

Podstawy Sterowania Optymalnego

Labolatorium 7

Sterowanie układami nieliniowymi przy pomocy metody SDRE

Prowadzący: mgr inż. Krzysztof Hałas

Wykonał: Ryszard Napierała

7 stycznia 2022

Imports

```
1 from typing import Callable
2 import numpy as np
3 from numpy.linalg import inv, eig
4 from scipy import integrate, interpolate
5 import matplotlib.pyplot as plt
6 import sympy as sym
7 from dataclasses import dataclass
```

Zadanie 2

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*.

```
11 def riccati(
12     p: np.ndarray,
13     t: float,
14     A: np.ndarray,
15     B: np.ndarray,
16     Q: np.ndarray,
17     R: np.ndarray) -> np.ndarray:
18     p = p.reshape((2, 2))
19     dp = p@A - p@(B@inv(R)@B.T)@p + A.T@p + Q
20     return dp.flatten()
21
22 R = 0.5
```

```

23 C = 0.5
24 L = 0.2
25
26 A = np.array([
27     [0, 1],
28     [-1/(L*C), -R/L]
29 ])
30 B = np.array([[0, 1/L]]).T
31
32 Q = np.array([
33     [5, 0],
34     [0, 1]
35 ])
36 R = np.array([[0.01]])
37
38 def solve_for_S(
39     a: np.ndarray,
40     b: np.ndarray,
41     q: np.ndarray,
42     r: np.ndarray) -> np.ndarray:
43
44     s11, s12, s22 = sym.symbols('s11 s12 s22')
45     A = sym.Matrix(a)
46     B = sym.Matrix(b)
47     Q = sym.Matrix(q)
48     R = sym.Matrix(r)
49     S = sym.Matrix([
50         [s11, s12],
51         [s12, s22]
52     ])
53     res = sym.solve(
54         A.T*S+S*A-S*B*(R**(-1))*B.T*S+Q,
55         (s11, s12, s22)
56     )
57
58     for s11, s12, s22 in res:
59         S = np.array([
60             [s11, s12],
61             [s12, s22],
62         ]).astype(float)
63         K = inv(r)@b.T@S
64         e1, e2 = eig(a - b@K)[0]
65         if e1 < 0 and e2 < 0:
66             return S
67
68
69 S = solve_for_S(A, B, Q, R)
70
71 t = np.linspace(5, 0, 100)

```

```

72
73 res = integrate.odeint(
74     riccati,
75     S.flatten(),
76     t,
77     (A, B, Q, R))

```

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

```

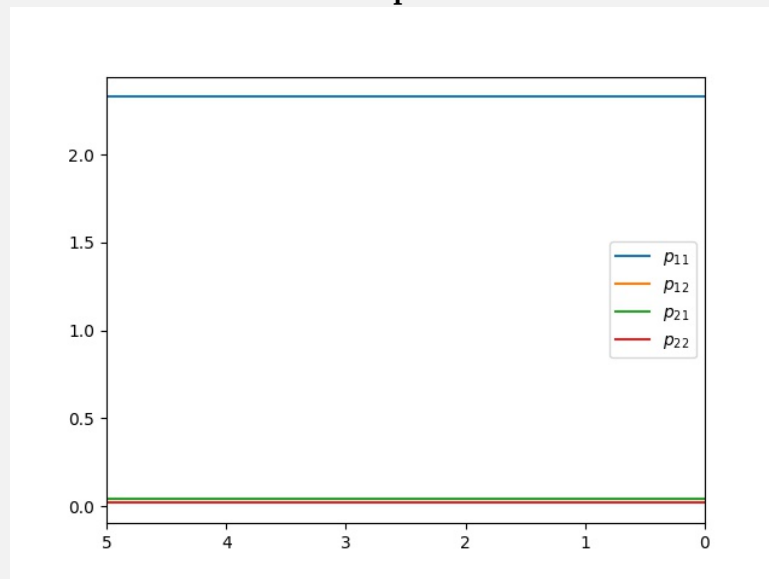
80 plt.plot(t, res)
81 plt.xlim(5, 0)
82 plt.legend(['$p_{11}$', '$p_{12}$', '$p_{21}$', '$p_{22}$'])
83 plt.show()
84 plt.close()
85 print(
86     '(S==P) = ',
87     np.all(np.isclose(S, res[-1].reshape((2,2))))
88 )

