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)* implementująca 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*.**

```
RESOLUTION = 300
L = 1 # R
m = 9
J = 1
g = 10
d = 0.5

def A_of_x(x: np.ndarray) -> np.ndarray:
    if x[0, 0] == 0:
        return np.array([
            [0, 1],
            [0, -d/J]
        ]).astype(float)
    return np.array([
        [0, 1],
        [-m*g*L*np.sin(x[0, 0])/(J*x[0, 0]), -d/J]
    ]).astype(float)

B = np.array([[0, 1/J]]).T
R = np.array([[0.01]])
Q = np.array([
    [1, 0],
    [0, 1]
])
```

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

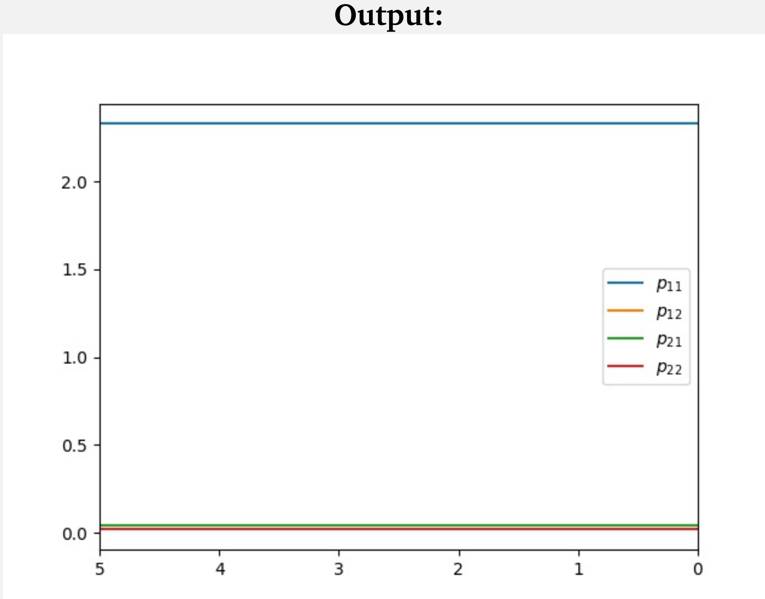
```
def riccati_finite_diff(
        x: np.ndarray,
        t: float,
        a: Callable,
        B: np.ndarray,
        Q: np.ndarray,
        R: np.ndarray) -> np.ndarray:
    P = x[:4].reshape((2, 2))
    x = x[4:].reshape((2, 1))
    A = a(x)
    dP = -P@A - P@B@inv(R)@B.T@P - A.T@P + Q
    p = P@x
    u = -inv(R)@B.T@p
    dx = A@x + B@u
    return np.concatenate((dP.flatten(), dx.flatten()))

def riccati_infinite_diff(
        x: np.ndarray,
        t: float,
        a: Callable,
        B: np.ndarray,
        Q: np.ndarray,
        R: np.ndarray) -> np.ndarray:
    x = x[4:].reshape((2, 1))
    A = a(x)
    P = solve_continuous_are(A, B, Q, R)
    u = -inv(R)@B.T@P@x
    dx = A@x + B@u
    return np.concatenate((P.flatten(), dx.flatten()))

def show_P(t: np.ndarray, res: np.ndarray, title: str) -> None:
    plt.plot(t, res[:, :4])
    plt.title(title)
    plt.xlim(t[0], t[-1])
    plt.legend(['$p_{11}$', '$p_{12}$', '$p_{21}$', '$p_{22}$'])

def concatenate_plots(
        plot_func: Callable,
        plots: List[Tuple[np.ndarray, np.ndarray, str]],
        suptitle: str,
        shape: Tuple[int, int],
        size: Tuple[int, int]=(8,10)) -> None:
    plt.figure(figsize=size)
    plt.suptitle(suptitle)
    for i, data in enumerate(plots):
        plt.subplot(*shape, i+1)
```

```
        plot_func(*data)
    plt.tight_layout()
    plt.show()
    plt.close()

x0 = [0]*2
P0 = [0]*4
t = np.linspace(5, 0, RESOLUTION)

res_fin = odeint(
    riccati_finite_diff,
    P0 + x0,
    t,
    (A_of_x, B, Q, R))

res_inf = odeint(
    riccati_infinite_diff,
    P0 + x0,
    t,
    (A_of_x, B, Q, R))
```

