Podstawy Sterowania Optymalnego - Labolatorium 2

Modelowanie układów liniowych przy pomocy zmiennych stanu.

Prowadzący: mgr inż. Krzysztof Hałas
Wykonał: Ryszard Napierała

23 Październik 2021

1 Zadanie 2

1. **W języku Python zaimportować biblioteki *numpy, scipy.signal, scipy.integrate.solve_ivp i matplotlib.pyplot*.**

```python
from scipy import integrate, signal
import matplotlib.pyplot as plt
import numpy as np
```

2. **Wprowadzić wartości** k_p, T, A, B, C, D

```python
kp = 3
T = 2
A = -1/T
B = kp/T
C = 1
D = 0
```

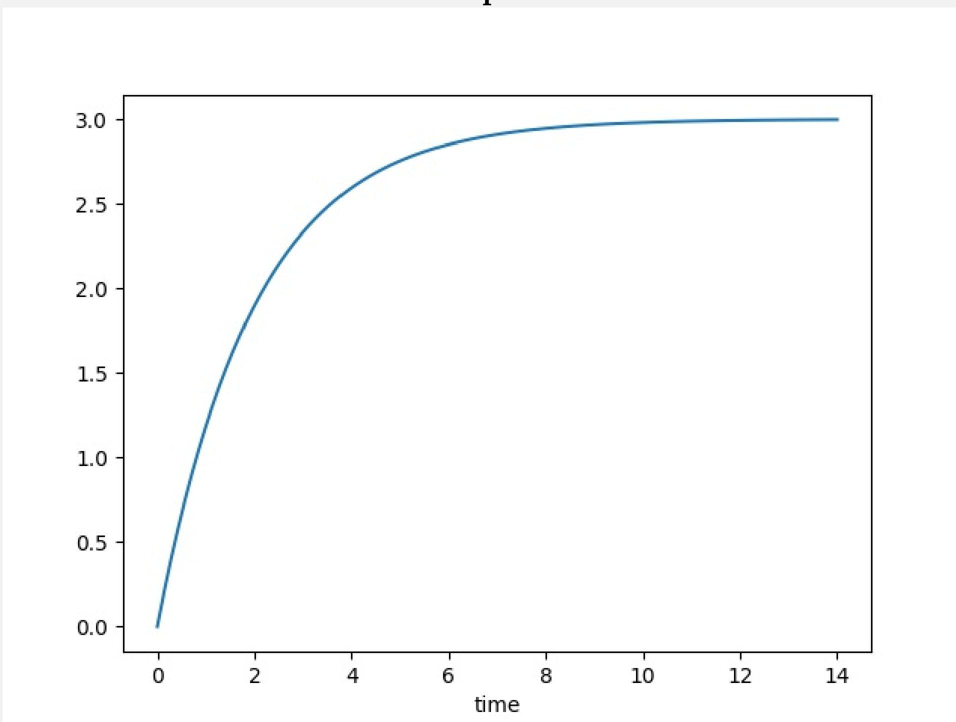
3. **Zapisać system w postaci transmitancji wykorzystując przykładowo polecenie *signal.TransferFunction(num,den)*.**

```python
G1 = signal.TransferFunction([kp], [T, 1])
```

4. **Wyznaczyć odpowiedź skokową układu (polecenie *signal.step(sys)*) oraz utworzyć jej wykres, przykładowo poleceniem *matplotlib.pyplot*.**

```python
x, y = signal.step(G1)
plt.plot(x, y)
plt.xlabel('time')
```

Output:

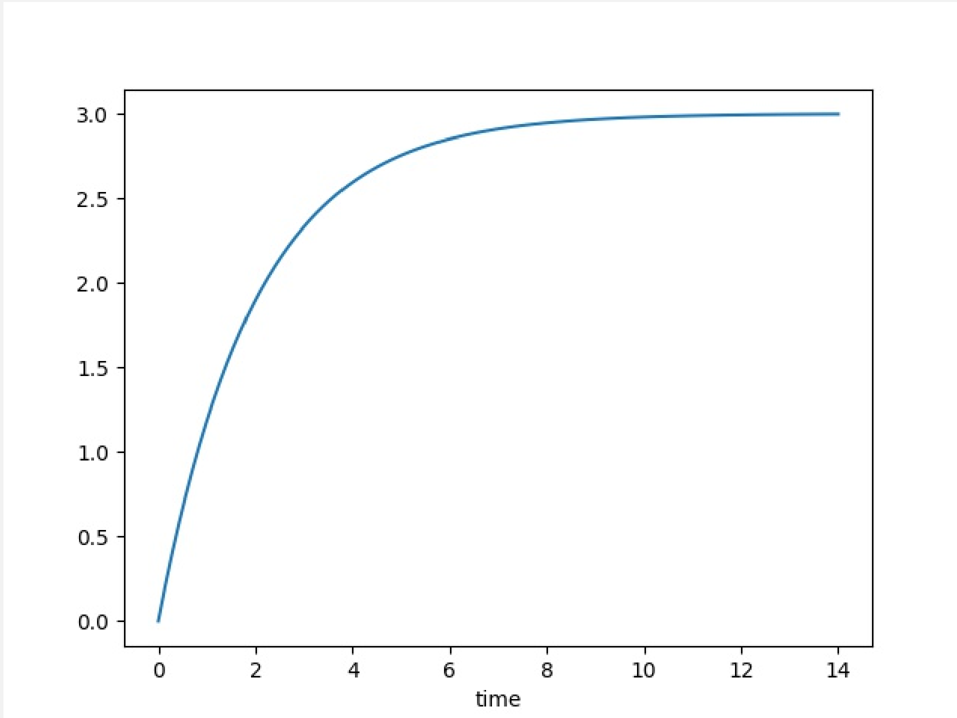


Czy odpowiedź skokowa odpowiada teoretycznym założeniom?
Odpowiedź skokowa odpowiada teoretycznym założeniom, jest zgodna z oczekiwaniami dla obiektu inercyjnego I-go rzędu.

5. **Zapisać system w postaci fazowych zmiennych stanu wykorzystując polecenie *signal.StateSpace(A,B,C,D)*. Wyznaczyć i wykreślić odpowiedź skokową układu.**

```python
sys = signal.StateSpace(A, B, C, D)
x, y = signal.step(sys)
plt.plot(x, y)
plt.xlabel('time')
```

Output:



6. **Utworzyć funkcję *model(t,y)* opisującą dynamikę systemu, przyjmującą jako argu- menty aktualny czas (t) oraz aktualny stan (y). Przyjąć $u(t) = \mathbb{1}(t)$.**

```
def u(t):
    return np.array([1.0])

def model(t, y, A, B, u):
    dy = A@y + B@u(t)
    return dy
```

7. **Utworzyć tablicę wartości czasu $t \in (0, 15)$.**

```
t_eval = np.linspace(0,15,100)
```