```

Output:

(S==P) = 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 .

```

91 def model(
92     x: np.ndarray,
93     t: float,
94     A: np.ndarray,
95     B: np.ndarray,
96     u: Callable) -> np.ndarray:
97     x = x.reshape((2, 1))
98     dx = A@x + B@u(t)
99     return dx.flatten()

```

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.

```

102 def model(
103     x: np.ndarray,
104     t: float,
105     A: np.ndarray,
106     B: np.ndarray,
107     R: np.ndarray,
108     p: Callable) -> np.ndarray:
109     x = x.reshape((2, 1))
110     k = inv(R)@B.T@p(t)
111     u = -k@x
112     dx = A@x + B@u
113     return dx.flatten()
114
115 i1d = interpolate.interp1d(
116     t,
117     res.T,
118     fill_value=res.T[:, -1],
119     bounds_error=False
120 )
121 p = lambda x: i1d(x).reshape((2, 2))

```

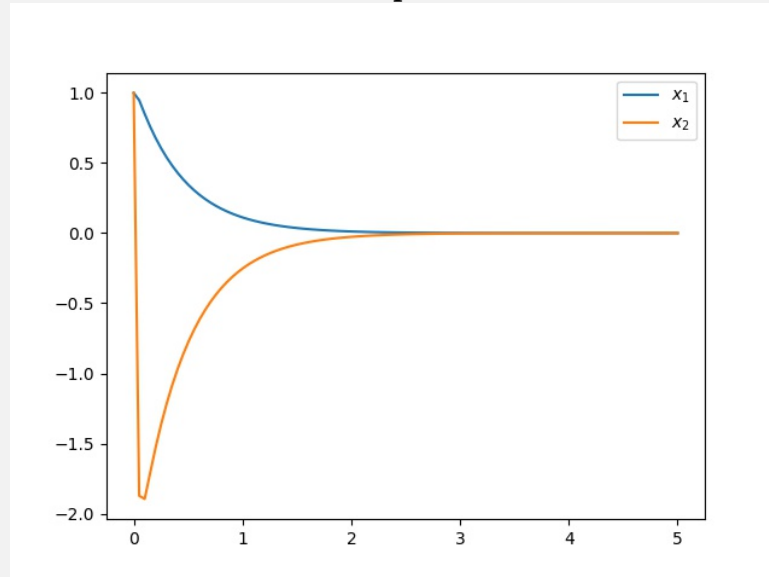
5. Przeprowadzić symulację odpowiedzi obiektu na wymuszenie skokowe w czasie $t/4 \in (0,5)s$ wykorzystując funkcję `odeint`.

```

124 t = t[:-1]
125 res = integrate.odeint(model, [1, 1], t, (A, B, R, p))
126
127 plt.plot(t, res)
128 plt.legend(['$x_1$', '$x_2$'])
129 plt.show()
130 plt.close()

```

Output:



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

```
133 @dataclass
134 class Experiment:
135     A: np.ndarray
136     B: np.ndarray
137     Q: np.ndarray
138     R: np.ndarray
139     t: np.ndarray
140
141     def __post_init__(self):
142         S = solve_for_S(
143             self.A, self.B, self.Q, self.R
144         )
145         res = integrate.odeint(
146             riccati,
147             S.flatten(),
148             self.t[::-1],
149             (self.A, self.B, self.Q, self.R)
150         )
151         i1d = interpolate.interp1d(
152             self.t,
153             res.T,
154             fill_value=res.T[:, -1],
155             bounds_error=False
156         )
157         self.p = lambda x: i1d(x).reshape((2, 2))
158
159     def model(
```

