HW6: dcmtrs.ipynb

In [ ]:
```
import numpy as np # numerical library
import matplotlib.pyplot as plt # plotting library
%config InlineBackend.figure_format='retina' # high-res plots
import control.matlab as ctm # matlab layer for control systems library
import control as ct # use regular control library for a few things
ct.set_defaults('statesp', latex_repr_type='separate') # ABCD matrices
```

plant model

The A and B matrices for a state equation of the dynamics of the DC Motors System in the University of Washington's Control Systems Laboratory are below.

The state vector is taken to be `[i1 i2 theta1 omega1 theta2 omega2].T`

the input is vector taken to be

`[e1 e2].T`

where

`i1` = Current of drive motor (A)

`i2` = Current of the load motor (A)

`theta1` = Angular position of shaft 1 (rad)

`omega1` = Angular velocity of shaft 1 (rad/sec)

`theta2` = Angular position of shaft 2 (rad)

`omega2` = Angular velocity of shaft 2 (rad/sec)

`e1` = Drive motor amplifier input voltage

`e2` = Load motor amplifier input voltage

In [ ]:
```
# Drive motor and drive motor amplifier parameters
K1 = 99e-3      # Motor constant (V/(rad/sec))
R1 = 2.13       # Armature resistance (ohms)
Dm1 = 1.27e-4   # Motor damping constant (N*m/(rad/sec))
L1 = 0.686e-3   # Armature inductance (H)
Jm1 = 26.9e-6   # Motor inertia (kg*m**2)
Ka1 = 32.2      # Gain of amplifier gain for drive motor (V/V)
Ra1 = 0.2       # Resistance of amplifier for drive motor (ohms)

# Load motor and load motor amplifier parameters
K2 = 62e-3      # Motor constant (V/(rad/sec))
R2 = 1.2        # Armature resistance (ohms)
Dm2 = 60e-6     # Motor damping constant (N*m/(rad/sec))
L2 = 2.1e-3     # Armature inductance (H)
Jm2 = 24.38e-6  # Motor inertia (kg*m**2)
Ka2 = 32.2      # Amplifier gain for drive motor (V/V)
Ra2 = 0.2       # Amplifier resistance for drive motor (ohms)

# Other parameters
J1 = 1.25e-3    # Inertial load on theta1 shaft (kg*m**2)
J2 = 1.0e-3     # Inertial load on theta2 shaft (kg*m**2)
D1 = 42.35e-6   # Viscous friction coefficient for theta1 shaft(N*m/(rad/sec))
D2 = 42.35e-6   # Viscous friction coefficient for theta2 shaft (N*m/(rad/sec))
n = 5.0         # Gear ratio
Ks = 100        # Shaft stiffness (N*m/rad)

# Generate State Model Matrices
Jeq1 = J1 + n**2*Jm1
Jeq2 = J2 + Jm2
Deq1 = D1 + n**2*Dm1
Deq2 = D2 + Dm2

a11 = -(Ra1+R1)/L1
a14 = -n*K1/L1
a22 = -(Ra2 + R2)/L2
a26 = -K2/L2
a41 = n*K1/Jeq1
a43 = -Ks/Jeq1
a44 = -Deq1/Jeq1
a45 = Ks/Jeq1
a62 = K2/Jeq2
a63 = Ks/Jeq2
a65 = -Ks/Jeq2
a66 = -Deq2/Jeq2
b11 = Ka1/L1
b22 = Ka2/L2

A = np.array(
```