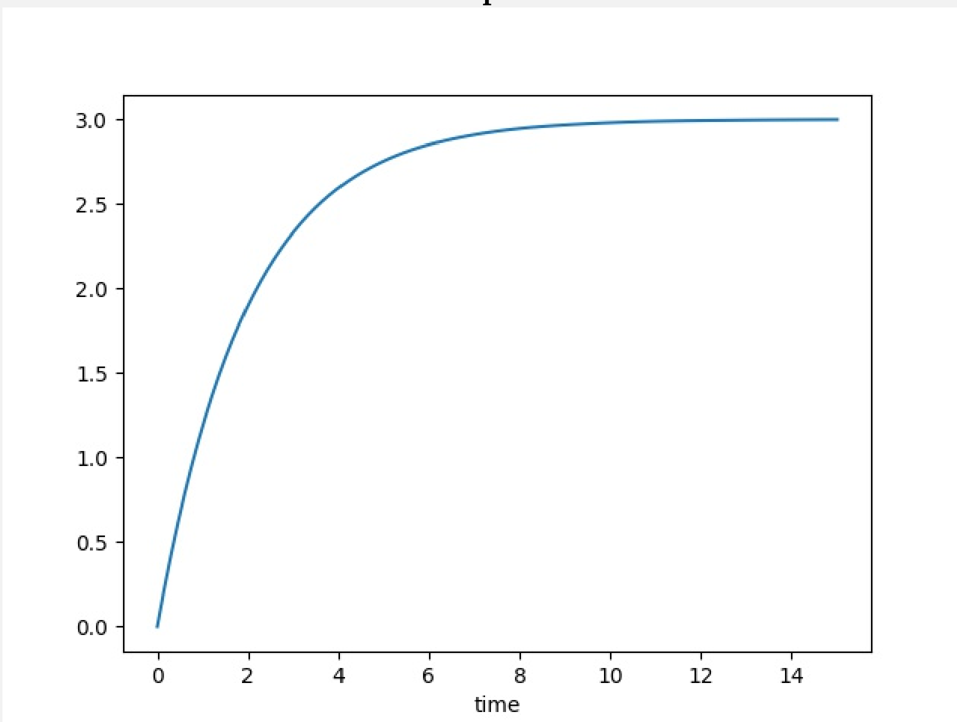
8. **Wyznaczyć rozwiązanie równania dla horyzontu czasowego z poprzedniego podpunktu i zerowych warunków początkowych. Wykorzystać komendę solve ivp z odpowiednią wartością parametru *t_eval*. Upewnić się, że model z utworzony w podpunkcie, jest zgodny z wymaganiami funkcji *solve_ivp*.**

```
result = integrate.solve_ivp(model, [0,15], [0], t_eval=t_eval, args=(A, B,
 ↪  u,))
```

9. **Wykreślić rozwiązanie równania dla pobudzenia skokiem jednostkowym, wykorzystu- jące polecenie *matplotlib.pyplot*. Należy pamiętać o ustawieniu takich samych rozmia- rów odpowiednich zmiennych (funkcja *reshape*).**

```
plt.plot(result.t, result.y[0])
plt.xlabel('time')
```

Output:



10. **Porównać odpowiedzi skokowe wszystkich trzech reprezentacji (systemu opisanemu w postaci transmitancji, w postaci zmiennych stanu oraz bezpośredniego rozwiązania równania różniczkowego).**
    Odpowiedzi skokowe weszystkich trzech reprezentacji są jednakowe.

2 Zadanie 3

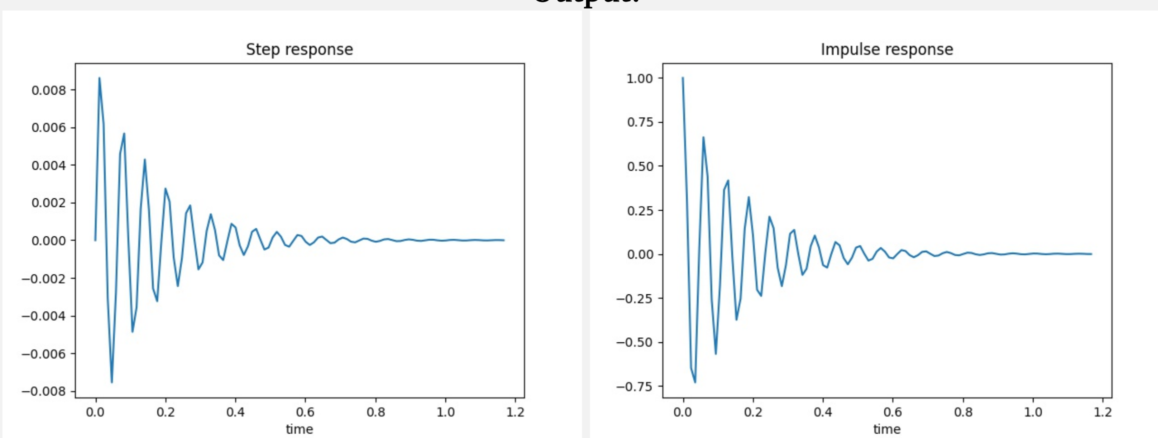
1. **W języku Python zamodelować układ dynamiczny przedstawiony na rys. 1 za pomocą transmitancji operatorowej oraz wykreślić jego odpowiedzi skokową i impulsową. Przyjąć następujące wartości zmiennych** $R = 12\Omega$, $L = 1H$ **oraz** $C = 100\mu F$ **.**

```
R = 12
L = 1
Cap = 0.000_1
num = [1, 0]
den = [L, R, 1/Cap]
G1 = signal.TransferFunction(num, den)
x, y = signal.step(G1)
```

```python
plt.plot(x, y)
plt.title('Step response')
plt.xlabel('time')

x, y = signal.impulse(G1)
plt.plot(x, y)
plt.title('Impulse response')
plt.xlabel('time')
```

