

# 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. Wyznaczyć macierze stanu dla wahadła z rysunku, a na następnie zaimplementować je do programu

```
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. Przygotować funkcję *riccati(p,t)* Rozwiązującego równanie SDRE zarówno dla skończonego jak i nieskończonego horyzontu czasowego. W wariancie z skończonym horyzontem czasowym zde-

finiować wektor chwil czasu od  $t_1$  do 0 przyjmując  $t_1 = 5s$  Wykorzystując funkcję `odeint` dla obu przebiegów wyznaczyć przebieg wartości macierzy  $P$  w czasie. Dla nieskończonego horyzontu czasowego wykorzystać `scipy.linalg.solve_continuous_are`. 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()`.

```

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     suptitle: str,
75     shape: Tuple[int, int],
76     size: Tuple[int, int]=(8,10)) -> None:
77     plt.figure(figsize=size)

```

```

78     plt.suptitle(suptitle)
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))

```

*Porównaj oba wyniki dla obu wariantów. Spróbuj zmienić początkowe wartości macierzy  $P$ ,  $Q$  oraz  $R$  pozwalają dowolnie kształtować przebieg uchybu regulacji? Czy istnieje jakaś zależność między doбором tych macierzy?*

Macierze  $Q$  i  $R$  pozwalają dowolnie kształtować przebieg uchybu regulacji

Macierz  $P$  jest zależna od  $Q$  i  $R$

Macierz  $Q$  określa funkcję kosztu dla zadanego uchybu regulacji

Macierz  $R$  określa funkcję kosztu dla zadanego sterowania

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

```

103 concatenate_plots(
104     show_P,
105     [
106         (t, res_fin, 'Finite Riccati'),
107         (t, res_inf, 'Infinite Riccati')
108     ],
109     '$P$ components vs time',
110     (2, 1)
111 )
112
113 S = solve_continuous_are(
114     A_of_x(
115         np.array([0,0]).reshape((2, 1))
116     ), B, Q, R

```

```

117 )
118
119 print('S =\n', S)
120 print('P_fin =\n', res_fin[-1, :4].reshape((2, 2)))
121 print('P_inf =\n', res_inf[-1, :4].reshape((2, 2)))

```

### Output:

```

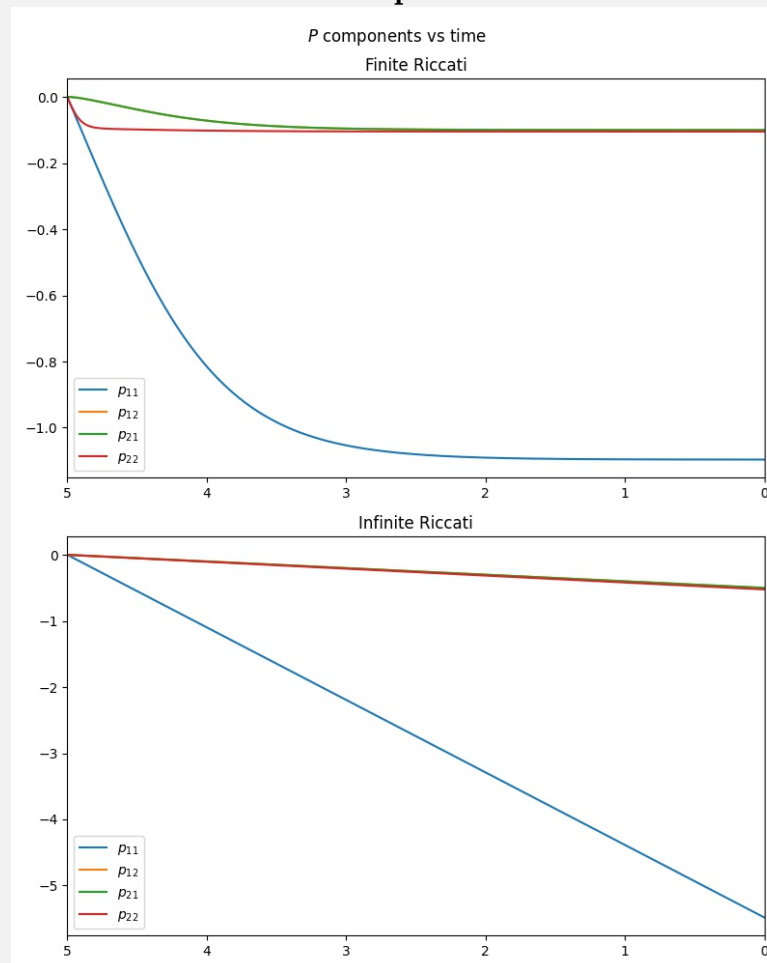
S =      [[1.09658561  0.1          ]
          [0.1         0.10465856]]

