

# POLITECHNIKA POZNAŃSKA

Wydział Automatyki, Robotyki i Elektrotechniki



## Sprzężenie od stanu - stabilizacja w punkcie Teoria sterowania

Sprawozdanie z zajęć

Dawid Sobczak, 144627  
dawid.r.sobczak@student.put.poznan.pl  
Maja Zelmanowska, 144582  
maja.zelmanowska@student.put.poznan.pl

Prowadzący:  
dr inż. Paulina Superczyńska  
paulina.superczynska@put.poznan.pl

12-12-2021

## Spis treści

<b>1</b>	<b>Wstęp</b>	<b>2</b>
1.1	Badany układ . . . . .	2
<b>2</b>	<b>Przekształtnik elektryczny</b>	<b>2</b>
2.1	Zadanie 1.1 . . . . .	2
2.2	Zadanie 1.2 . . . . .	2
2.3	Zadanie 1.3 . . . . .	3
<b>3</b>	<b>Sprzężenie od stanu</b>	<b>3</b>
3.1	Zadanie 2.1 . . . . .	3
3.2	Zadanie 2.2 . . . . .	4
3.3	Zadanie 2.3 . . . . .	4
3.3.1	Symulacja 1: $\omega_c = 5$ . . . . .	4
3.3.2	Symulacja 2: $\omega_c = 10$ . . . . .	5
3.3.3	Symulacja 3: $\omega_c = 25$ . . . . .	6
3.3.4	Symulacja dodatkowa 1: $\omega_c = 0$ . . . . .	6
3.3.5	Symulacja dodatkowa 2: $\omega_c = -5$ . . . . .	7
3.3.6	Wnioski . . . . .	8
<b>4</b>	<b>Sprzężenie od stanu z obserwatorem Luenbergera</b>	<b>8</b>
4.1	Zadanie 3.1 . . . . .	8
4.2	Zadanie 3.2 . . . . .	9
4.3	Zadanie 3.3 . . . . .	9
4.4	Zadanie 3.4 . . . . .	11
<b>5</b>	<b>Wnioski</b>	<b>11</b>

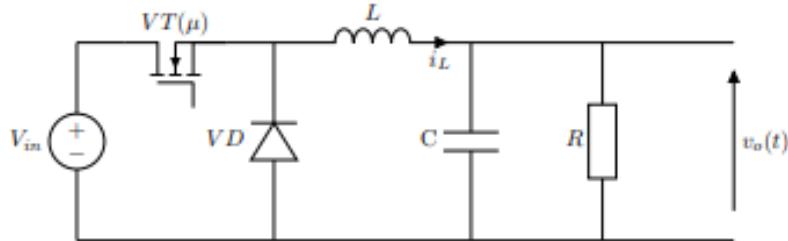
## Spis rysunków

1	Schemat ideowy badanego układu . . . . .	2
2	Wyliczone wzory na wzmocnienia $k_1$ oraz $k_2$ . . . . .	3
3	Zerowe warunki początkowe. . . . .	4
4	Niezerowe warunki początkowe. . . . .	5
5	Zerowe warunki początkowe. . . . .	5
6	Niezerowe warunki początkowe. . . . .	5
7	Zerowe warunki początkowe. . . . .	6
8	Niezerowe warunki początkowe. . . . .	6
9	Zerowe warunki początkowe. . . . .	6
10	Niezerowe warunki początkowe. . . . .	7
11	Zerowe warunki początkowe. . . . .	7
12	Niezerowe warunki początkowe. . . . .	7
13	Wyznaczenie wzorów na wzmocnienia obserwatora Luenbergera . . . . .	8
14	Zerowe warunki początkowe. . . . .	9
15	Niezerowe warunki początkowe. . . . .	10
16	. . . . .	11

# 1 Wstęp

## 1.1 Badany układ

ożwiera się układ elektrycznego przekształtnika obniżającego napięcie stałe przedstawiony podglądowno na ??



Rysunek 1: Schemat ideowy badanego układu

Układ opisany jest następującymi równaniami:

$$\begin{aligned} \frac{d}{dt}v_0(t) &= \frac{1}{CR}i_L(t) - \frac{1}{CR}v_0(t) \\ \frac{d}{dt}i_L(t) &= \frac{V_{LN}}{L}\mu(t) - \frac{1}{L}v_0(t) \end{aligned} \quad (1)$$