Output:

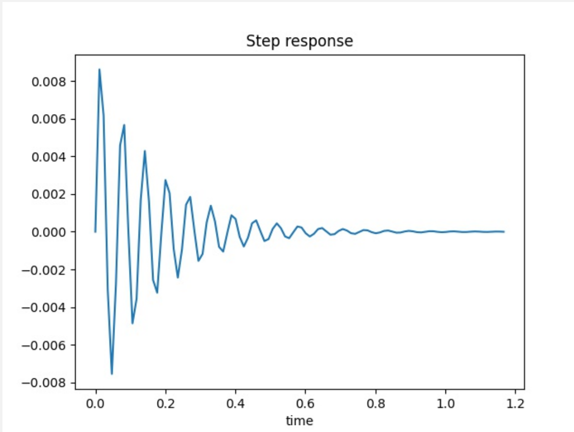


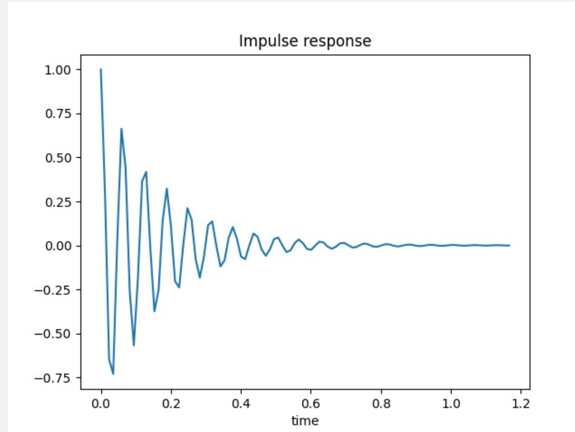
2. **W języku Python zamodelować ten sam układ przy pomocy zmiennych stanu oraz wykreślić jego odpowiedzi czasowe. Porównać z wykresami z poprzedniego podpunktu**

```python
A = np.array([
    [0, 1],
    [-1/(L*Cap), -R/L]
])
B = np.array([[0], [1/L]])
C = np.array([0, 1])
D = np.array([0])
sys1 = signal.StateSpace(A, B, C, D)
x, y = signal.step(sys1)
plt.plot(x, y)
plt.title('Step response')
plt.xlabel('time')

x, y = signal.impulse(G1)
plt.plot(x, y)
plt.title('Impulse response')
plt.xlabel('time')
```

Output:





Czy wykresy się pokrywają?
Wykresy się pokrywają.

3. **Dokonać przekształceń pomiędzy transmitancją a zmiennymi stanu, wykorzystując polecenia *tf2ss(num, den)* oraz *ss2tf(A, B, C, D[, input])*.**

```
G2 = ss2tf(A, B, C, D)
sys2 = tf2ss(num, den)
print(f'G1 = {G1}\n')
print(f'G2 = {G2}\n')
print(f'sys1 = {sys1}\n')
print(f'sys2 = {sys2}\n')
```

Output:

```
G1 = TransferFunctionContinuous(
array([1., 0.]),
array([1.0e+00, 1.2e+01, 1.0e+04]),
dt: None
)

G2 = (array([[0.00000000e+00, 1.00000000e+00, 3.63797881e-12]]), array([1.0e+

sys1 = StateSpaceContinuous(
array([[ 0.0e+00,  1.0e+00],
       [-1.0e+04, -1.2e+01]]),
array([[0.],
       [1.]]),
array([[0, 1]]),
array([[0]]),
dt: None
)

sys2 = (array([[-1.2e+01, -1.0e+04],
       [ 1.0e+00,  0.0e+00]]), array([[1.],
       [0.]]), array([[1., 0.]]), array([[0.]]))
```

Czy wyprowadzone postaci modeli pokrywają się z tymi wyznaczonymi w Pythonie? Dlaczego?

