magL1)

In [9]: # Step response
   magT1 = ct.feedback(magL1)
   magT1.name = "closed loop system"
   ct.step_response(magT1).plot()

Out[9]: array([[[list([<matplotlib.lines.Line2D object at 0x1306f32f0>])]]],
dtype=object)
Try to improve performance by increasing DC gain

# System with gain increased
magC2 = magC1*5;  # increased gain
magL2 = magP * magC2;  # loop transfer function
magS2 = ct.feedback(1, magP * magC2);  # sensitivity function
magT2 = ct.feedback(magP * magC2, 1);  # closed loop response

# System with gain increased even more
magC3 = magC1*20;  # increased gain
magL3 = magP*magC3;  # loop transfer function
magS3 = ct.feedback(1, magP * magC3);  # sensitivity function
magT3 = ct.feedback(magP * magC3, 1);  # closed loop response

# name systems
magT1.name = "system 1"
magT2.name = "system 2"
magT3.name = "system 3"
magS1.name = "system 1"
magS2.name = "system 2"
magS3.name = "system 3"

# Plot step responses for different systems
colors = ['b', 'g', '#FF7F50']
for sys in [magT1, magT2, magT3]:
    ct.step_response(sys).plot(color=colors.pop())

# Bode plot for sensitivity function
plt.figure()
lines = ct.bode_plot([magS1, magS2, magS3], plot_phase=False)

# Add magnitude of 1
xdata = lines[0][0][0].get_xdata()
ydata = np.ones_like(xdata)
plt.plot(xdata, ydata, color='k', linestyle='--')

Out[10]: [<matplotlib.lines.Line2D at 0x131241e50>]

Step response for system 1, 2, 3
# Bode integral calculation

```python
omega = np.linspace(0, 1e6, 100000)
for name, sys in zip(['C1', 'C2', 'C3'], [magS1, magS2, magS3]):
    freqresp = ct.frequency_response(sys, omega)
    bodeint = np.trapz(np.log(freqresp.magnitude), omega)
    print(f"Bode integral for {name} = {bodeint}")

print("pi * sum[ Re(pk) ]", pi * np.sum(magP.poles()[magP.poles().real > 0]))
```

Bode integral for C1 = [84.10451626]
Bode integral for C2 = [83.96609718]
Bode integral for C3 = [83.44702558]
pi * sum[ Re(pk) ] (84.13912140726038+0j)