## 2 Przekształtnik elektryczny

### 2.1 Zadanie 1.1

Korzystając z układu równań (1) oraz przyjmując  $x = [x_1 \ x_2]^T = [v_0 \ \dot{v}_0]^T$ ,  $u = \mu$  oraz  $y = v_0$  uzyskujemy następującą postać równań stanu układu liniowego:

$$\begin{aligned} \dot{\underline{x}} &= \begin{bmatrix} 0 & 1 \\ -\frac{1}{CL} & -\frac{1}{CR} \end{bmatrix} \cdot \underline{x} + \begin{bmatrix} 0 \\ \frac{V_{LN}}{CL} \end{bmatrix} \cdot \mu \\ y &= [1 \ 0] \cdot \underline{x} \end{aligned} \quad (2)$$

### 2.2 Zadanie 1.2

Aby wyznaczyć wartości własne macierzy  $A$ , podstawiamy do układu parametry obiektu, uzyskaliśmy następujące wartości własne:

$$\begin{aligned} \lambda_1 &= -10 + 5\sqrt{6}i \\ \lambda_2 &= -10 - 5\sqrt{6}i \end{aligned} \quad (3)$$

Uzyskane wartości własne to **para pierwiastków zespolonych sprzężonych**, znajdująca się w lewej półpłaszczyźnie zespolonej.

- **Układ jest stabilny.**
- **Układ ma tendencje do oscylacji,** jest to układ stabilny, oscylacyjny.

## 2.3 Zadanie 1.3

Implementacja układu w środowisku Python prezentuje się następująco:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.integrate import solve_ivp
4
5 # params
6 L = 0.1      # [H]
7 Vin = 8       # [V]
8 R = 5        # [Omega]
9 C = 0.01     # [F]
10
11 # object
12 A = np.array([[0, 1],
13               [-1/(C*L), -1/(C*R)]])
14
15 B = np.array([[0],
16               [Vin/(C*L)]])
17
18 C = np.array([[1, 0]])
19
20 D = np.array([[0]])
21
22 def u(t):
23     return np.array([[mi]])
24
25
26 def model(t, x):
27     x = np.array([x]).T
28     dx = A @ x + B @ u(t);
29     return np.ndarray.tolist(dx.T[0])

```

## 3 Sprzężenie od stanu

### 3.1 Zadanie 2.1

(2.1) Wyznaczenie wartości zmiennych

$$H = A - BK \quad , \quad K = [k_1 \ k_2]$$

$$H = \begin{bmatrix} 0 & 1 \\ -1000 & -20 \end{bmatrix} - \begin{bmatrix} 0 \\ 8000 \end{bmatrix} \begin{bmatrix} k_1 & k_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1000 & -20 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 8000k_1 & 8000k_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1000-8000k_1 & -20-80k_2 \end{bmatrix}$$

$$(\lambda I - H) = \begin{bmatrix} \lambda & -1 \\ 1000+8000k_1 & \lambda+20+8000k_2 \end{bmatrix} \xrightarrow{\text{det}} \lambda(\lambda+20+8000k_2) + 1000 + 8000k_1 = 0$$

$$\lambda^2 + (20+8000k_2)\lambda + 1000 + 8000k_1 = 0$$

referencyjnie:  $\lambda = -\omega_C$

$$\varphi(\lambda) = (\lambda + \omega_C)^2 = 0 \Rightarrow \lambda^2 + 2\omega_C\lambda + \omega_C^2 = 0$$

$$2\omega_C = 20 + 8000k_2 \Rightarrow 8000k_2 = 2\omega_C - 20 \Rightarrow k_2 = \frac{2\omega_C - 20}{8000} = \frac{\omega_C - 10}{4000}$$

$$\omega_C^2 = 1000 + 8000k_1 \Rightarrow 8000k_1 = \omega_C^2 - 1000 \Rightarrow k_1 = \frac{\omega_C^2 - 1000}{8000}$$

Rysunek 2: Wyliczone wzory na wzmacnienia  $k_1$  oraz  $k_2$

- **Jak dobierać wartości  $\omega_c$ ?**

Wartość musi być większa od 0, aby sterownik mógł zadziałać.

- **Czy sterownik zadziała dla dowolnych wartości?**

Sterownik nie zadziała dla wartości mniejszych od 0, dodatkowo, zbyt duża wartość wpływa negatywnie na jakość regulacji.

### 3.2 Zadanie 2.2

Implementacja prawa sterowania w środowisku Python:

```

1  """
2  ZADANIE 2
3  """
4  # wzory do zadania 2.2/2.1
5  omega_c = 5    # omega z zakresu {5, 10, 25}
6  vd = 5        # wartość referencyjna
7  VD = np.array([[vd],
8                 [0]])
9
10 # wzmacnienia sterownika
11 k1 = ((omega_c**2)-1000)/8000
12 k2 = (2*omega_c-20)/8000
13
14 K = np.array([[k1, k2]])
15
16 def model2(t, x):
17     x = np.array([x]).T
18
19     dxe = A @ x + B * (vd/Vin - (-K @ (VD - x)))
20
21     return np.ndarray.tolist(dxe.T[0])

```

### 3.3 Zadanie 2.3

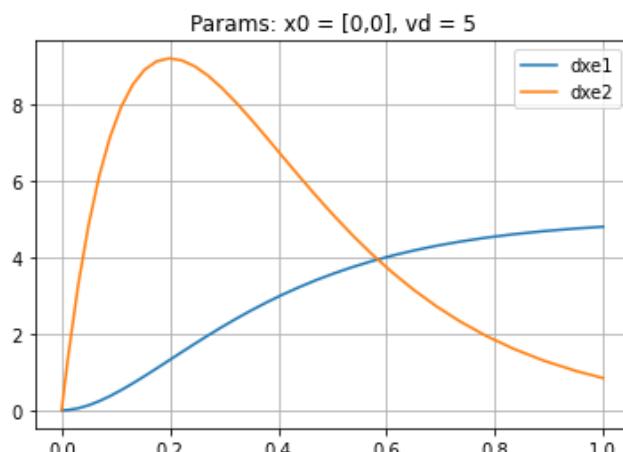
- **Symulacje przeprowadzone na układzie ze sprzężeniem od stanu.**

Badamy układ przy dwóch stanach:

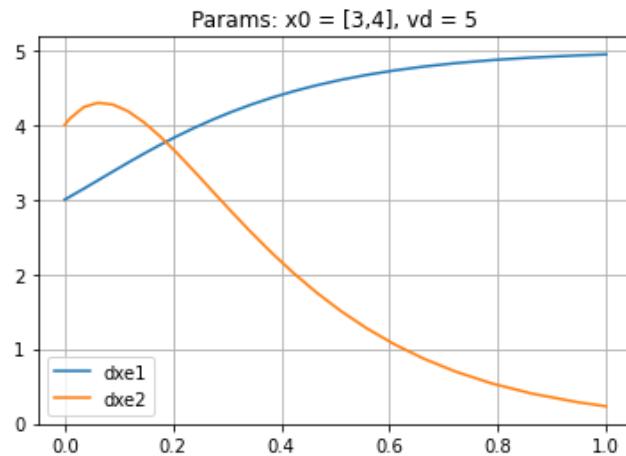
- zerowe warunki początkowe:  $\underline{x}(0) = [0 \ 0]^T$
- niezerowe warunki początkowe:  $\underline{x}(0) = [3 \ 4]^T$

Dla wartości  $\omega_c$  ze zbioru:  $\{5, 10, 25\}$ , oraz wartości referencyjnej  $v_d = 5$ .

#### 3.3.1 Symulacja 1: $\omega_c = 5$

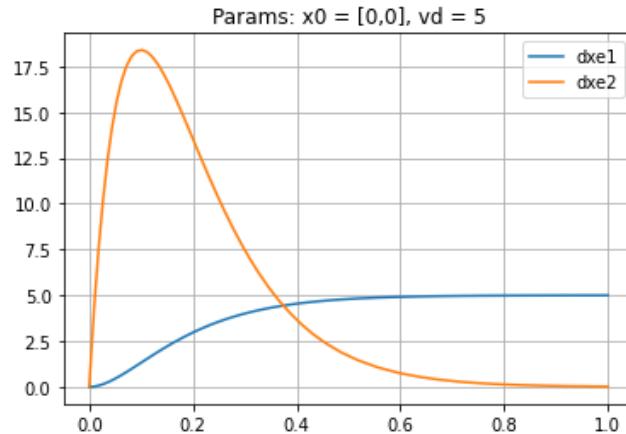


Rysunek 3: Zerowe warunki początkowe.

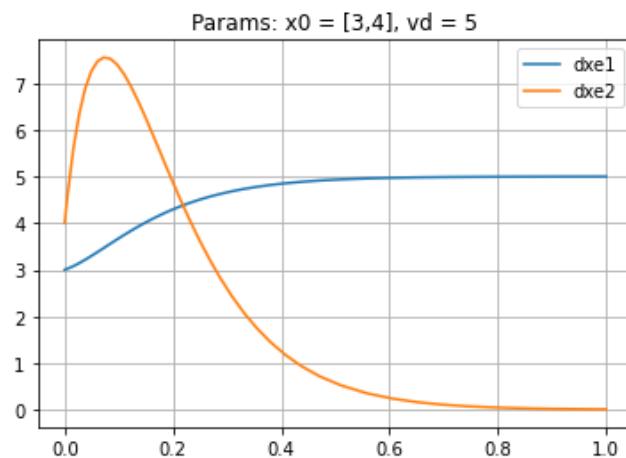


Rysunek 4: Niezerowe warunki początkowe.

### 3.3.2 Symulacja 2: $\omega_c = 10$

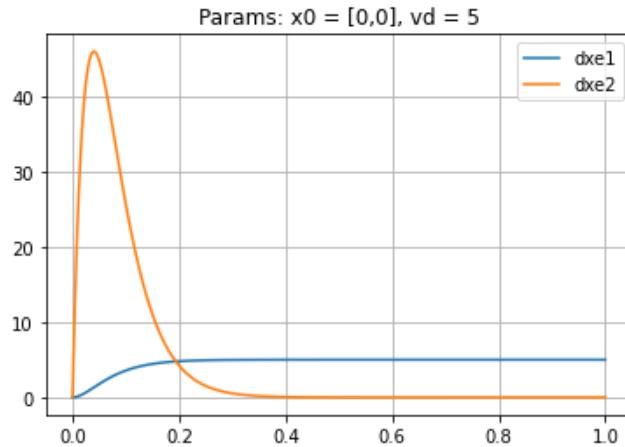


Rysunek 5: Zerowe warunki początkowe.

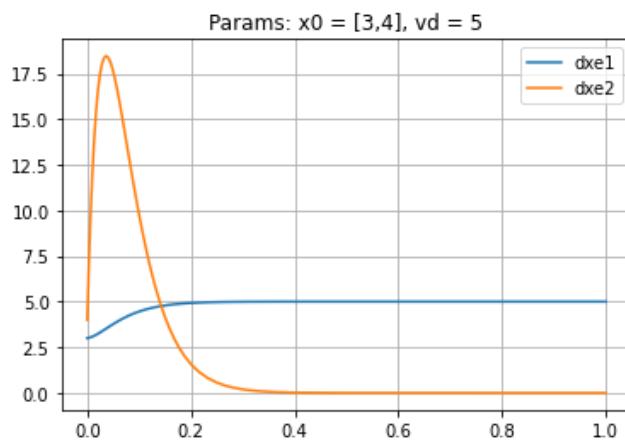


Rysunek 6: Niezerowe warunki początkowe.