Wyprowadzone postaci modeli pokrywają się z tymi wyznaczonymi w Pythonie.
Jednakże ostatni wspóczynnik w mianowniku G1 i G2 *jest różny, co jest spowodowane przybliżeniami liczb zmiennoprzecinkowych, a liczba jest bliska zeru.*
sys1 i sys2 *mają zamienione miejscami współczynniki jednakże w efekcie końcowym wyniki są równe.*
Modele po przekształceniach pokrywają się dlatego że są różnymi reprezentacjami tego samego układu.

4. **Zmienić wartość indukcyjności na $L = 0.15H$. Dokonać przekształceń pomiędzy transmi- tancją a zmiennymi stanu.**

```python
L = 0.15
den = [L, R, 1/Cap]
A = np.array([
    [0, 1],
    [-1/(L*Cap), -R/L]
])
B = np.array([[0], [1/L]])
G1 = signal.TransferFunction(num, den)
sys1 = signal.StateSpace(A, B, C, D)
G2 = ss2tf(A, B, C, D)
sys2 = tf2ss(num, den)
print(f'G1 = {G1}\n')
print(f'G2 = {G2}\n')
```

```python
print(f'sys1 = {sys1}\n')
print(f'sys2 = {sys2}\n')
```

Output:

```
G1 = TransferFunctionContinuous(
array([6.66666667, 0.        ]),
array([1.00000000e+00, 8.00000000e+01, 6.66666667e+04]),
dt: None
)

G2 = (array([[0.        , 6.66666667, 0.        ]]), array([1.00000000e+00, 8.

sys1 = StateSpaceContinuous(
array([[ 0.00000000e+00,  1.00000000e+00],
       [-6.66666667e+04, -8.00000000e+01]]),
array([[0.        ],
       [6.66666667]]),
array([[0, 1]]),
array([[0]]),
dt: None
)

sys2 = (array([[-8.00000000e+01, -6.66666667e+04],
       [ 1.00000000e+00,  0.00000000e+00]]), array([[1.],
       [0.]]), array([[6.66666667, 0.        ]]), array([[0.]]))
```

Czy wyprowadzone postaci modeli pokrywają się z tymi wyznaczonymi w Pythonie?
Dlaczego?
Wyprowadzone postaci modeli pokrywają się z tymi wyznaczonymi w Pythonie.
Modele po przekształceniach pokrywają się dlatego że są różnymi reprezentacjami tego samego układu.

3 Zadanie 4

1. **Zakładając następujące zmienne stanu $x_1 = \Theta$ i $x_2 = \dot{\Theta}$, wejście $u = \tau_m$ oraz wyjście $y = x_1$ wyznaczyć równania stanu dla obiektu.**

$$\ddot{\Theta} = \frac{1}{J}\tau_m - \frac{d}{J}\dot{\Theta}$$

$$J = \frac{1}{3}mL^2$$

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -\frac{d}{J}x_2 + \frac{1}{J}u \\ y = x_1 \end{cases}$$

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{d}{J} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{J} \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} u$$