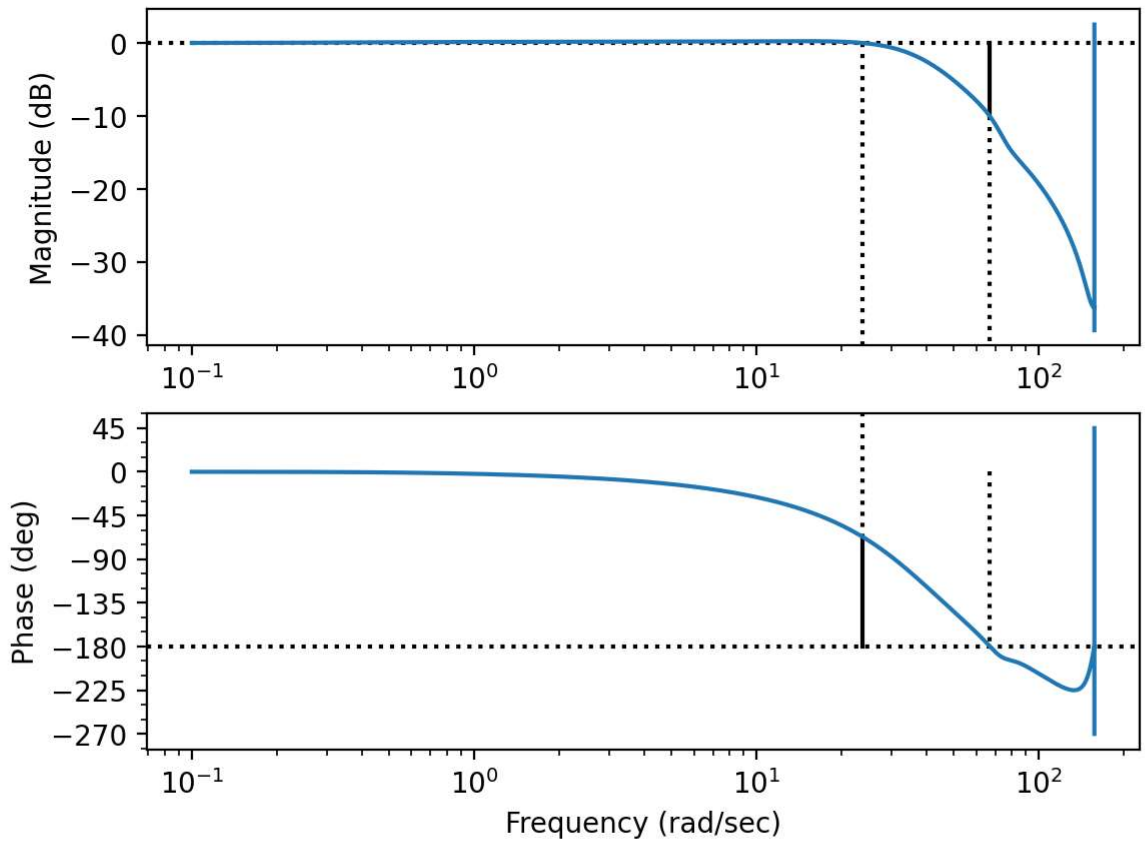
```
    [[a11,     0,        0,      a14,       0,        0],
     [0,     a22,        0,        0,       0,      a26],
     [0,       0,        0,        1,       0,        0],
     [a41,     0,      a43,      a44,     a45,        0],
     [0,       0,        0,        0,       0,        1],
     [0,     a62,      a63,        0,     a65,      a66]])
B = np.array(
    [[b11,   0],
     [0,   b22],
     [0,     0],
     [0,     0],
     [0,     0],
     [0,     0]])
```

1a.

```
In [ ]: T = 0.02
C = np.array([[0, 0, 0, 0, 1, 0]])
D = np.array([[0, 0]])
plant = ctm.ss(A, B, C, D, inputs=['e1', 'e2'], outputs=['theta2'])
plant_simulator = ctm.c2d(plant, T, 'zoh')
controller = ctm.tf([55, -80.03325, 25.29849575], [100, -120, 20], T, inputs=['e'], outputs=['e1'])
sum = ct.summing_junction(['theta2_ref', '-theta2'], 'e')

sys = ct.interconnect([controller, plant_simulator, sum], inputs=['theta2_ref'], outputs=['theta2'])

ctm.bode(sys, margins=True);
```

```
c:\Users\YENPANG_HUANG\AppData\Local\Programs\Python\Python311\Lib\site-packages\control\iosys.py:1503: UserWarning: Unused input(s) in InterconnectedSystem: (1, 1)=sys[251]$sampled.e2
  warn(msg)
```

