

Podstawy Sterowania Optymalnego

Labolatorium 7

Sterowanie układami nieliniowymi przy pomocy metody SDRE

Prowadzący: mgr inż. Krzysztof Hałas

Wykonał: Ryszard Napierała

8 stycznia 2022

Zadanie 4

1. Przygotować funkcję *riccati(p,t)* implementującą różniczkowe równanie Riccatiego. Zdefiniować wektor chwil czasu od t_1 do 0 przyjmując $t_1 = 5s$ Wykorzystując funkcję *odeint* wyznaczyć przebieg wartości macierzy P w czasie. Zwrócić uwagę na konieczność konwersji macierzy P do postaci wektorowej dla uzyskania zgodności z funkcją *odeint*. Wykorzystać na przykład *np.reshape*, *squeeze* oraz *np.tolist*.

```
9  RESOLUTION = 300
10 L = 1 # R
11 m = 9
12 J = 1
13 g = 10
14 d = 0.5
15
16 def A_of_x(x: np.ndarray) -> np.ndarray:
17     if x[0, 0] == 0:
18         return np.array([
19             [0, 1],
20             [0, -d/J]
21         ]).astype(float)
22     return np.array([
23         [0, 1],
24         [-m*g*L*np.sin(x[0, 0])/(J*x[0, 0]), -d/J]
25     ]).astype(float)
26
27 B = np.array([[0, 1/J]]).T
28 R = np.array([[0.01]])
29 Q = np.array([
30     [1, 0],
31     [0, 1]
32 ])
```

2. Wykreślić przebieg elementów macierzy $P(t)$ w czasie. Zweryfikować poprawność wyników poprzez porównanie z warunkiem krańcowym.

```
35 def riccati_finite_diff(
36     x: np.ndarray,
37     t: float,
38     a: Callable,
39     B: np.ndarray,
40     Q: np.ndarray,
41     R: np.ndarray) -> np.ndarray:
42     P = x[:4].reshape((2, 2))
43     x = x[4:].reshape((2, 1))
44     A = a(x)
45     dP = -P@A - P@B@inv(R)@B.T@P - A.T@P + Q
46     p = P@x
47     u = -inv(R)@B.T@p
48     dx = A@x + B@u
49     return np.concatenate((dP.flatten(), dx.flatten()))
50
51 def riccati_infinite_diff(
52     x: np.ndarray,
53     t: float,
54     a: Callable,
55     B: np.ndarray,
56     Q: np.ndarray,
57     R: np.ndarray) -> np.ndarray:
58     x = x[4:].reshape((2, 1))
59     A = a(x)
60     P = solve_continuous_are(A, B, Q, R)
61     u = -inv(R)@B.T@P@x
62     dx = A@x + B@u
63     return np.concatenate((P.flatten(), dx.flatten()))
64
65 def show_P(t: np.ndarray, res: np.ndarray, title: str) -> None:
66     plt.plot(t, res[:, :4])
67     plt.title(title)
68     plt.xlim(t[0], t[-1])
69     plt.legend(['$p_{11}$', '$p_{12}$', '$p_{21}$', '$p_{22}$'])
70
71 def concatenate_plots(
72     plot_func: Callable,
73     plots: List[Tuple[np.ndarray, np.ndarray, str]],
74     supitle: str,
75     shape: Tuple[int, int],
76     size: Tuple[int, int]=(8,10)) -> None:
77     plt.figure(figsize=size)
78     plt.suptitle(supitle)
79     for i, data in enumerate(plots):
80         plt.subplot(*shape, i+1)
```

```

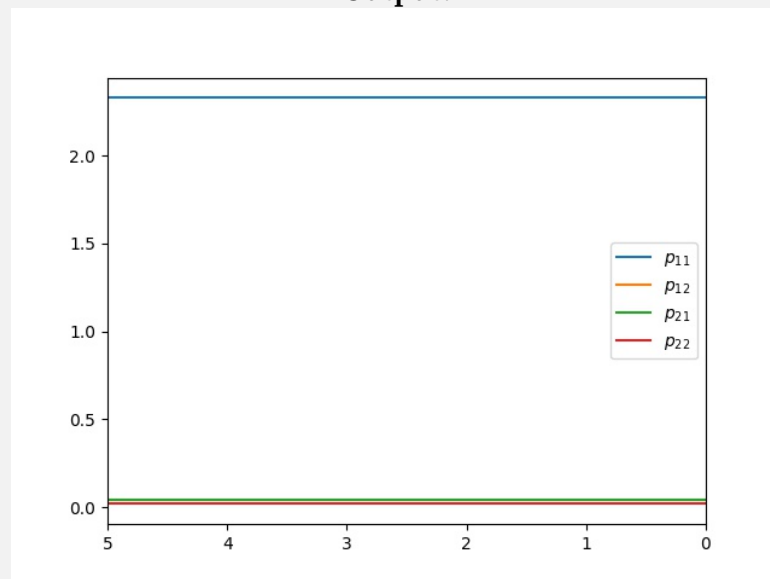
81     plot_func(*data)
82     plt.tight_layout()
83     plt.show()
84     plt.close()
85
86     x0 = [0]*2
87     P0 = [0]*4
88     t = np.linspace(5, 0, RESOLUTION)
89
90     res_fin = odeint(
91         riccati_finite_diff,
92         P0 + x0,
93         t,
94         (A_of_x, B, Q, R))
95
96     res_inf = odeint(
97         riccati_infinite_diff,
98         P0 + x0,
99         t,
100        (A_of_x, B, Q, R))