### 3.3.3 Symulacja 3: $\omega_c = 25$

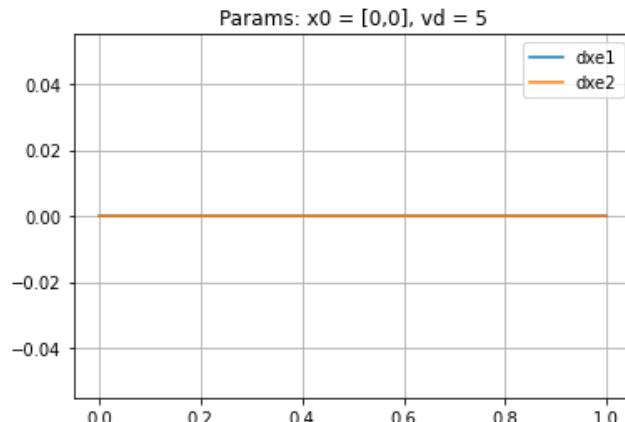


Rysunek 7: Zerowe warunki początkowe.

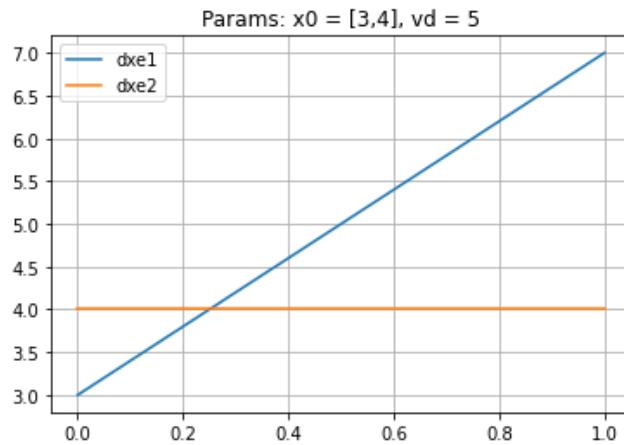


Rysunek 8: Niezerowe warunki początkowe.

### 3.3.4 Symulacja dodatkowa 1: $\omega_c = 0$

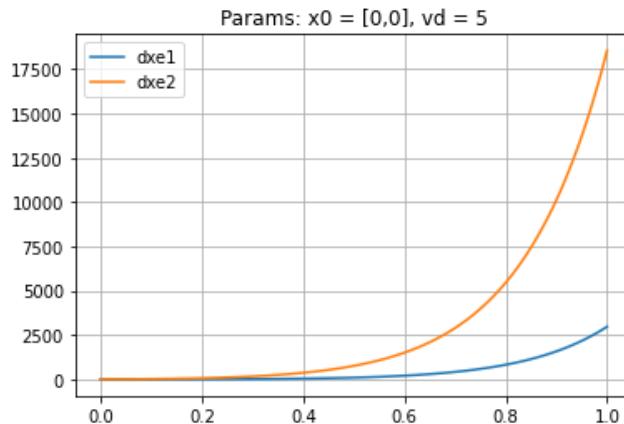


Rysunek 9: Zerowe warunki początkowe.

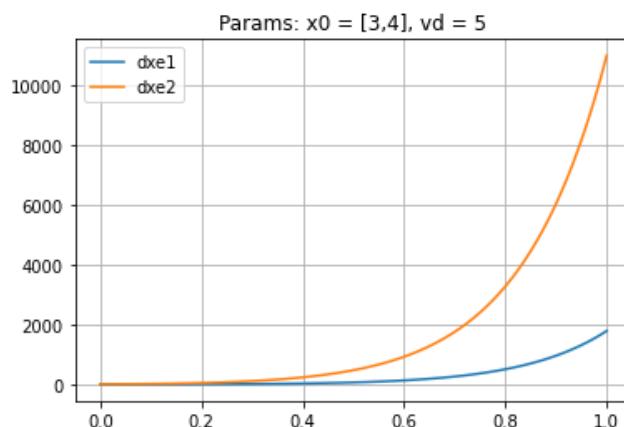


Rysunek 10: Niezerowe warunki początkowe.

### 3.3.5 Symulacja dodatkowa 2: $\omega_c = -5$



Rysunek 11: Zerowe warunki początkowe.



Rysunek 12: Niezerowe warunki początkowe.

### 3.3.6 Wnioski

Zwiększenie wartości  $\omega_c$  wpływa na szybkość regulacji na wartości zadanej  $v_d$  przez układ. Im większe  $\omega_c$ , tym szybciej układ się ustali, jednocześnie zmenna  $dxe2$  osiąga coraz większe przeregulowanie. Ważne jest, aby wartość  $\omega_c$  dobierać tak, aby uzyskać kompromis między szybkością układu, a nieznaczącą wartością przeregulowania. Niezerowe warunki początkowe również wpływają na wymienione wcześniej wartości. Przykład dla zerowej oraz ujemnej wartości  $\omega_c$  dowodzi, że układ nie zadziała dla dowolnych wartości.

## 4 Sprzężenie od stanu z obserwatorem Luenbergera

### 4.1 Zadanie 3.1

Wyznaczamy dynamikę błędu estymacji. Dobieramy wzmacnienia obserwatora tak, aby wartości własne macierzy stanu dynamiki błędu ( $H_L$ ) były równe:  $\lambda = -\omega_o$ .