Gm = 10.05 dB (at 67.02 rad/s), Pm = 112.93 deg (at 23.74 rad/s)

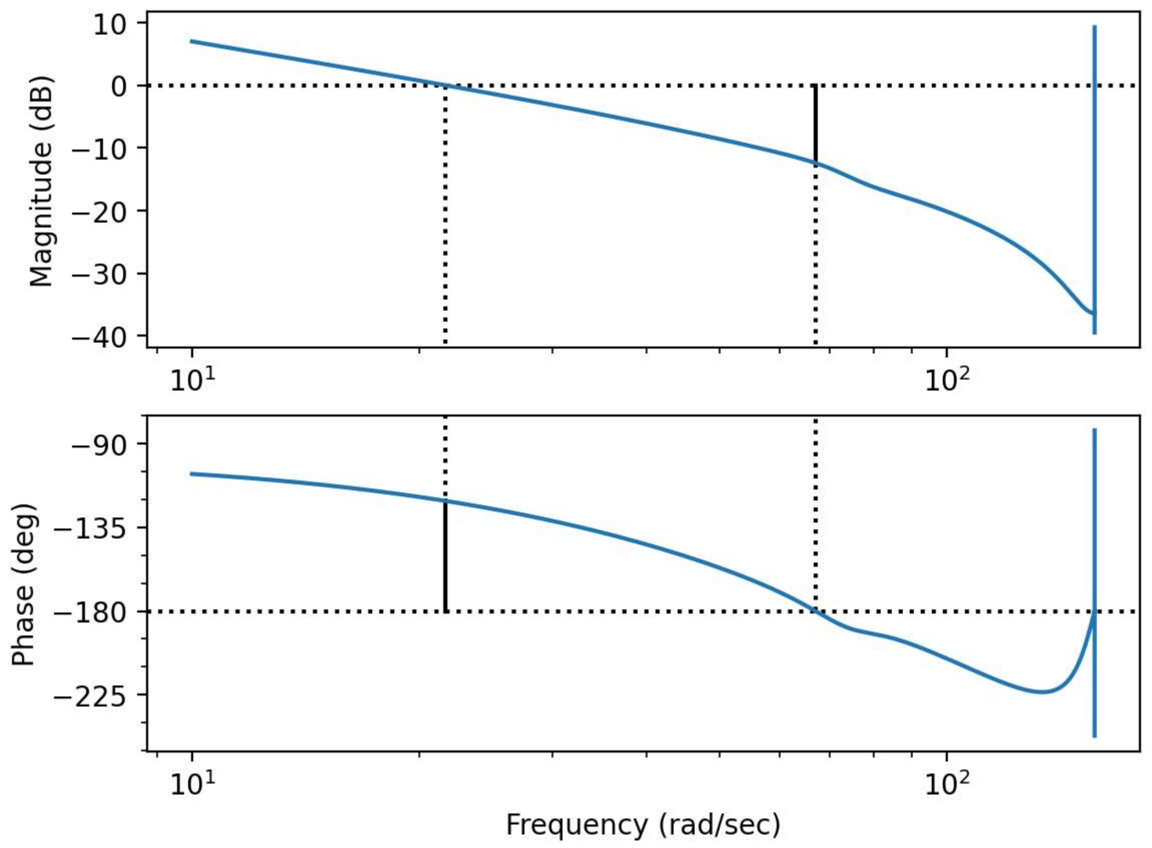


b.

```
In [ ]: T = 0.02
C = np.array([[0, 0, 1, 0, 0, 0],
              [0, 0, 0, 0, 1, 0]])
D = 0
plant = ctm.ss(A , B, C, D)
plant_simulator = ctm.c2d(plant, T, 'zoh')
controller = 55/100*ctm.tf([1, -0.46415], [1, -0.2], T)*ctm.tf([1, -0.991], [1, -1], T)
sys = controller*plant_simulator[1, 0]
nyquist = np.pi/T
gm, pm, wcg, wcp = ctm.margin(sys)
ctm.bode(sys, omega_limits=(10, nyquist), margins=True);
```

```
c:\Users\YENPANG_HUANG\AppData\Local\Programs\Python\Python311\Lib\site-packages\control\lti.py:181: UserWarning: __call__: evaluation above Nyquist frequency
  warn("__call__: evaluation above Nyquist frequency")
```

Gm = 12.42 dB (at 67.02 rad/s), Pm = 59.14 deg (at 21.66 rad/s)



b.