```

Output:

$(S \Rightarrow P) = \text{True}$

Output:



3. Przygotować funkcję $model(x, t)$ implementującą model dynamiki układu otwartego zgodnie z równaniem. Funkcja powinna przyjmować na wejściu stan układu x oraz aktualną chwilę czasu t .

```

103 # Macierz Q określa funkcję kosztu dla zadanego uchybu regulacji
104 # Macierz R określa funkcję kosztu dla zadanego sterowania
105
106 # 4.3
107 concatenate_plots(
108     show_P,
109     [
110         (t, res_fin, 'Finite Riccati'),
111         (t, res_inf, 'Infinite Riccati')
112     ],
113     '$P$ components vs time',
114     (2, 1)
115 )
116
117 S = solve_continuous_are(
118     A_of_x(
119         np.array([0,0]).reshape((2, 1))
120     ), B, Q, R
121 )

```

4. Zmodyfikować funkcję $model(x,t)$ tak, by wprowadzić do niej wyznaczone wcześniej wartości macierzy $P(t)$. Wykorzystać `interpolate.interp1d` w celu określenia wartości macierzy $P(t)$ w wybranej chwili czasu.

```

124 print('P_fin =\n', res_fin[-1, :4].reshape((2, 2)))
125 print('P_inf =\n', res_inf[-1, :4].reshape((2, 2)))
126 # P dla nieskończonego horyzontu jest nieskończone
127 # P dla skończonego horyzontu dąży do -S, nie udało mi się dojść do tego
128     ↪ dlaczego
129
130 # 4.4
131 def show_x(t: np.ndarray, res: np.ndarray, title:str) -> None:
132     plt.plot(t, res[:, 4:6])
133     plt.title(title)
134     plt.xlim(t[0], t[-1])
135     plt.legend(['$x_1$', '$x_2$'])
136
137 t = np.linspace(0, 5, RESOLUTION)
138
139 res_fin = odeint(
140     riccati_finite_diff,
141     P0 + x0,
142     t,
143     (A_of_x, B, Q, R))
144
145 res_inf = odeint(
146     riccati_infinite_diff,

```

```

146     P0 + x0,
147     t,
148     (A_of_x, B, Q, R))
149
150 concatenate_plots(
151     show_x,
152     [