P_fin = [[-1.09647881 -0.09998928]
          [-0.09998928 -0.10465748]]

P_inf = [[-5.48292805 -0.5         ]
          [-0.5        -0.5232928  ]]

```

### Output:



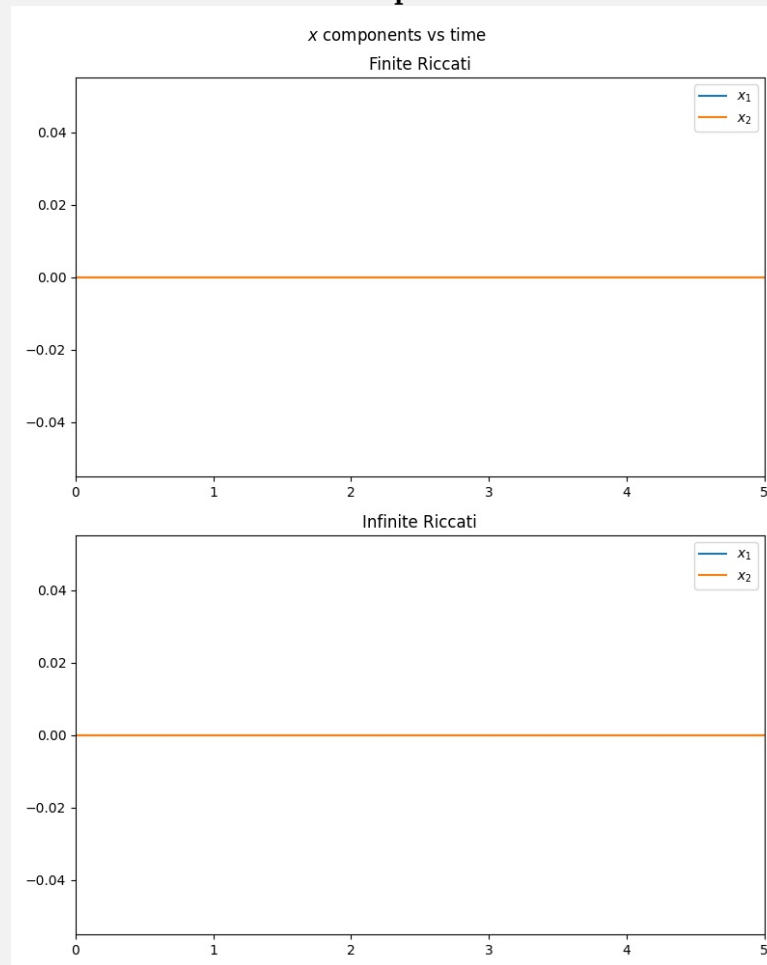
$P$  dla nieskończonego horyzontu jest nieskończone

$P$  dla skończonego horyzontu dąży do  $-S$ , nie udało mi się dojść do tego dlaczego

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