In [ ]:
```
k = 10**(12.42/20)
print(k)
```

```
4.178303666466218
```

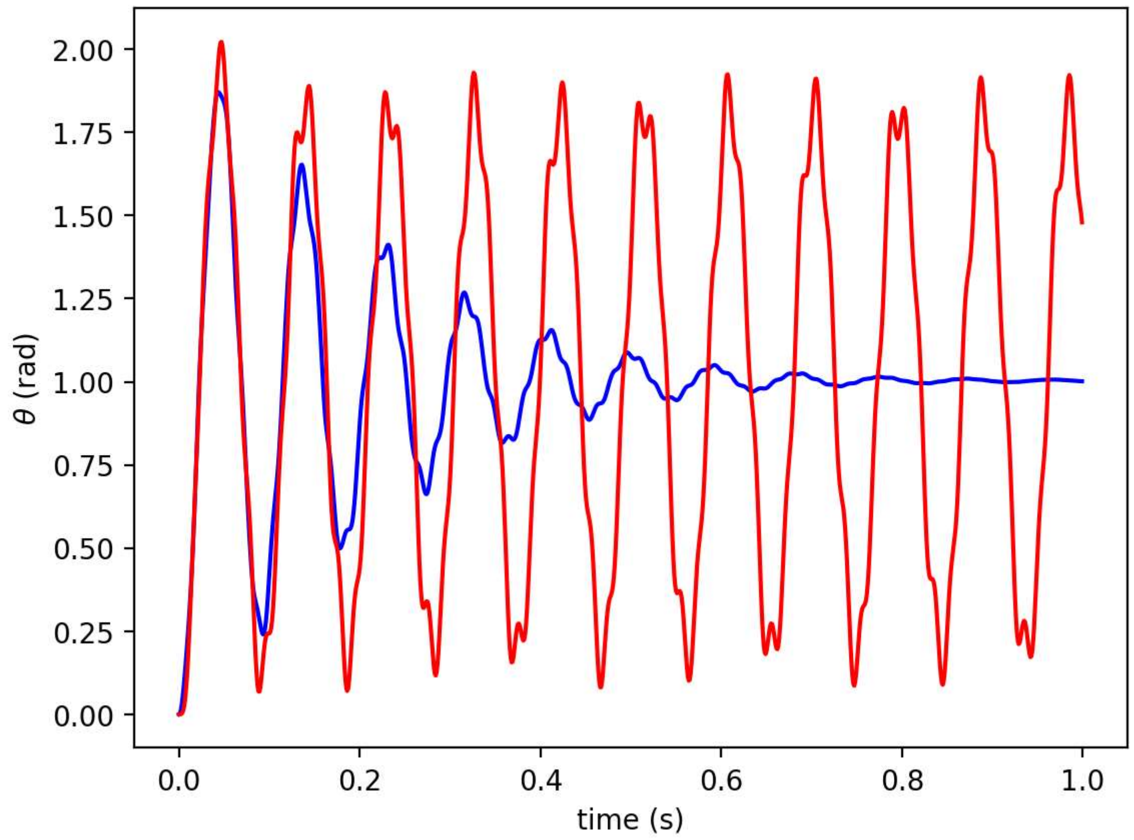
c.

