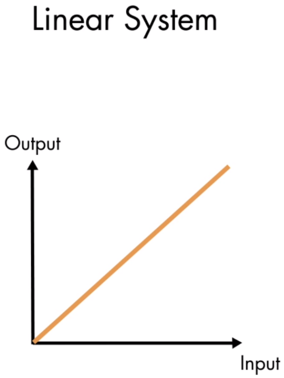
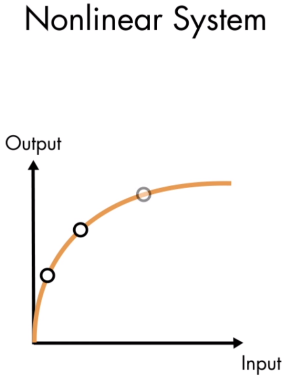
5ELEN018W - Robotic Principles
Lecture 9–10: More on Bode Plots, Linearisation and PID Control Implementation

Dr Dimitris C. Dracopoulos

Linear vs Nonlinear Systems

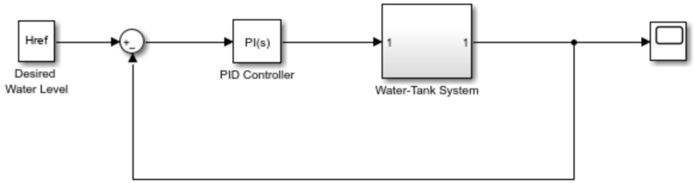


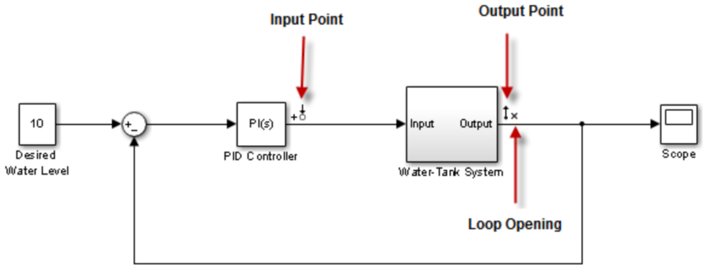


- In real life all systems are nonlinear, however many of them can be linearised about their operation point.

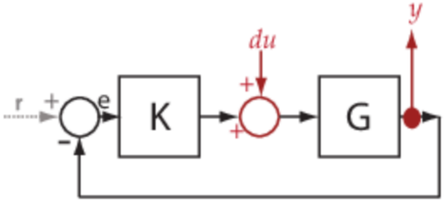
Linear systems are easier to analyse and prove mathematically their behaviour and properties.

How to Specify the Portion of a Model to Linearise





Example 1

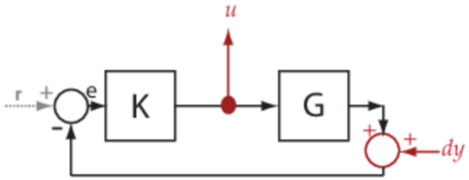


Show that the transfer function for $\frac{y}{du}$ is given by $\frac{G}{1+GK}$

$$(y * K + du) * G = u \tag{1}$$

Solve manually for $\frac{y}{du}$ or using Sympy. **Both ways were shown in the lecture!**

Example 2



Show that the transfer function for $\frac{u}{dy}$ is given by $\frac{-K}{1+KG}$

$$(u * G + dy) * K = u \qquad (2)$$

Solve manually for $\frac{u}{dy}$ or using Sympy. **Both ways were shown in the lecture!**

Bode Plots Revisited

Consider the watertank of the previous lecture and tutorial:

$$\frac{d}{dt} Vol = A\frac{dH}{dt} = bV - \alpha\sqrt{H} \qquad (3)$$

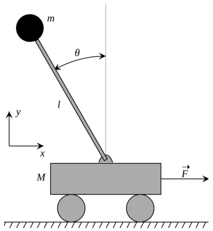
- $A = 20$
- $\alpha = 2$
- $b = 5$
- $H_{ref} = 10$ (reference signal - desired)
- Initial condition $H(0) = 1$
- PID parameters: $P = 1.599340, P_I = 0.079967, P_D = 0.$

Plot the Bode diagram.

You need to create a subsystem for the part of the plant that you would like to generate the Bode plot! The output of the PID controller (input of the subsystem) should be the first signal to the Bode plot block. The output of the subsystem should be the second signal selected for the Bode plot block.

The Inverted Pendulum Control Problem - PID Control

Consider the inverted pendulum control problem we have seen in the last tutorial implementing a PID controller using Simulink:



$$(M+m)\,\ddot{x} + m\ell\cos\theta\,\ddot{\theta} - m\ell\dot{\theta}^2\sin\theta = F - b_c\dot{x}, \tag{4}$$

$$m\ell\cos\theta\,\ddot{x} + m\ell^2\,\ddot{\theta} - mg\ell\sin\theta = -b_p\dot{\theta}. \tag{5}$$

Implement a PID controller for it using Python (not Simulink). For the values of the parameters see the last tutorial specification.