$$\begin{aligned}
 (3.1) \quad \tilde{x} &= x - \hat{x} \\
 L &= \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} \\
 \dot{\tilde{x}} &= \underbrace{(A - LC)}_{H_L} \otimes \tilde{x}
 \end{aligned}$$

$$H_L = \begin{bmatrix} 0 & 1 \\ -1000 & -20 \end{bmatrix} - \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1000 & -20 \end{bmatrix} - \begin{bmatrix} l_1 & 0 \\ l_2 & 0 \end{bmatrix} = \begin{bmatrix} -l_1 & 1 \\ -1000 - l_2 & -20 \end{bmatrix}$$

$$\begin{aligned}
 (\lambda I - H_L) &= \begin{bmatrix} \lambda + l_1 & -1 \\ 1000 + l_2 & \lambda + 20 \end{bmatrix} \xrightarrow{\text{det}} \lambda^2 - 20\lambda - (\lambda + l_1)(\lambda + 20) + 1000 + l_2 = \\
 &= \lambda^2 + 20\lambda + l_1\lambda + 20l_1 + 1000 + l_2 = \\
 &= \lambda^2 + (20 + l_1)\lambda + 20l_1 + l_2 + 1000
 \end{aligned}$$

$$\varphi_2(\lambda) = (\lambda + \omega_o)^2 = \lambda + 2\omega_o\lambda + \omega_o^2$$

$$20 + l_1 = 2\omega_o \Rightarrow l_1 = 2\omega_o - 20 \quad \uparrow \quad 20(2\omega_o - 20) = 40\omega_o - 400$$

$$20l_1 + l_2 + 1000 = \omega_o^2 \Rightarrow l_2 = \omega_o^2 - 1000 - 20l_1 \Rightarrow$$

$$\Rightarrow l_2 = \omega_o^2 - 40\omega_o - 600$$

Rysunek 13: Wyznaczenie wzorów na wzmacnienia obserwatora Luenbergera

## 4.2 Zadanie 3.2

Implementacja obserwatora Luenbergera w środowisku Python:

```

1  """
2  ZADANIE 3
3  """
4
5  omega_o = 25
6  l1 = 2*omega_o - 20
7  l2 = omega_o**2 - 40*omega_o - 600
8
9  L = np.array([[l1],
10   [l2]])
11
12  def model3(t, x):
13      x = np.array([x]).T
14
15      x1 = np.array([x[0],
16                  x[1]])
17
18      x2 = np.array([x[2],
19                  x[3]])
20
21      dx = A @ x1 + B * force(t)
22
23      dx_est = A @ x1 + B * force(t) + L * (C @ x1 - C @ x2)
24
25      output = np.array([dx[0],
26                         dx[1],
27                         dx_est[0],
28                         dx_est[1]])
29
30  return np.ndarray.tolist(output.T[0])

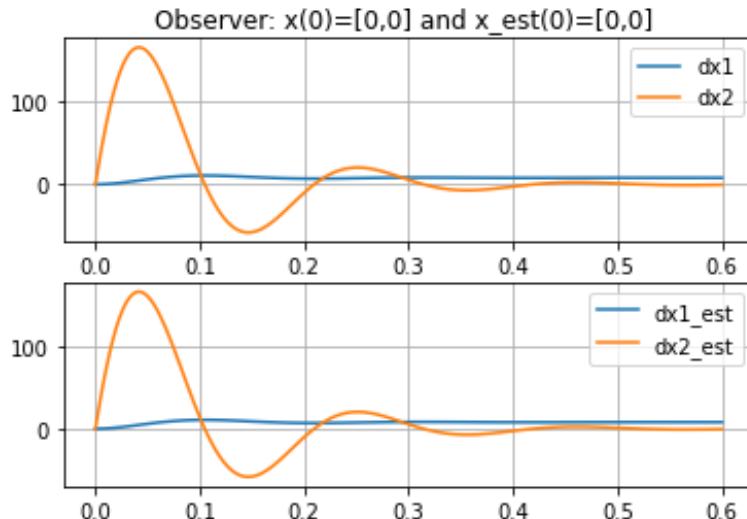
```

## 4.3 Zadanie 3.3

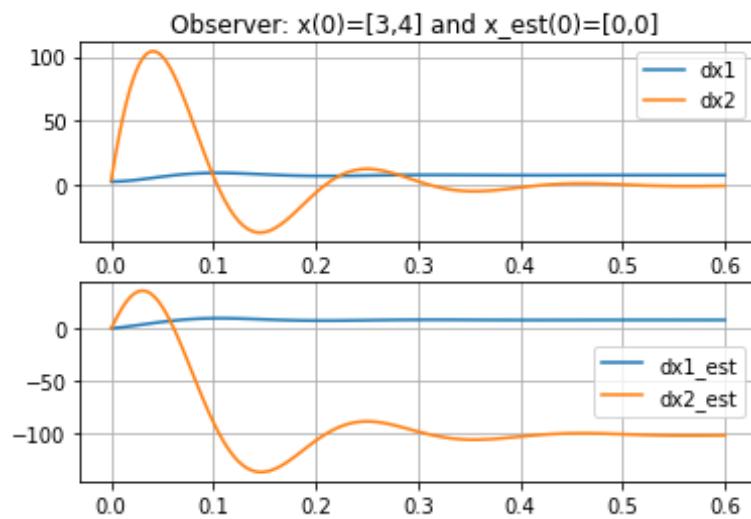
- **Symulacje przeprowadzone w układzie obserwatora Luenbergera:** Badamy układ przy dwóch stanach:

- zerowe warunki początkowe:  $\underline{x}(0) = [0 \ 0]^T$ ,  $\hat{x}(0) = [0 \ 0]^T$
- niezerowe warunki początkowe:  $\underline{x}(0) = [3 \ 4]^T$ ,  $\hat{x}(0) = [0 \ 0]^T$

Dla wartości  $\omega_c = 25$ , oraz wartości referencyjnej  $v_d = 5$ .



Rysunek 14: Zerowe warunki początkowe.



Rysunek 15: Niezerowe warunki początkowe.

#### 4.4 Zadanie 3.4

Wykorzystujemy estmaty stanu w sygnale sterującym.

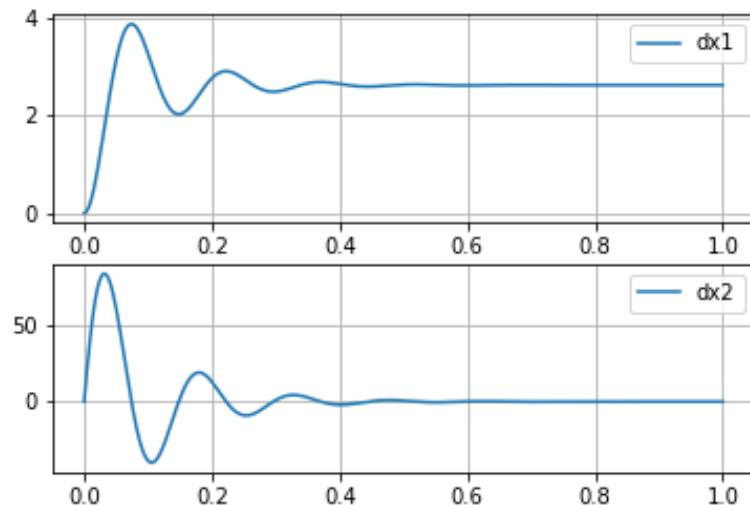
- **Implementacja:**