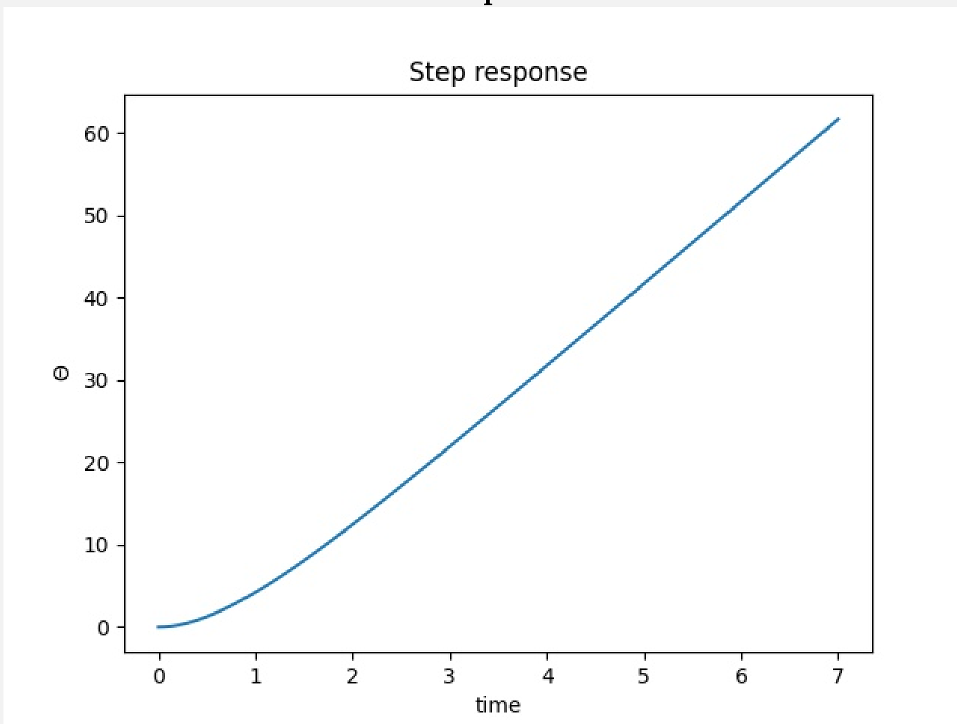
```
from scipy import signal
import matplotlib.pyplot as plt
import numpy as np

m = 1
L = 0.5
d = 0.1
J = m*L**2/3
A = np.array([
    [0, 1],
    [0, -d/J]
])
B = np.array([
    [0],
    [1/J]
])
C = np.array([
    [1, 0]
])
D = np.array([[0]])
sys = signal.StateSpace(A, B, C, D)
```

2. **Wyznaczyć odpowiedź skokową obiektu wykorzystując polecenie *signal.step(sys2)*.**

```
x, y = signal.step(sys)
plt.plot(x, y)
plt.title('Step response')
plt.xlabel('time')
plt.ylabel('$\Theta$')
```

Output:



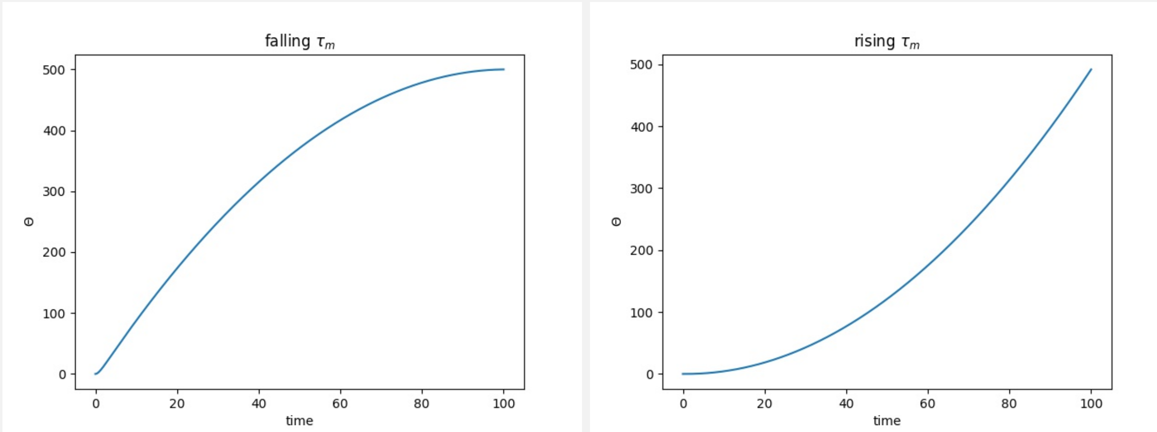
Jaki jest charakter odpowiedzi skokowej obiektu?
Odpowiedź skokowa ma charakter całkujący z inercją.