PID Controller in Python

```
from sympy import *

g=9.81
m=0.1
M=1
l=0.5
g=9.81
b_c = 0.1
b_p = 0.01

#theta_dotdot, theta_dot, theta, x_dotdot, x_dot, x, M, m, l, b_c, b_p, F =
# symbols('theta_dotdot, theta_dot, theta, x_dotdot, x_dot, x, M, m, l, b_c, b_p, F')

theta_dotdot, theta_dot, theta, x_dotdot, x_dot, x, F = \
                     symbols('theta_dotdot, theta_dot, theta, x_dotdot, x_dot, x, F')

# equation 1
eq1 = (M + m)*x_dotdot + m*l*cos(theta)*theta_dotdot - (m*l*theta_dot**2)*sin(theta) - F + b_c*x_dot

# equation 2
eq2 = m*l*cos(theta)*x_dotdot + (m*(l**2))*theta_dotdot - m*g*l*sin(theta) +  b_p*theta_dot

sol = nonlinsolve([eq1, eq2], (x_dotdot, theta_dotdot))
sol = simplify(sol)
print(list(sol)[0][0])

# x_dotdot formula
s1 = list(sol)[0][0]
# theta_dotdot formula
s2 = list(sol)[0][1]

print(s2)

dt = 0.01  # time step for the simulation
```

PID Controller in Python (cont'ed))

```python
# Implements the dynamic system (plant) - system inputs are
# action_u and the current state (x, x_dot, theta_theta_dot).
# The function returns the new state of the plant
# (new_x, new_x_dot, new_theta, new_theta_dot).
def plant(action_u, x1, x1_dot, theta1, theta1_dot):
    # equations of motion:
    #(M + m)\,\ddot{x} + m\ell\cos\theta\,\ddot{\theta} - m\ell\dot{\theta}^{2}\sin\theta &= F - b_c\dot{x
    # m\ell\cos\theta\,\ddot{x} + m\ell^{2}\,\ddot{\theta} - mg\ell\sin\theta &= - b_p\dot{\theta}.

    #action_u is F
    # m=0.1, M=1, l=0.5, g=9.81, b_c = 0.1, b_p = 0.01

    new_x_dot_dot = s1.subs({F:action_u, x_dot:x1_dot, x:x1, theta_dot:theta1_dot, theta:theta1})
    new_theta_dot_dot = s2.subs({F:action_u, x_dot:x1_dot, x:x1, theta_dot:theta1_dot, theta:theta1})

    # new angular velocity
    new_theta_dot = theta1_dot + dt*new_theta_dot_dot
    # new angle theta
    new_theta = theta1 + dt*new_theta_dot

    # new velocity x_dot
    new_x_dot = x1_dot + dt*new_x_dot_dot
    # new position x
    new_x = x1 + dt*new_x_dot

    return  new_x, new_x_dot, new_theta, new_theta_dot
```

PID Controller in Python (cont'ed))

```python
theta1 = 0.4  # current speed initialised to 0
previous_theta = 0  # the speed at the previous time step

theta1_dot = 0 # initial angular velocity
x1 = 0 # initial position
x1_dot = 0 # initial speed

start_time = 0.0
end_time = 10.0

current_time = start_time

theta_ref = 0  # the desired angle for balancing

K_p = -71.39  # proportional gain
K_i = -90    # integral gain
K_d = -9   # derivative gain

previous_error = 0.0
integral = 0

# log values to file
f = open('myfile.txt', "w")
# write initial values to the file
f.write(f'{theta1}  {current_time}\n')
```

PID Controller in Python (cont'ed))

```
''' Simulate the system operation from the beginning till
    the end of the siimulation '''
while current_time <= end_time:
    error = theta_ref - theta1

    # I(integral) component of the PID controller
    integral = integral + error*dt

    # (D)erivative component of the PID controller
    deriv = (error - previous_error)/dt

    # the output (action) of the PID controller
    action = K_p*error + K_i*integral + K_d*deriv

    # remember the last error when the previous
    # action was applied to the plant
    previous_error = error

    # apply the new action to the plant to calculate
    # the new state
    x1, x1_dot, theta1, theta1_dot = plant(action, x1, x1_dot, theta1, theta1_dot)

    print(f'Time: {current_time} Action: {action}, Angle={theta1}')
    f.write(f'{theta1}  {current_time}\n')

    # advance the time
    current_time += dt

f.close()
```

Plotting the Error

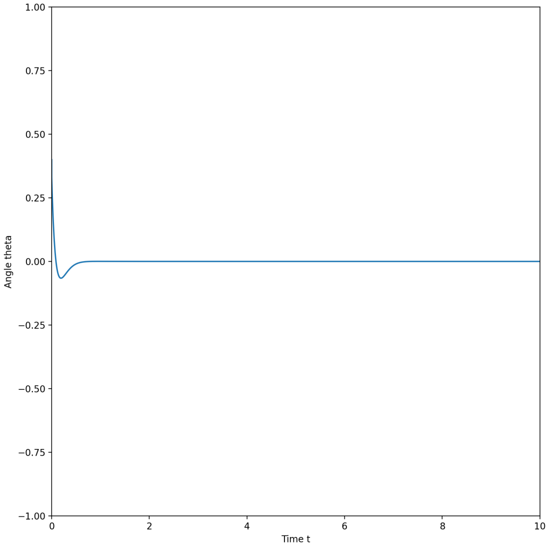
```
import matplotlib.pyplot as plt

f = open('myfile.txt')
v = []
time =[]
for i in f:
    [y, t] = i.split()
    #print(y, t)
    v.append(float(y))
    time.append(float(t))

print(v)
plt.plot(time, v)
plt.axis([0, 10, -1, 1])
plt.xlabel('Time t')
plt.ylabel('Angle theta')

plt.show()
```