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.**

```
# Macierz Q określa funkcję kosztu dla zadanego uchybu regulacji
# Macierz R określa funkcję kosztu dla zadanego sterowania

# 4.3
concatenate_plots(
    show_P,
    [
        (t, res_fin, 'Finite Riccati'),
        (t, res_inf, 'Infinite Riccati')
    ],
    '$P$ components vs time',
    (2, 1)
)

S = solve_continuous_are(
    A_of_x(
        np.array([0,0]).reshape((2, 1))
    ), B, Q, R
)
```

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.**

```
print('P_fin =\n', res_fin[-1, :4].reshape((2, 2)))
print('P_inf =\n', res_inf[-1, :4].reshape((2, 2)))
# P dla nieskończonego horyzontu jest nieskończone
# P dla skończonego horyzontu dąży do -S, nie udało mi się dojść do tego
 ↪  dlaczego

# 4.4
def show_x(t: np.ndarray, res: np.ndarray, title:str) -> None:
    plt.plot(t, res[:, 4:6])
    plt.title(title)
    plt.xlim(t[0], t[-1])
    plt.legend(['$x_1$', '$x_2$'])

t = np.linspace(0, 5, RESOLUTION)

res_fin = odeint(
    riccati_finite_diff,
    P0 + x0,
    t,
    (A_of_x, B, Q, R))

res_inf = odeint(
    riccati_infinite_diff,
```

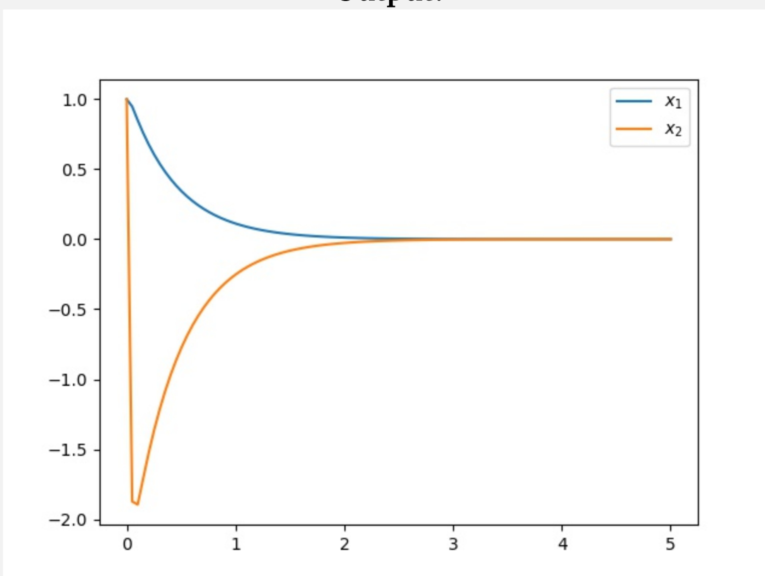
```
    P0 + x0,
    t,
    (A_of_x, B, Q, R))

concatenate_plots(
    show_x,
    [
```

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

```
    ],
    '$x$ components vs time',
    (2, 1)
)

# 4.5
x0 = [np.pi/2, 0]
QR = [ # q11, q22, r11
    (1, 0.01, 0.01),
    (0.01, 1, 0.01),
    (5, 5, 0.01),
    (10, 1, 0.01),
    (1, 10, 0.01),
    (1, 1, 0.01),
    (1, 1, 0.1),
    (1, 1, 1),
    (1, 1, 10),
]
def experiment(
        riccati_func: Callable,
        QR: List[Tuple[float, float, float]],
        x0: List[float],
        t: np.ndarray,
        suptitle:str) -> np.ndarray:
    res = []
    titles = []

    for q11, q22, r11 in QR:
        Q = np.array([[q11, 0], [0, q22]])
        R = np.array([[r11]])
        res.append(odeint(
            riccati_func,
            P0 + x0,
            t,
            (A_of_x, B, Q, R)))
        titles.append(f'$q11={q11}, q22={q22}, r11={r11}$')
```

```

    concatenate_plots(
        show_x,
        [(t, res[i], titles[i]) for i in range(len(res))],
        suptitle,
        (3, 3),
        (10, 10)
    )

experiment(
    riccati_finite_diff, QR, x0, t,
    '$x$ components vs time with different $Q$ and $R$\n\
        (finite riccati)'
)
```