```
124 def show_x(t: np.ndarray, res: np.ndarray, title:str) -> None:
125     plt.plot(t, res[:, 4:6])
126     plt.title(title)
127     plt.xlim(t[0], t[-1])
128     plt.legend(['$x_1$', '$x_2$'])
129
130 t = np.linspace(0, 5, RESOLUTION)
131
132 res_fin = odeint(
133     riccati_finite_diff,
134     P0 + x0,
135     t,
136     (A_of_x, B, Q, R))
137
138 res_inf = odeint(
139     riccati_infinite_diff,
140     P0 + x0,
141     t,
142     (A_of_x, B, Q, R))
143
144 concatenate_plots(
145     show_x,
146     [
147         (t, res_fin, 'Finite Riccati'),
148         (t, res_inf, 'Infinite Riccati')
149     ],
150     '$x$ components vs time',
151     (2, 1)
152 )
```

## Output:



5. Przeprowadzić symulację układu dla niezerowych warunków początkowych. Zbadać wpływ macierzy  $Q$  oraz  $R$  na przebieg odpowiedzi układu.

```
155 x0 = [np.pi/2, 0]
156 QR = [ # q11, q22, r11
157         (1, 0.01, 0.01),
158         (0.01, 1, 0.01),
159         (5, 5, 0.01),
160         (10, 1, 0.01),
161         (1, 10, 0.01),
162         (1, 1, 0.01),
163         (1, 1, 0.1),
164         (1, 1, 1),
165         (1, 1, 10),
166     ]
167 def experiment(
168     riccati_func: Callable,
169     QR: List[Tuple[float, float, float]],
170     x0: List[float],
171     t: np.ndarray,
172     suptitle: str) -> np.ndarray:
```

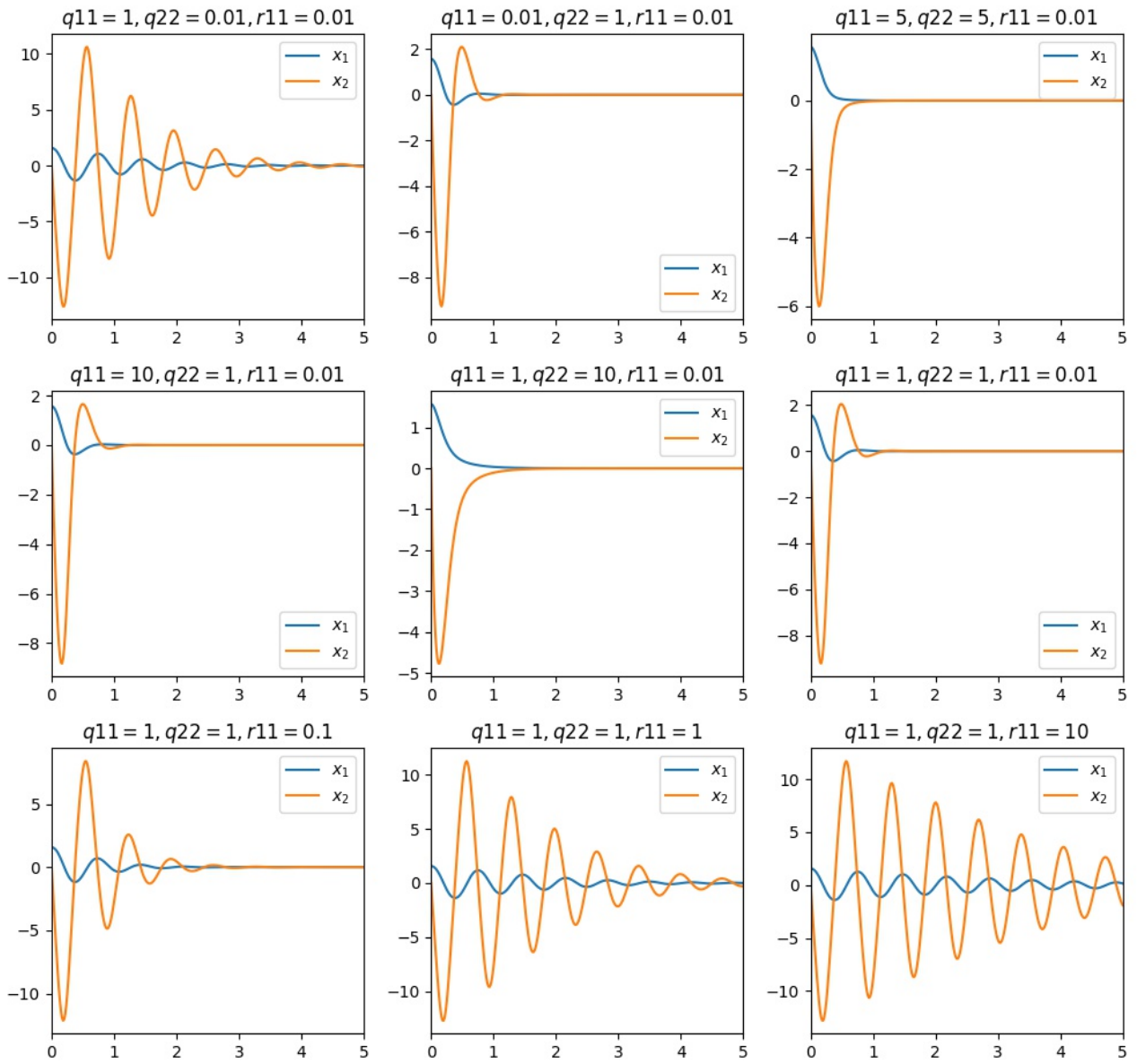
```