Plotting the Error (cont'ed)



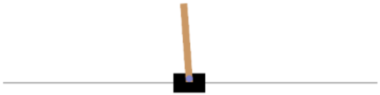
The Gymnasium Environment

Installing Gymnasium
Execute in a a terminal (inside Anaconda, if you are using Anaconda):

```
pip install swig
pip install gymnasium[toy_text]
pip install gymnasium[box2d]
```

You can follow the link below for more detailed instructions:
https://github.com/Farama-Foundation/Gymnasium

The Cart Pole Problem in Gymnasium



A simplified version of the inverted pendulum problem.

- Decision: Force to the left or right.

Action Space:

- 0: Push cart to the left
- 1: Push cart to the right

The Cart Pole System (cont'd)

Observation Space:
A 4-dimensional vector:

- Cart position x
- Cart velocity $\dot{x}$
- Pole Angle θ
- Angular velocity $\dot{\theta}$

https://gymnasium.farama.org/environments/classic_control/cart_pole/

Example of Robot Agent (Random Policy) for Cart Pole Balancing

```python
import gymnasium as gym
import time

# Create our training environment - a cart with a pole that needs balancing
env = gym.make("CartPole-v1", render_mode="human")

# Reset environment to start a new episode
observation, info = env.reset()
# observation: what the agent can "see" - cart position, velocity, pole angle, etc.
# info: extra debugging information (usually not needed for basic learning)

episode_over = False
total_reward = 0

while not episode_over:
    # Choose an action: 0 = push cart left, 1 = push cart right
    action = env.action_space.sample()  # Random action for now - real agents will be smarter!

    # Take the action and see what happens
    observation, reward, terminated, truncated, info = env.step(action)

    # reward: +1 for each step the pole stays upright
    # terminated: True if pole falls too far (agent failed)
    # truncated: True if we hit the time limit (500 steps)

    total_reward += reward
    episode_over = terminated or truncated

print(f"Episode finished! Total reward: {total_reward}")
time.sleep(3)  # wait for 3 secs

env.close()
```

https://gymnasium.farama.org/introduction/basic_usage/

PID Controller for Cart Pole Balancing in Gymnasium

Write some code to implement a PID controller for the cart-pole system and apply it in the Gymnasium environment (tutorial exercise).

PID Controller for Cart Pole Balancing in Gymnasium (cont'ed))

```python
import gymnasium as gym
import time

# Create our training environment - a cart with a pole that needs balancing
env = gym.make("CartPole-v1", render_mode="human")

# Reset environment to start a new episode
observation, info = env.reset()
# observation: cart position, velocity, pole angle, angular velocity
# info: extra debugging information (usually not needed for basic learning)

episode_over = False
total_reward = 0

#K_p = 135  # proportional gain
#K_i = 96.5    # integral gain
#K_d =  47.5  # derivative gain
K_p = -150
K_i = -110
K_d = -50

previous_error = 0.0
integral = 0
theta_ref = 0  # the desired balancing angle

dt = 0.01  # the time interval between each step
```

PID Controller for Cart Pole Balancing in Gymnasium (cont'ed))

```python
while not episode_over:
    error = theta_ref - observation[2]

    # I(integral) component of the PID controller
    integral = integral + error*dt
    # the above line can be replaced with the line below as the simulator
    # takes sufficiently small steps so the integral will be the same
    #integral = integral + error

    # (D)erivative component of the PID controller - angular velocity
    deriv = (error - previous_error)/dt
    # the following can replace the line above as obsrvation[3] is the angular velocity
    #deriv = -observation[3]

    # the output (action) of the PID controller
    action = K_p*error + K_i*integral + K_d*deriv

    # remember the last error when the previous
    # action was applied to the plant
    previous_error = error

    # debug
    print(f'Angle:{observation[2]}, angular_velocity: {observation[3]}, Action: {action}')

    # the action is binary in this version of the problem
    if action >= 0:
        action = 1
    else:
        action = 0
```

PID Controller for Cart Pole Balancing in Gymnasium (cont'ed))

```
        # Take the action and see what happens
        observation, reward, terminated, truncated, info = env.step(action)

        # reward: +1 for each step the pole stays upright
        # terminated: True if pole falls too far (agent failed)
        # truncated: True if we hit the time limit (500 steps)
        total_reward += reward
        episode_over = terminated or truncated

    # the loop has terminated - print the results
    print(f"Episode finished! Total reward: {total_reward}")
    time.sleep(3)  # wait for 3 secs

    env.close()
```