3. **Wyznaczyć odpowiedzi obiektu dla różnych sygnałów wejściowych (sygnał τ_m liniowo narastający dla wartości początkowej równej 0, sygnał τ_m liniowo odpadający dla wartości początkowej równej 1) wykorzystując polecenie *scipy.signal.lsim2*. Do poprawnego wykonania zadania konieczna wcześniejsza deklaracja wektorów czasu i zadanego sygnału wejściowego.**

```
t_max = 100
t = np.linspace(0, t_max, 1000)
u = t/t_max
x, y, _ = signal.lsim2(sys, u, t)
plt.plot(x, y)
plt.title('rising $\\tau_m$')
plt.xlabel('time')
plt.ylabel('$\Theta$')

x, y, _ = signal.lsim2(sys, u[::-1], t)
plt.plot(x, y)
plt.title('falling $\\tau_m$')
plt.xlabel('time')
plt.ylabel('$\Theta$')
```

Output:



Jaki jest charakter odpowiedzi obiektu?
Odpowiedź ma charakter całkujący z inercją.

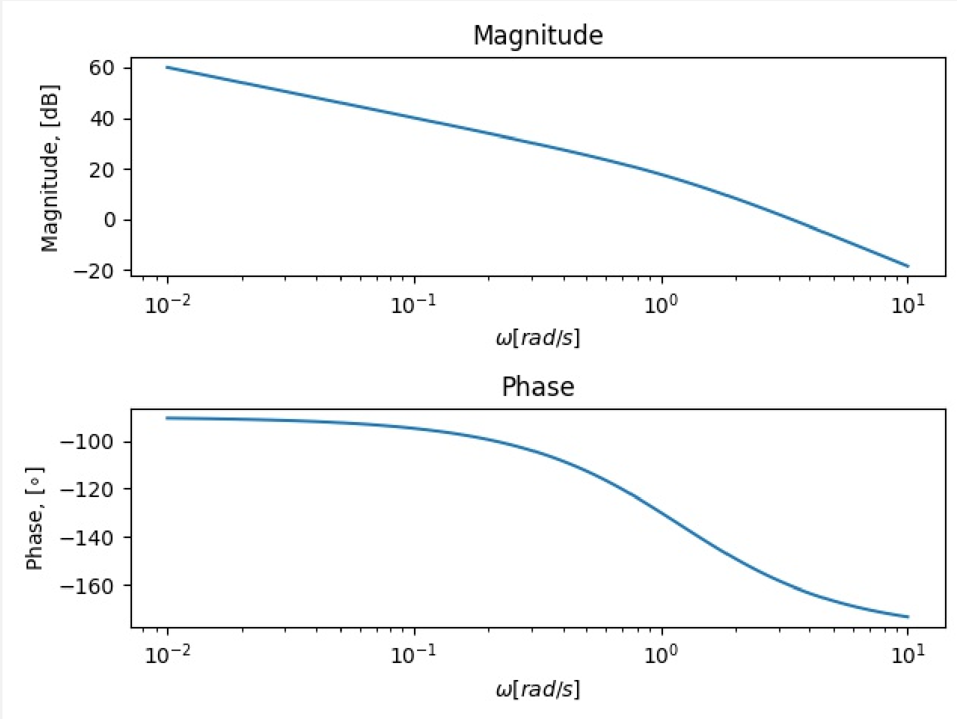
4. **Wyznaczyć charakterystykę Bodego dla obiektu wykorzystując polecenie *scipy.signal.bode*. Wykreślić charakterystykę w skali logarytmicznej wykorzystując polecenie *plt.semilogx***

```
w, mag, phase = signal.bode(sys)

plt.subplot(2, 1, 1)
plt.semilogx(w, mag)
plt.title('Magnitude')
plt.ylabel('Magnitude, [dB]')
plt.xlabel('$\omega[rad/s]$')

plt.subplot(2, 1, 2)
plt.semilogx(w, phase)
plt.title('Phase')
plt.ylabel('Phase, [$\circ$]')
plt.xlabel('$\omega[rad/s]$')
plt.tight_layout()
```

Output:



Czy wykresy Bodego odpowiadają typowi obiektu?
Wykresy odpowiadają typowi obiektu.