173     res = []
174     titles = []
175
176     for q11, q22, r11 in QR:
177         Q = np.array([[q11, 0], [0, q22]])
178         R = np.array([[r11]])
179         res.append(odeint(
180             riccati_func,
181             P0 + x0,
182             t,
183             (A_of_x, B, Q, R)))
184         titles.append(f'$q11={q11}, q22={q22}, r11={r11}$')
185
186     concatenate_plots(
187         show_x,
188         [(t, res[i], titles[i]) for i in range(len(res))],
189         suptitle,
190         (3, 3),
191         (10, 10)
192     )
193
194     experiment(
195         riccati_finite_diff, QR, x0, t,
196         '$x$ components vs time with different $Q$ and $R$\n\
197         (finite riccati)'
198     )
199
200     experiment(
201         riccati_infinite_diff, QR, x0, t,
202         '$x$ components vs time with different $Q$ and $R$\n\
203         (infinite riccati)'
204     )

```

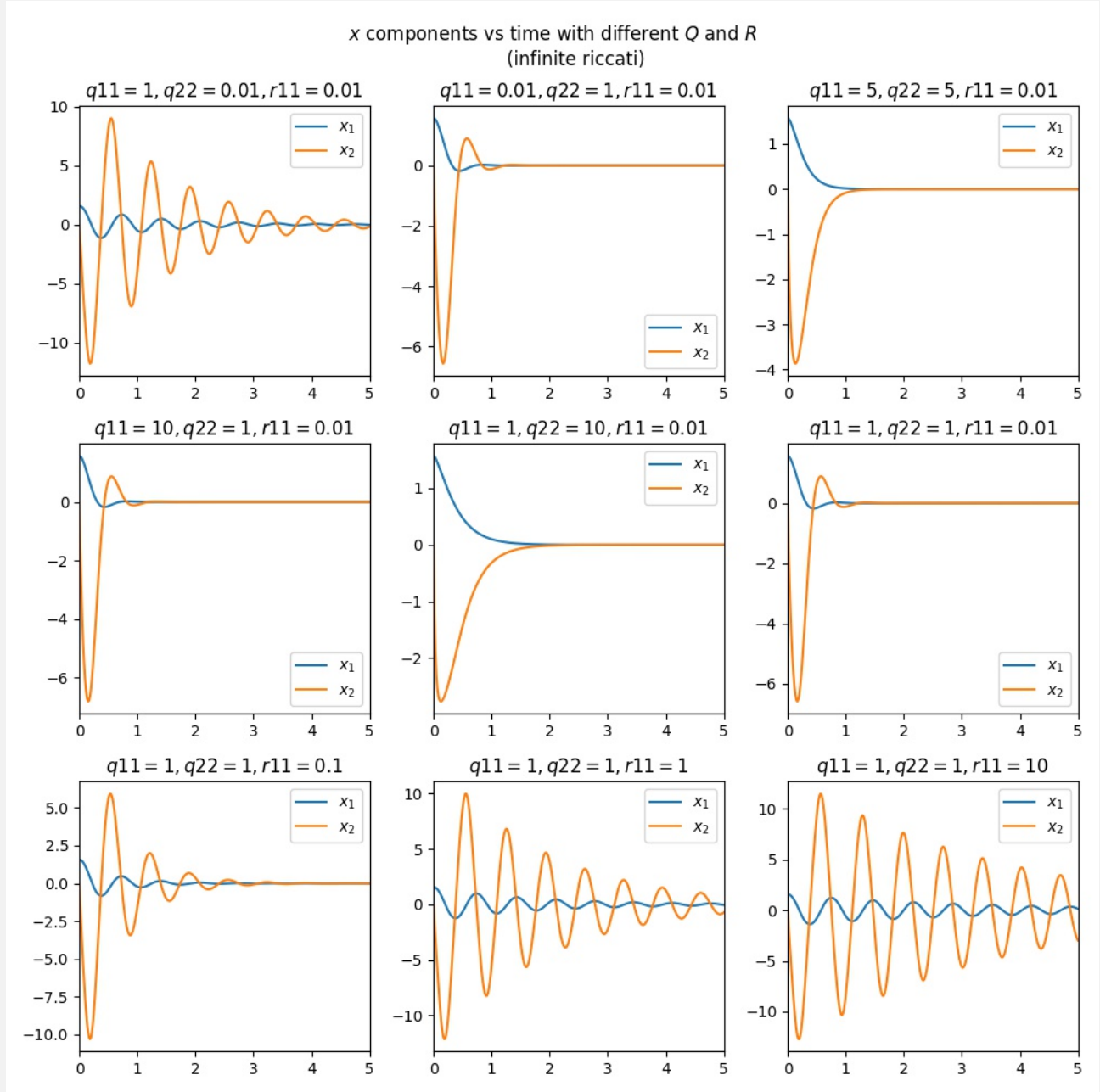
## Output:

x components vs time with different  $Q$  and  $R$   
(finite riccati)





## Output:



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

Macierze  $Q$  i  $R$  pozwalają dowolnie kształtować przebieg uchybu regulacji

Macierz  $Q$  określa funkcję kosztu dla zadanego uchybu regulacji

Macierz  $R$  określa funkcję kosztu dla zadanego sterowania

6. 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