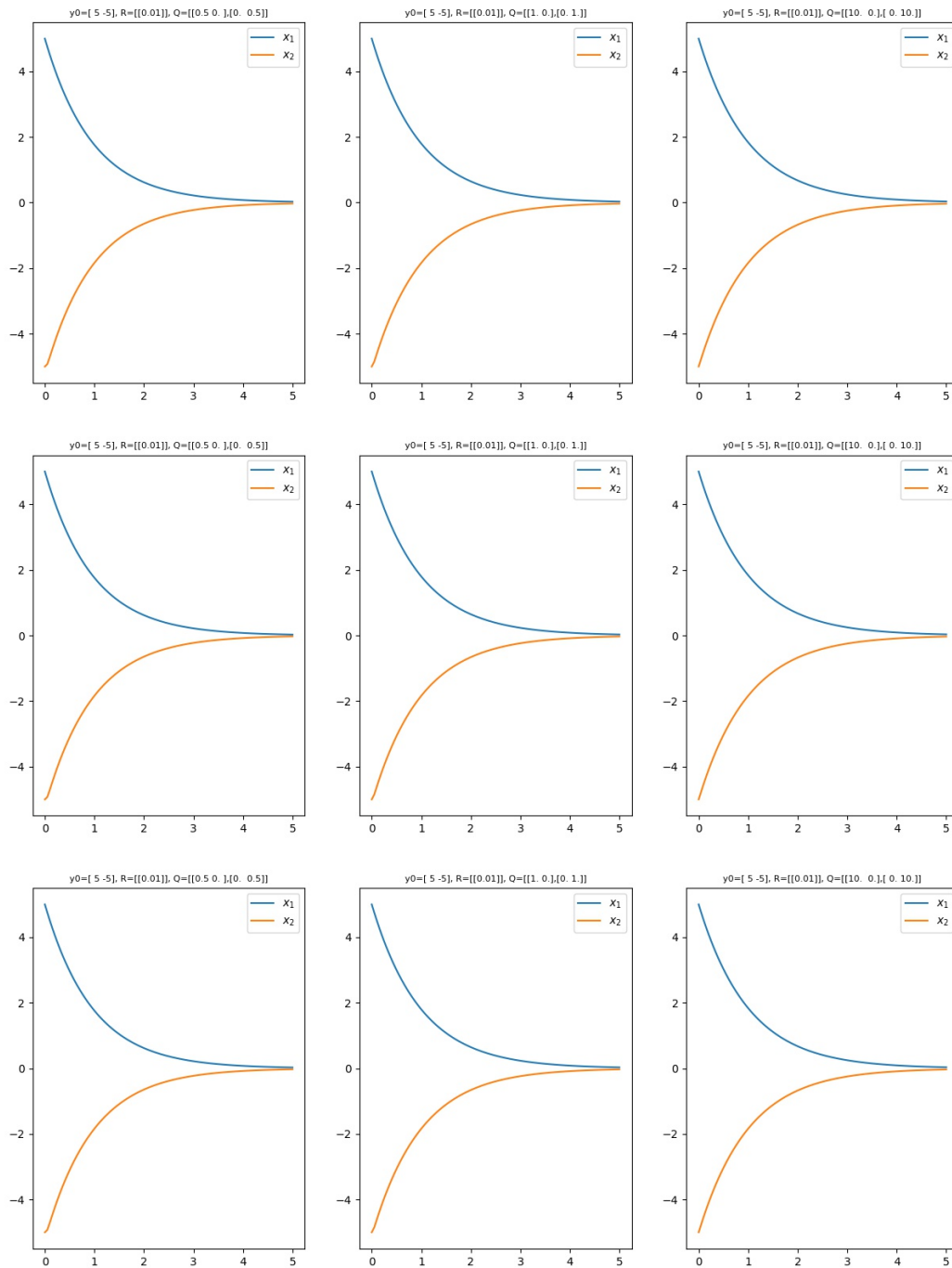
```

160         self,
161         x: np.ndarray,
162         t: float,
163         A: np.ndarray,
164         B: np.ndarray,
165         R: np.ndarray,
166         p: Callable) -> np.ndarray:
167
168     x = x.reshape((2, 1))
169     k = inv(R)@B.T@p(t)
170     u = -k@x
171     dx = A@x + B@u
172     return dx.flatten()
173
174     def run(self, y0: np.ndarray, t: np.ndarray) -> np.ndarray:
175         return integrate.odeint(
176             self.model,
177             y0,
178             t,
179             (self.A, self.B, self.R, self.p)
180         )
181
182     def plot(self, y0: np.ndarray, ax: plt.Axes) -> None:
183         res = self.run(y0, self.t)
184         ax.plot(self.t, res)
185         ax.set_title(
186             f'y0={y0}, R={self.R}, Q=[{self.Q[0]},{self.Q[1]}]',
187             fontsize=8
188         )
189         ax.legend(['$x_1$', '$x_2$'])
190
191 Rs = [np.array([[x]]) for x in [0.01, 0.5, 10]]
192 Qs = [np.eye(2)*x for x in [0.5, 1, 10]]
193
194 t = np.linspace(0, 5, 100)
195 experiments = [[]]*len(Rs)
196 for i, r in enumerate(Rs):
197     for q in Qs:
198         experiments[i].append(
199             Experiment(A, B, q, r, t)
200         )
201
202 fig, axes = plt.subplots(len(Rs), len(Qs))
203 for i in range(len(Rs)):
204     for j in range(len(Qs)):
205         experiments[i][j].plot(
206             np.array([5, -5]), axes[i][j]
207         )
208