```

1  """
2  Zadanie 3.4
3  """
4  def model4(t, x):
5      x = np.array([x]).T
6
7      x1 = np.array([x[0],
8                     x[1]])
9
10     x2 = np.array([x[2],
11                    x[3]])
12
13
14     dx = A @ x1 + B * (vd/Vin - (-K @ x2))
15     dx_est = A @ x1 + B * (vd/Vin - (K @ x2)) + L @ (C @ x1 - C @ x2)
16
17     output = np.array([dx[0],
18                        dx[1],
19                        dx_est[0],
20                        dx_est[1]])
21
22     return np.ndarray.tolist(output.T[0])
23

```

- **Uzyskana odpowiedź:**



Rysunek 16:

### 5 Wnioski

Niezależnie od przyjmowanych warunków początkowych, estmaty zmiennych stanu są wyliczane prawidłowo. W przypadku zastosowania estmaty stanu w ciągu sygnału sterującego uzyskujemy szybciej wartość zadaną, jednakże układ traci na tym wpadając w oscylacje, które nie miały miejsca w przypadku zastosowania tylko sprzężenia od stanu. W przypadku różnych wartości początkowych dla zmiennych stanu i zmiennych estmowanych,  $dx_1$  jest estymowany prawidłowo, a dla  $dx_2$  pojawiają się błędne wartości spowodowane błędną estmacją.

## Literatura

[1] Kod programu do wykonanego laboratorium:

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Jan  4 23:31:34 2022
4
5  @author: Dawid Sobczak i Maja Zelmanowska
6  @note: LABORATORIUM 5 - SPRZĘT STANU/
7          STABILIZACJA W PUNKCIE
8
9
10 import numpy as np
11 import matplotlib.pyplot as plt
12 from scipy.integrate import solve_ivp
13 from scipy import signal
14
15 # params
16 L = 0.1      # [H]
17 Vin = 8       # [V]
18 R = 5         # [Omega]
19 C = 0.01     # [F]
20
21 #mi = np.linspace(0, 1, num=50)      # force value <0, 1>
22 mi = 1
23
24 # wzory do zadania 2.2/2.1
25 #omega_c = 10    # omega z zakresu {5, 10, 25}
26 #vd = 5        # wartość referencyjna
27
28 # wzmacniania sterownika
29 #k1 = (omega_c*omega_c)/(8000) - 400
30 #k2 = omega_c/4000 - 0.125
31
32 #K = np.array([[k1, k2]])
33
34
35 # object
36 A = np.array([[0, 1],
37               [-1/(C*L), -1/(C*R)]])
38
39 B = np.array([[0],
40               [Vin/(C*L)]])
41
42 C = np.array([[1, 0]])
43
44 D = np.array([[0]])
45
46 # wektor stanu referencyjnego
47 #xd = np.array([[vd],
48 #                [0]])
49
50
51 """
52 ZADANIE 1
53 """
54 def force(t):
55     return np.array([[mi]])
56
57
58 def model(t, x):
59     x = np.array([x]).T
60
61     dx = A @ x + B @ force(t);
62
63     return np.ndarray.tolist(dx.T[0])
64
65 G = signal.lti(A, B, C, D)
66
67 result_1 = solve_ivp(model, [0,1.0], [0,0], rtol=1e-10)
68
69 # step
70 plt.figure()
```

```
71     plt.plot(result_1.t, result_1.y[0])
72     plt.plot(result_1.t, result_1.y[1])
73     plt.grid()
74     plt.legend(['dx1', 'dx2'])
75     plt.title("Odpowied skokowa uk adu")
76
77 """
78 ZADANIE 2
79 """
80 # wzory do zadania 2.2/2.1
81 omega_c = 10 # omega z zakresu {5, 10, 25}
82 vd = 5 # warto referencyjna
83 VD = np.array([[vd],
84                [0]])
85
86 # wzmocnienia sterownika
87 k1 = ((omega_c**2)-1000)/8000