```

207 def riccati_finite_diff_with_J(
208     x: np.ndarray,
209     t: float,
210     a: Callable,

```

```

211         B: np.ndarray,
212         Q: np.ndarray,
213         R: np.ndarray) -> np.ndarray:
214     P = x[:4].reshape((2, 2))
215     J = x[-1:].reshape((1, 1))
216     previous_t = x[-2]
217     x = x[4:6].reshape((2, 1))
218     A = a(x)
219     dP = -P@A - P@B@inv(R)@B.T@P - A.T@P + Q
220     p = P@x
221     u = -inv(R)@B.T@p
222     dx = A@x + B@u
223     J += (x.T@Q@x + u.T@R@u)*(t - previous_t)
224     return np.concatenate(
225         (dP.flatten(), dx.flatten(), [t], J.flatten())
226     )
227
228 def riccati_infinite_diff_with_J(
229     x: np.ndarray,
230     t: float,
231     a: Callable,
232     B: np.ndarray,
233     Q: np.ndarray,
234     R: np.ndarray) -> np.ndarray:
235     J = x[-1:].reshape((1, 1))
236     previous_t = x[-2]
237     x = x[4:6].reshape((2, 1))
238     A = a(x)
239     P = solve_continuous_are(A, B, Q, R)
240     u = -inv(R)@B.T@P@x
241     dx = A@x + B@u
242     J += (x.T@Q@x + u.T@R@u)*(t - previous_t)
243     return np.concatenate(
244         (P.flatten(), dx.flatten(), [t], J.flatten())
245     )
246
247 def show_J(t: np.ndarray, res: np.ndarray, title: str) -> None:
248     plt.plot(t, res[:, -1])
249     plt.title(title)
250     plt.xlim(t[0], t[-1])
251     plt.legend(['$J$'])
252
253 J0 = [0]
254 t0 = [0]
255
256 res_fin = odeint(
257     riccati_finite_diff_with_J,
258     P0 + x0 + t0 + J0,
259     t,