```

```
209 fig.set_size_inches(15, 20)
210 plt.show()
211 plt.close()
```

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. Rozszerzyć funkcję $model(x, t)$ o wyznaczanie wartości wskaźnika jakości J . Funkcja $model(x, t)$ powinna wyznaczać pochodną (tj. wyrażenie podcałkowe) wskaźnika J jako dodatkową zmienną stanu – zostanie ona scałkowana przez `odeint`, a jej wartość zwrócona po zakończeniu symulacji

```
214 def model(  
215     x: np.ndarray,  
216     t: float,  
217     A: np.ndarray,  
218     B: np.ndarray,  
219     R: np.ndarray,  
220     Q: np.ndarray,  
221     p: Callable) -> np.ndarray:  
222     last_t = x[0]  
223     dj = x[1]  
224     x = x[2:].reshape((2, 1))  
225  
226     k = inv(R)@B.T@p(t)  
227     u = -k@x  
228     dx = A@x + B@u  
229  
230     dj += (x.T@Q@x + u.T@R@u)*(t - last_t)  
231     return np.append(  
232         np.array([t]),  
233         np.append(dj.flatten(), dx.flatten())  
234     )  
235  
236 res = integrate.odeint(  
237     model,  
238     [0, 0, 1, 1],  
239     np.linspace(0, 5, 100),  
240     (A, B, R, Q, p)  
241 )  
242 x1 = res.T[2:, 0]  
243 J = res.T[1, -1] + x1.T@S@x1  
244 print('J = ', J)
```

Output:

J = 48.035868144125565

Czy wyznaczona wartość rzeczywiście odpowiada minimalizowanemu wyrażeniu? W jakim horyzoncie czasu została ona wyznaczona?