```

5. Przeprowadzić symulację odpowiedzi obiektu na wymuszenie skokowe w czasie $t/4 \in (0, 5)s$ wykorzystując funkcję *odeint*.

```

155     ],
156     '$x$ components vs time',
157     (2, 1)
158 )
159
160 # 4.5
161 x0 = [np.pi/2, 0]
162 QR = [ # q11, q22, r11
163     (1, 0.01, 0.01),
164     (0.01, 1, 0.01),
165     (5, 5, 0.01),
166     (10, 1, 0.01),
167     (1, 10, 0.01),
168     (1, 1, 0.01),
169     (1, 1, 0.1),
170     (1, 1, 1),
171     (1, 1, 10),
172 ]
173 def experiment(
174     riccati_func: Callable,
175     QR: List[Tuple[float, float, float]],
176     x0: List[float],
177     t: np.ndarray,
178     suptitle:str) -> np.ndarray:
179     res = []
180     titles = []
181
182     for q11, q22, r11 in QR:
183         Q = np.array([[q11, 0], [0, q22]])
184         R = np.array([[r11]])
185         res.append(odeint(
186             riccati_func,
187             P0 + x0,
188             t,
189             (A_of_x, B, Q, R)))
190         titles.append(f'$q11={q11}, q22={q22}, r11={r11}$')

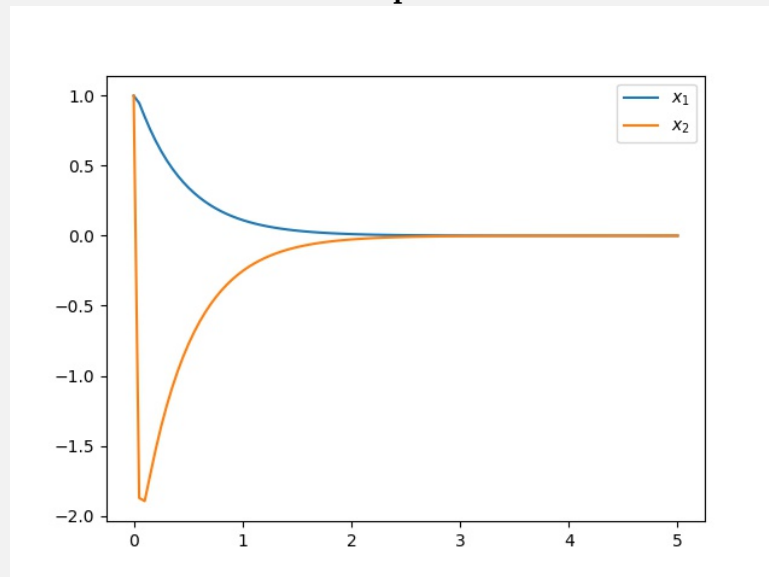
```

```

191
192 concatenate_plots(
193     show_x,
194     [(t, res[i], titles[i]) for i in range(len(res))],
195     supitle,
196     (3, 3),
197     (10, 10)
198 )
199
200 experiment(
201     riccati_finite_diff, QR, x0, t,
202     '$x$ components vs time with different $Q$ and $R$\n\
203     (finite riccati)'
204 )

```

Output:



6. Przeprowadzić symulację układu dla niezerowych warunków początkowych. Zbadać wpływ macierzy S , Q oraz R na przebieg odpowiedzi układu.

```

207 riccati_infinite_diff, QR, x0, t,
208 '$x$ components vs time with different $Q$ and $R$\n\
209     (infinite riccati)'
210 )
211
212 # Macierze Q i R pozwalają dowolnie kształtować przebieg uchybu regulacji
213 # Macierz Q określa funkcję kosztu dla zadanego uchybu regulacji
214 # Macierz R określa funkcję kosztu dla zadanego sterowania
215
216 # 4.6
217 def riccati_finite_diff_with_J(