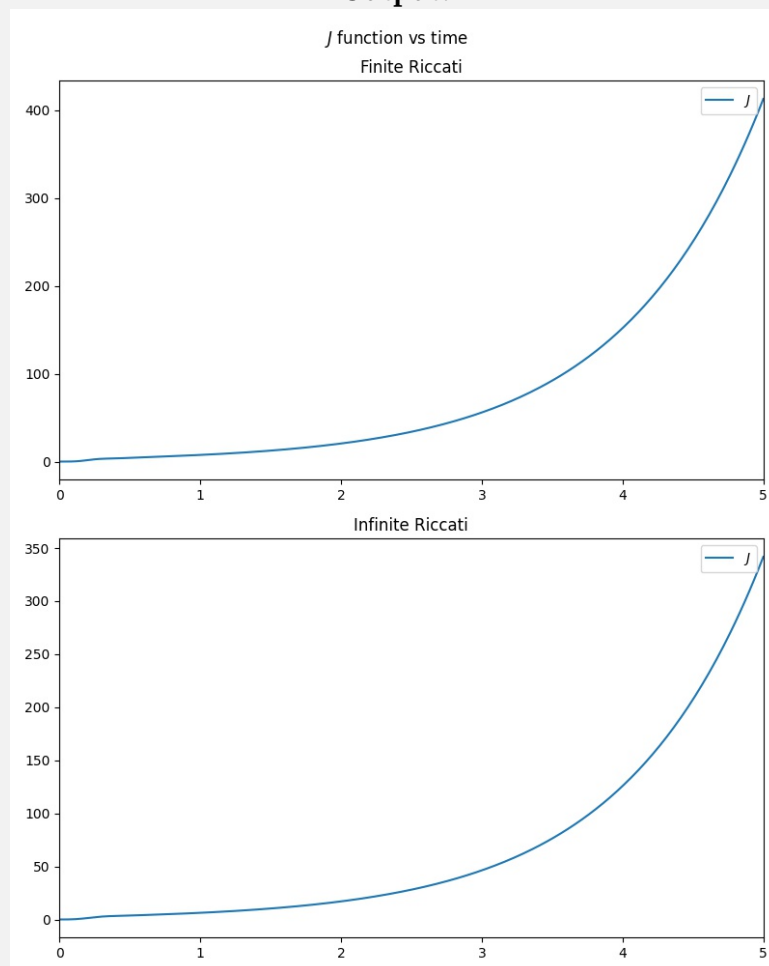
```

```

260     (A_of_x, B, Q, R))
261
262 res_inf = odeint(
263     riccati_infinite_diff_with_J,
264     P0 + x0 + t0 + J0,
265     t,
266     (A_of_x, B, Q, R))
267
268 concatenate_plots(
269     show_J,
270     [
271         (t, res_fin, 'Finite Riccati'),
272         (t, res_inf, 'Infinite Riccati')