88 k2 = (2*omega_c-20)/8000
89
90 K = np.array([[k1, k2]])
91
92 def model2(t, x):
93     x = np.array([x]).T
94
95     dxe = A @ x + B * (vd/Vin - (-K @ (VD - x)))
96
97     return np.ndarray.tolist(dxe.T[0])
98
99 # initial values = [0,0]
100 result_2 = solve_ivp(model2, [0, 1.0], [0, 0], rtol=1e-10)
101
102 plt.figure()
103 plt.plot(result_2.t, result_2.y[0])
104 plt.plot(result_2.t, result_2.y[1])
105 plt.grid()
106 plt.legend(['dxe1', 'dxe2'])
107 plt.title("Params: x0 = [0,0], vd = 5")
108
109 # initial values = [3,4]
110 result_2 = solve_ivp(model2, [0, 1.0], [3, 4], rtol=1e-10)
111
112 plt.figure()
113 plt.plot(result_2.t, result_2.y[0])
114 plt.plot(result_2.t, result_2.y[1])
115 plt.grid()
116 plt.legend(['dxe1', 'dxe2'])
117 plt.title("Params: x0 = [3,4], vd = 5")
118
119 """
120 ZADANIE 3
121 """
122 omega_o = 25
123 l1 = 2*omega_o - 20
124 l2 = omega_o**2 - 40*omega_o - 600
125
126 L = np.array([[l1],
127               [l2]])
128
129 def model3(t, x):
130     x = np.array([x]).T
131
132     x1 = np.array([x[0],
133                   x[1]])
134
135     x2 = np.array([x[2],
136                   x[3]])
137
138     dx = A @ x1 + B * force(t)
139
140     dx_est = A @ x1 + B * force(t) + L * (C @ x1 - C @ x2)
141
142     output = np.array([dx[0],
143                       dx[1],
```

```
144         dx_est[0] ,
145         dx_est[1]))
146
147     return np.ndarray.tolist(output.T[0])
148
149 res = solve_ivp(model3, [0,0.6], [0,0,0,0], rtol=1e-10)
150 plt.figure()
151 plt.subplot(2,1,1)
152 plt.title("Observer: x0=[0,0], x_est0=[0,0] ")
153 plt.plot(res.t, res.y[0])
154 plt.plot(res.t, res.y[1])
155 plt.legend(['dx1', 'dx2'])
156 plt.grid()
157 plt.subplot(2,1,2)
158 plt.plot(res.t, res.y[2])
159 plt.plot(res.t, res.y[3])
160 plt.legend(['dx1_est', 'dx2_est'])
161 plt.grid()
162
163 plt.figure()
164 res = solve_ivp(model3, [0,0.6], [3,4,0,0], rtol=1e-10)
165 plt.subplot(2,1,1)
166 plt.title("Observer: x0=[3,4], x_est0=[0,0] ")
167 plt.plot(res.t, res.y[0])
168 plt.plot(res.t, res.y[1])
169 plt.grid()
170 plt.legend(['dx1', 'dx2'])
171 plt.subplot(2,1,2)
172 plt.plot(res.t, res.y[2])
173 plt.plot(res.t, res.y[3])
174 plt.legend(['dx1_est', 'dx2_est'])
175 plt.grid()
176
177 """
178 Zadanie 3.4
179 """
180 def model4(t, x):
181     x = np.array([x]).T
182
183     x1 = np.array([x[0],
184                   x[1]])
185
186     x2 = np.array([x[2],
187                   x[3]])
188
189
190     dx = A @ x1 + B * (vd/Vin - (-K @ x2))
191     dx_est = A @ x1 + B * (vd/Vin - (K @ x2)) + L @ (C @ x1 - C @ x2)
192
193     output = np.array([dx[0],
194                       dx[1],
195                       dx_est[0],
196                       dx_est[1]])
197
198     return np.ndarray.tolist(output.T[0])
199
200 res = solve_ivp(model4, [0, 1.0], [0,0,0,0], rtol=10e-10)
201 plt.figure()
202 plt.subplot(2,1,1)
203 plt.plot(res.t, res.y[0])
204 plt.legend(['dx1'])
205 plt.grid()
206 plt.subplot(2,1,2)
207 plt.plot(res.t, res.y[1])
208 plt.legend(['dx2'])
209 plt.grid()
```