In [ ]:
```
def sampled_data_system(sysd, simulation_dt):
    assert ct.isdtime(sysd, True), "sysd must be discrete-time"
    sysd = ct.ss(sysd) # convert to state-space if not already
    nsteps = int(round(sysd.dt / simulation_dt))
    assert np.isclose(nsteps, sysd.dt/simulation_dt), \
        "simulation_dt must be an integral multiple of sysd.dt"
    st = 0
    y = np.zeros((sysd.noutputs, 1))
    def updatefunction(t, x, u, params):
        nonlocal st
        if st == 0: # is it time to sample?
            x = sysd._rhs(t, x, u)
        st += 1
        if st == nsteps:
            st = 0
        return x
    def outputfunction(t, x, u, params):
        nonlocal y
        if st == 0: # is it time to sample?
            y = sysd._out(t, x, u)
        return y
    return ct.ss(updatefunction, outputfunction, dt=simulation_dt,
                 name=sysd.name, inputs=sysd.input_labels,
                 outputs=sysd.output_labels, states=sysd.state_labels)
```

In [ ]:
```
simulation_dt = .001

# Theta 1
plant_simulator1 = ctm.ss(ct.c2d(plant[0,0], simulation_dt), inputs='g', outputs='theta1')
controller = ctm.ss(controller, inputs='e', outputs='u')
controller_simulator = sampled_data_system(controller, simulation_dt)
gain = ctm.tf2ss(gm, 1, inputs='u', outputs='g')
error_sum1 = ct.summing_junction(inputs=['theta1_ref', '-theta1'], outputs='e')

sys1 = ct.interconnect((controller_simulator, error_sum1, plant_simulator1, gain),
                       inputs='theta1_ref', outputs='theta1')

# Theta 2
plant_simulator2 = ctm.ss(ct.c2d(plant[1,0], simulation_dt), inputs='g', outputs='theta2')
controller = ctm.ss(controller, inputs='e', outputs='u')
controller_simulator = sampled_data_system(controller, simulation_dt)
gain = ctm.tf2ss(gm, 1, inputs='u', outputs='g')
error_sum2 = ct.summing_junction(inputs=['theta2_ref', '-theta2'], outputs='e')

sys2 = ct.interconnect((controller_simulator, error_sum2, plant_simulator2, gain),
                       inputs='theta2_ref', outputs='theta2')
```

In [ ]:
```
time = np.arange(0, 1, simulation_dt)
step_input = np.ones_like(time)
t1, y1  = ct.input_output_response(sys1, time, step_input)
t2, y2  = ct.input_output_response(sys2, time, step_input)
plt.plot(t1, y1, 'b', t2, y2, 'r')
```

```
plt.xlabel('time (s)')
plt.ylabel(r'$\theta$ (rad)');
```



d.

In [ ]:
```
delay = pm*np.pi/180/wcp
```

e.

In [ ]:
```
def time_delay_system(delay, dt, inputs=1, outputs=1, **kwargs):
    assert delay >= 0, "delay must be greater than or equal to zero"
    n = int(round(delay/dt))
    ninputs = inputs if isinstance(inputs, (int, float)) else len(inputs)
    assert ninputs == 1, "only one input supported"
    A = np.eye(n, k=-1)
    B = np.eye(n, 1)
    C = np.eye(1, n, k=n-1)
    D = np.zeros((1,1))
    return ct.ss(A, B, C, D, dt, inputs=inputs, outputs=outputs, **kwargs)
```

In [ ]:
```
simulation_dt = 0.001
plant_simulator = ctm.ss(ct.c2d(plant[1,0], simulation_dt), inputs='u', outputs='theta2')
controller = ctm.ss(controller, inputs='e_delay', outputs='u')
controller_simulator = sampled_data_system(controller, simulation_dt)
error_sum = ct.summing_junction(inputs=['theta2_ref', '-theta2'], outputs='e')
delayer = time_delay_system(delay, simulation_dt, inputs='e', outputs='e_delay')
sys_delay = ct.interconnect((controller_simulator, error_sum, plant_simulator, delayer),
                       inputs='theta2_ref', outputs='theta2')

time = np.arange(0, 1, simulation_dt)
step_input = np.ones_like(time)
t_delay, y_delay  = ct.input_output_response(sys_delay, time, step_input)
plt.plot(t_delay, y_delay)
```

Out[ ]: [<matplotlib.lines.Line2D at 0x1f36d6740d0>]