273     ],
274     '$J$ function vs time',
275     (2, 1)
276 )

```

### Output:



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

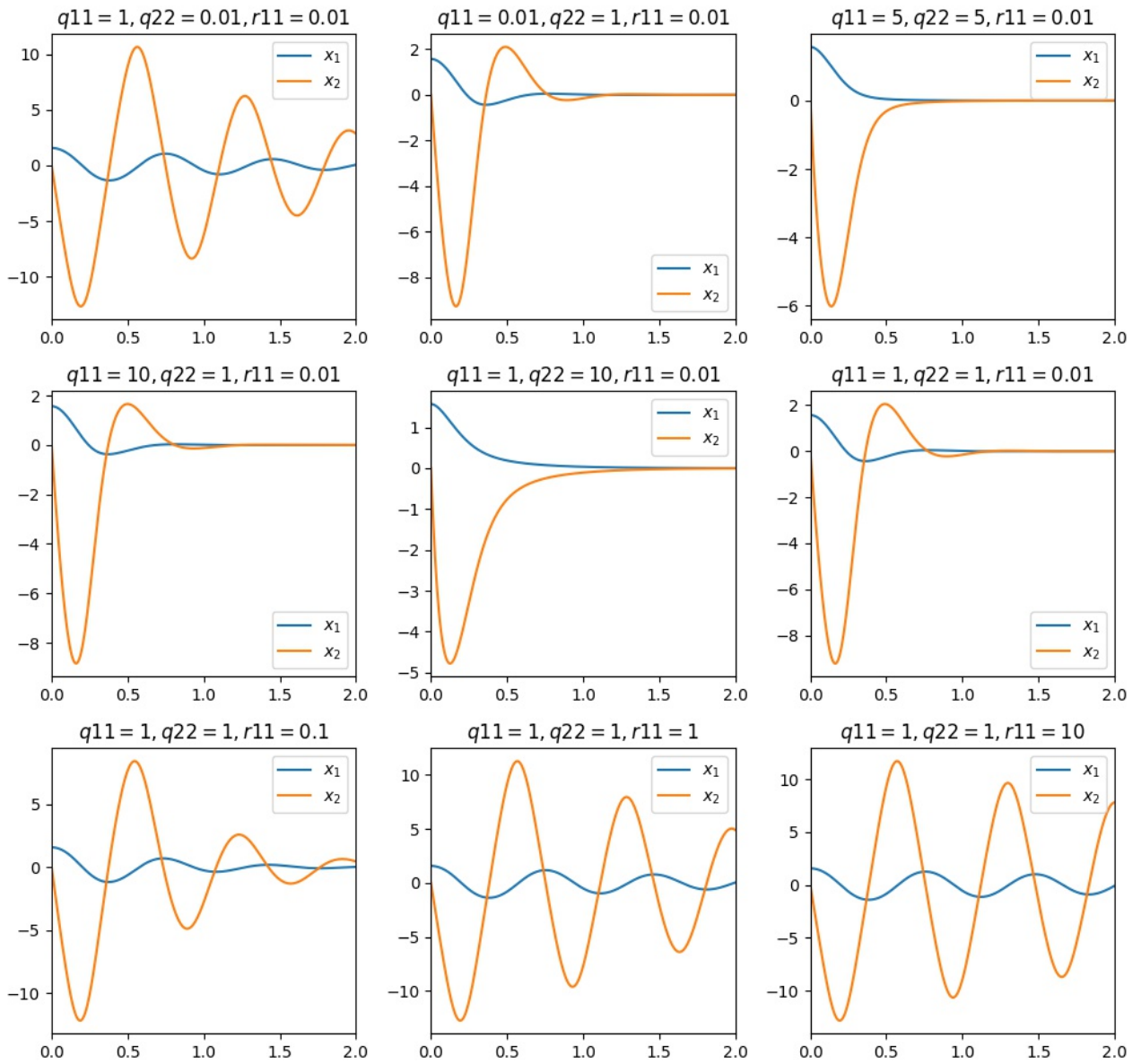
Wyznaczona wartość  $J$  odpowiada minimalnemu koszcie regulacji  
Została wyznaczona w czasie  $t \in (0, 5)$

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