```

```

218     x: np.ndarray,
219     t: float,
220     a: Callable,
221     B: np.ndarray,
222     Q: np.ndarray,
223     R: np.ndarray) -> np.ndarray:
224     P = x[:4].reshape((2, 2))
225     J = x[-1:].reshape((1, 1))
226     previous_t = x[-2]
227     x = x[4:6].reshape((2, 1))
228     A = a(x)
229     dP = -P@A - P@B@inv(R)@B.T@P - A.T@P + Q
230     p = P@x
231     u = -inv(R)@B.T@p
232     dx = A@x + B@u
233     J += (x.T@Q@x + u.T@R@u)*(t - previous_t)
234     return np.concatenate(
235         (dP.flatten(), dx.flatten(), [t], J.flatten())
236     )
237
238 def riccati_infinite_diff_with_J(
239     x: np.ndarray,
240     t: float,
241     a: Callable,
242     B: np.ndarray,
243     Q: np.ndarray,
244     R: np.ndarray) -> np.ndarray:
245     J = x[-1:].reshape((1, 1))
246     previous_t = x[-2]
247     x = x[4:6].reshape((2, 1))
248     A = a(x)
249     P = solve_continuous_are(A, B, Q, R)
250     u = -inv(R)@B.T@P@x
251     dx = A@x + B@u
252     J += (x.T@Q@x + u.T@R@u)*(t - previous_t)
253     return np.concatenate(
254         (P.flatten(), dx.flatten(), [t], J.flatten())
255     )
256
257 def show_J(t: np.ndarray, res: np.ndarray, title:str) -> None:
258     plt.plot(t, res[:, -1])
259     plt.title(title)
260     plt.xlim(t[0], t[-1])
261     plt.legend(['$J$'])
262
263 J0 = [0]
264 t0 = [0]
265
266 res_fin = odeint(

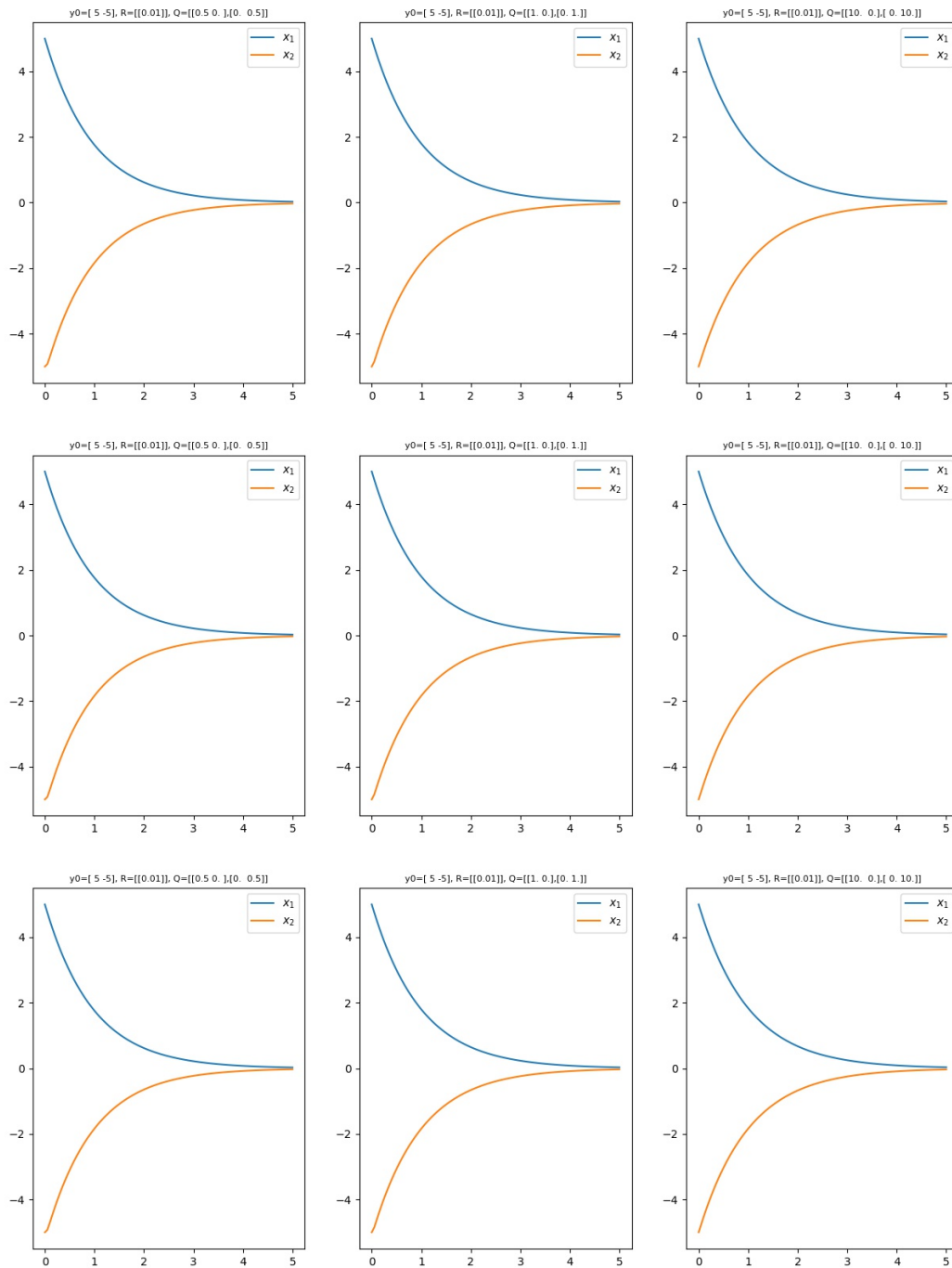
```

```

267     riccati_finite_diff_with_J,
268     P0 + x0 + t0 + J0,
269     t,
270     (A_of_x, B, Q, R))
271
272 res_inf = odeint(
273     riccati_infinite_diff_with_J,
274     P0 + x0 + t0 + J0,
275     t,
276     (A_of_x, B, Q, R))

```


Output:



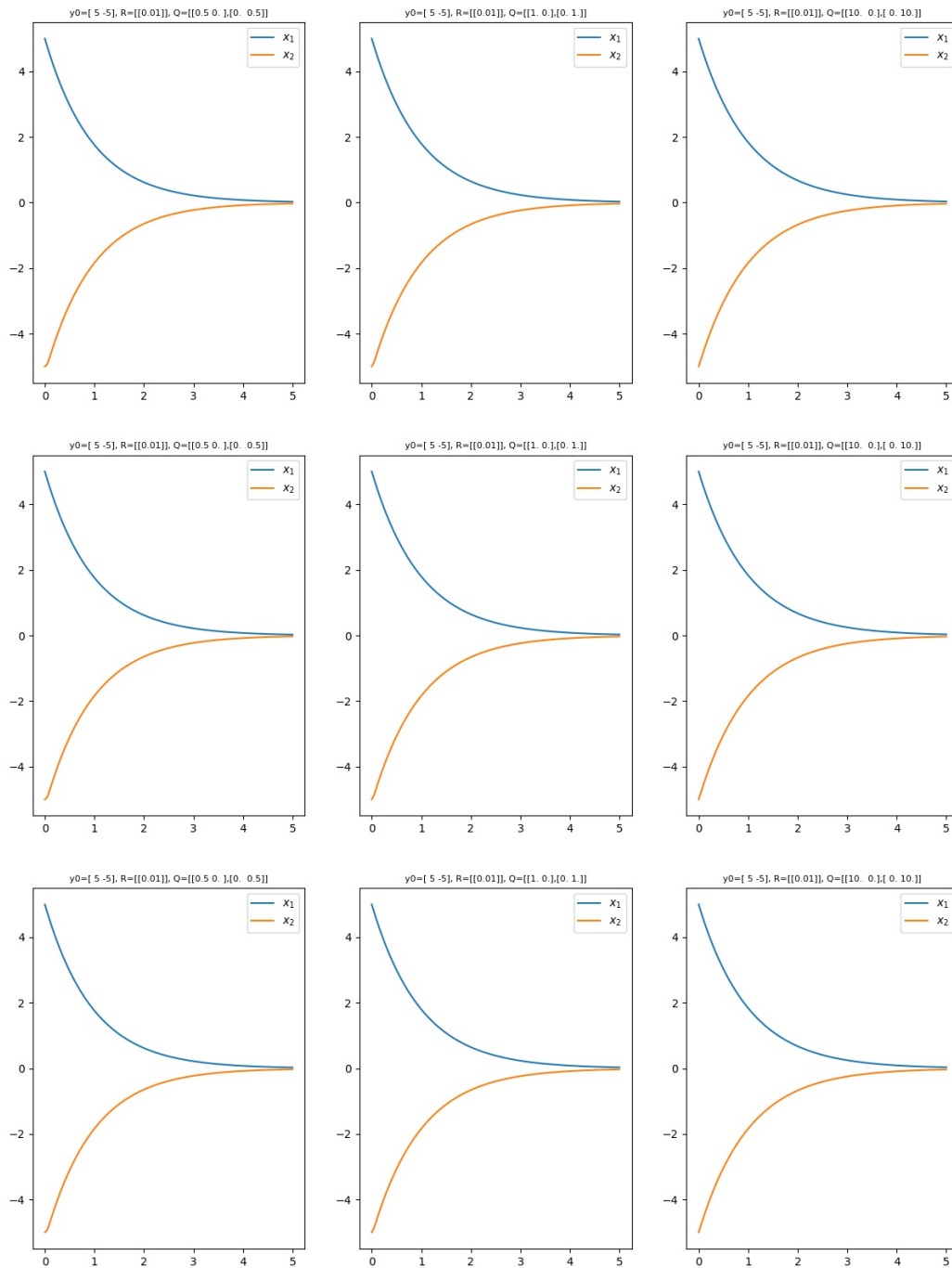
Czy macierze S , Q oraz R pozwalają dowolnie kształtować przebieg uchybu regulacji? Czy istnieje jakaś zależność między doбором tych macierzy?

Macierze Q, R, S nie pozwalają dowolnie kształtować przebiegu uchybu regulacji. Macierz S jest zależna od macieży Q oraz S .

7. Przeprowadzić symulację układu dla niezerowych warunków początkowych. Zbadać wpływ macierzy S, Q oraz R na przebieg odpowiedzi układu.

```
279     show_J,  
280     [  
281         (t, res_fin, 'Finite Riccati'),  
282         (t, res_inf, 'Infinite Riccati')  
283     ],  
284     '$J$ function vs time',  
285     (2, 1)  
286 )  
287 # Wyznaczona wartość J odpowiada minimalnemu koszcie regulacji  
288 # Została wyznaczona w czasie t \in [0, 5]  
289  
290 # 4.7  
291 t = np.linspace(0, 2, RESOLUTION)
```

Output:



Czy macierze S , Q oraz R pozwalają dowolnie kształtować przebieg uchybu regulacji? Czy istnieje jakaś zależność między doбором tych macierzy?

Macierze Q, R, S nie pozwalają dowolnie kształtować przebiegu uchybu regulacji. Macierz S jest zależna od macieży Q oraz S .