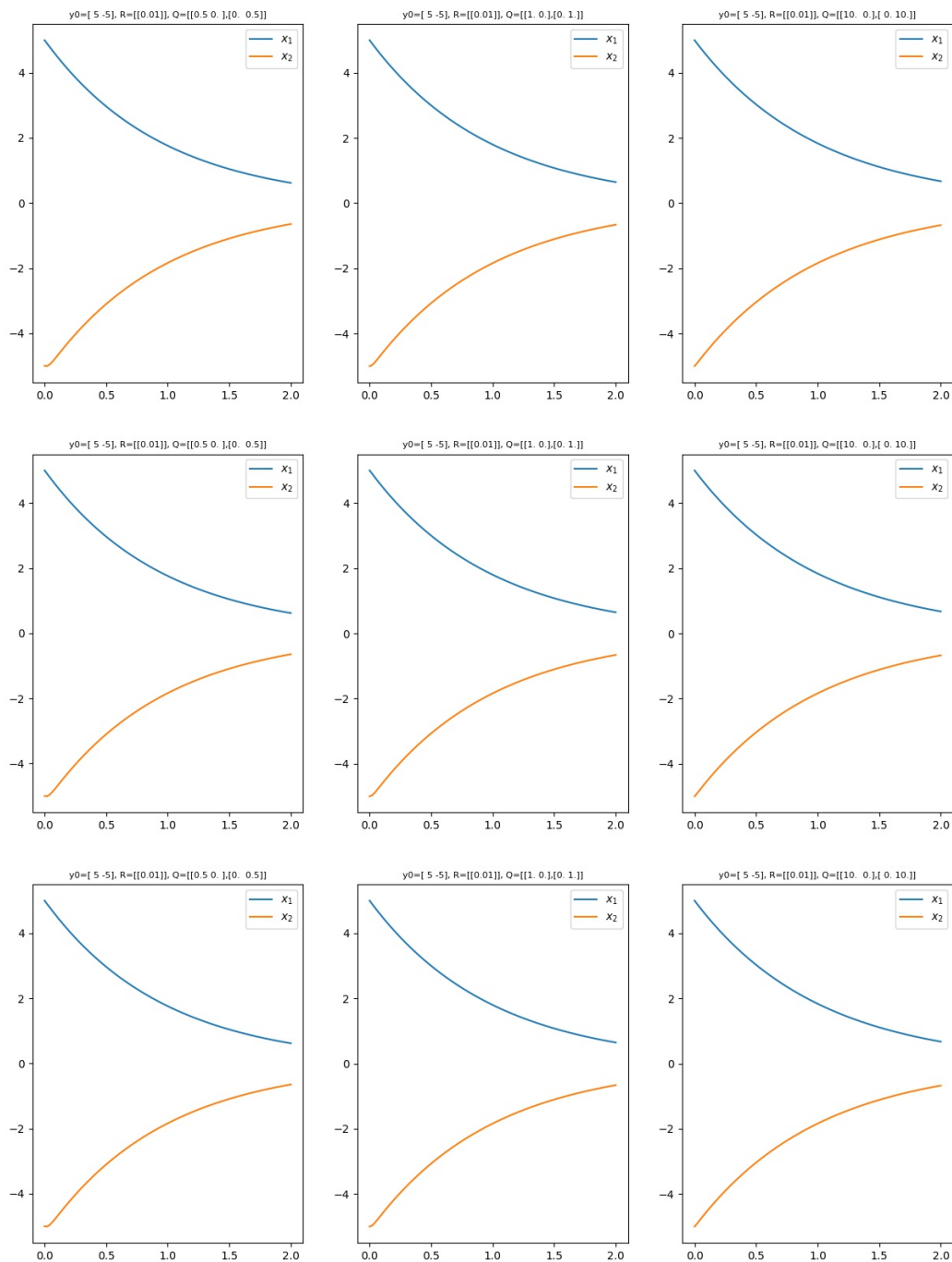
Wartość odpowiada minimalnemu wyrażeniu.

Została ona wyznaczona w przedziale $t \in (0, 5)$.

8. Powtórzyć symulację dla $t_1 = 2s$ oraz zmiennych wartości nastaw S , Q , R .

```
247 class Experiment2(Experiment):
248
249     def plot(
250         self,
251         y0: np.ndarray,
252         ax: plt.Axes,
253         t: np.ndarray) -> None:
254
255         res = self.run(y0, t)
256         ax.plot(t, res)
257         ax.set_title(
258             f'y0={y0}, R={self.R}, Q=[{self.Q[0]},{self.Q[1]}]',
259             fontsize=8
260         )
261         ax.legend(['$x_1$', '$x_2$'])
262
263 t = np.linspace(0, 5, 100)
264 experiments = [[]]*len(Rs)
265 for i, r in enumerate(Rs):
266     for q in Qs:
267         experiments[i].append(
268             Experiment2(A, B, q, r, t)
269         )
270
271 fig, axes = plt.subplots(len(Rs), len(Qs))
272 for i in range(len(Rs)):
273     for j in range(len(Qs)):
274         experiments[i][j].plot(
275             np.array([5, -5]),
276             axes[i][j],
277             np.linspace(0, 2, 100)
278         )
279
280 fig.set_size_inches(15, 20)
281 plt.show()
282 plt.close()
```

Output:



Czy układ osiąga stan ustalony? Jaki teraz wpływ mają poszczególne nastawy?
W krótszym czasie układ nie osiąga stanu ustalonego.

Wpływ nastaw jest identyczny jak w podpunkcie 6.