```
279 t = np.linspace(0, 2, RESOLUTION)
280
281 experiment(
282     riccati_finite_diff, QR, x0, t,
283     '$x$ components vs time with different $Q$ and $R$\n\
284     (finite riccati)'
285 )
286
287 experiment(
288     riccati_infinite_diff, QR, x0, t,
289     '$x$ components vs time with different $Q$ and $R$\n\
290     (infinite riccati)'
291 )
```

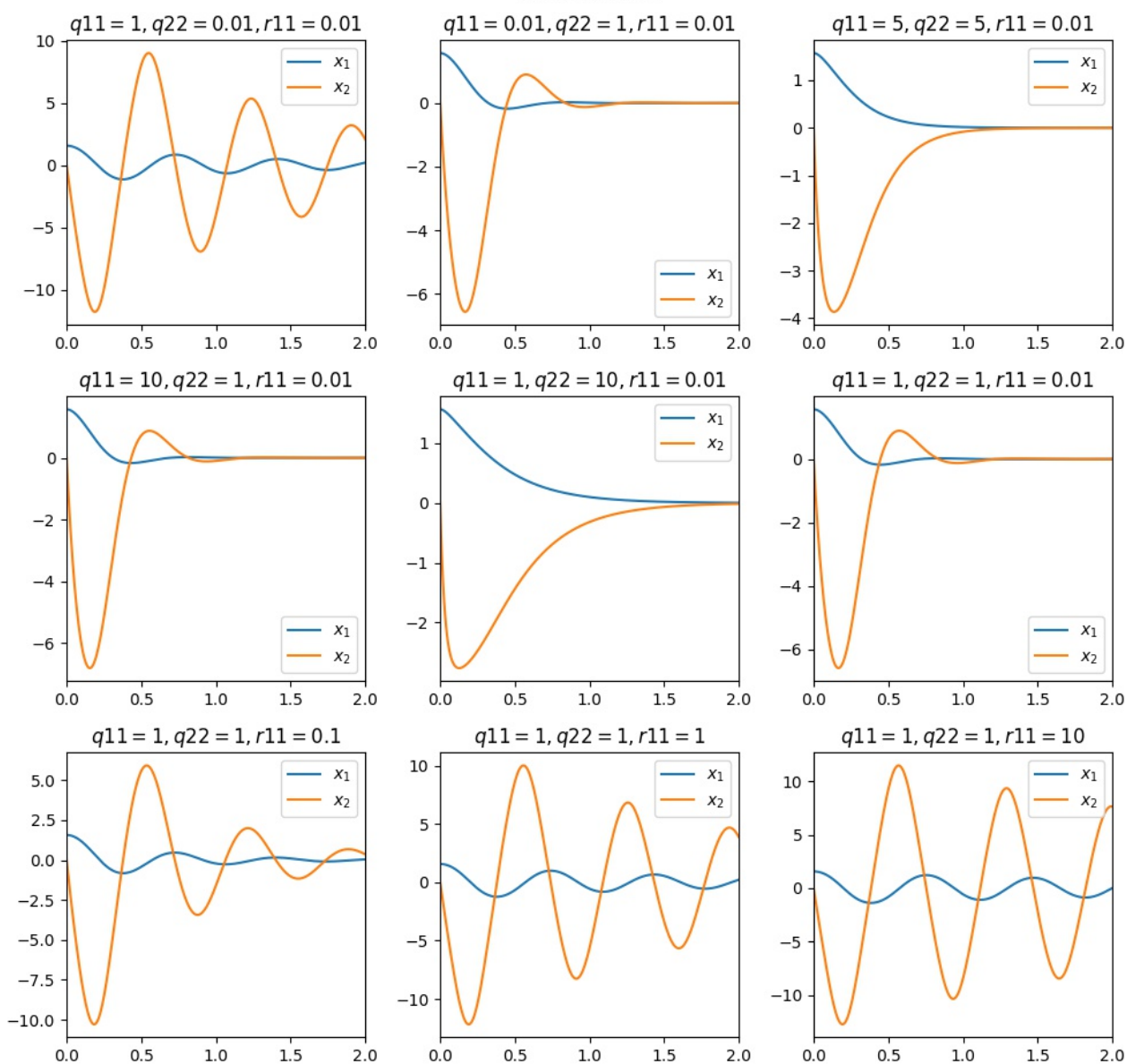
## Output:

x components vs time with different  $Q$  and  $R$   
(finite riccati)



## Output:

x components vs time with different Q and R  
(infinite riccati)



Czy układ osiąga stan ustalony? Jaki teraz wpływ mają poszczególne nastawy?

Układ osiąga stan ustalony jedynie przy niektórych nastawach

Nastawy mają identyczny wpływ jak w przypadku symulacji dla  $t_1 = 5s$