Output:



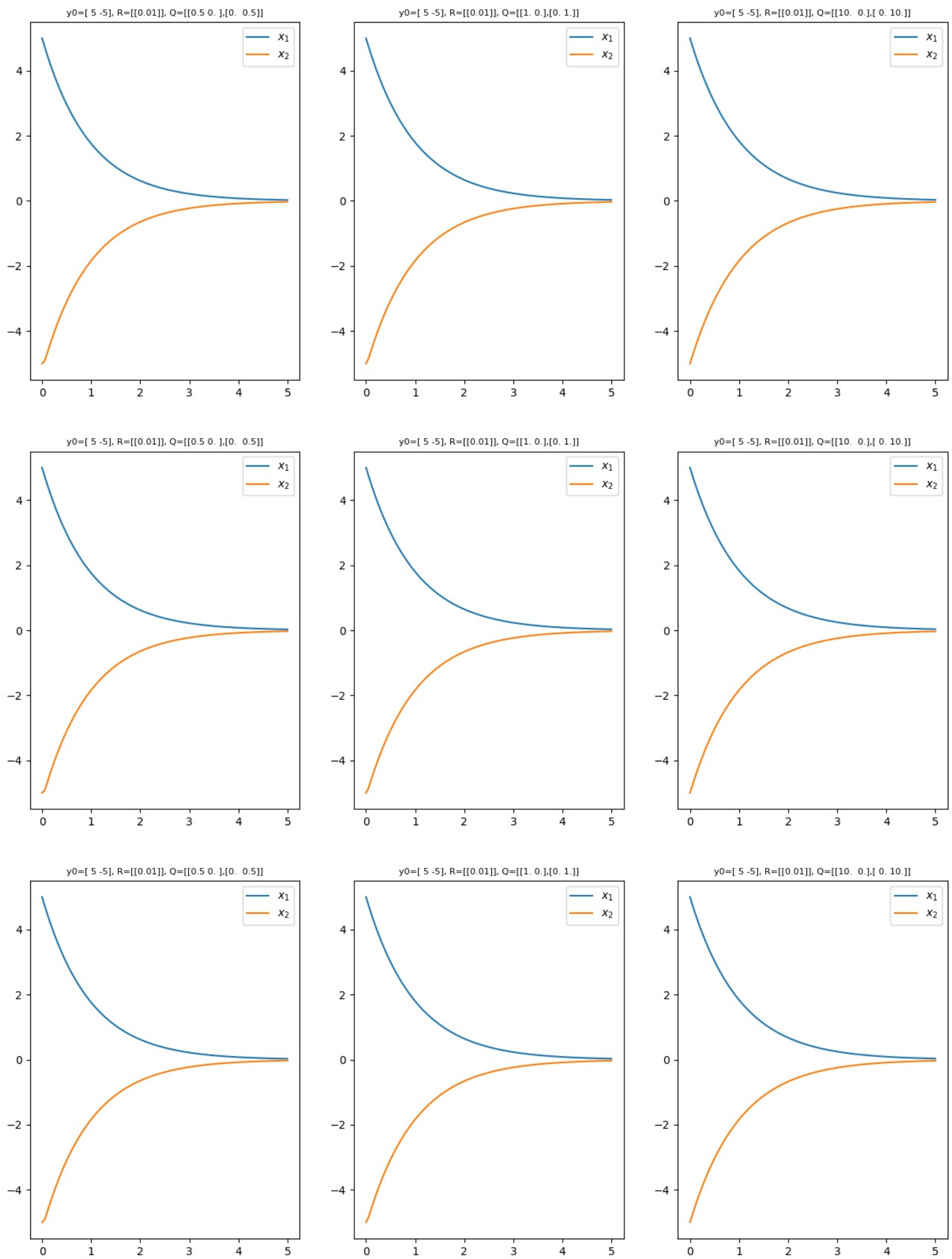
6. **Przeprowadzić symulację układu dla niezerowych warunków początkowych. Zbadać wpływ macierzy S, Q oraz R na przebieg odpowiedzi układu.**

```
    riccati_infinite_diff, QR, x0, t,
    '$x$ components vs time with different $Q$ and $R$\n\
        (infinite riccati)'
)

# 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

# 4.6
def riccati_finite_diff_with_J(
```

```
        x: np.ndarray,
        t: float,
        a: Callable,
        B: np.ndarray,
        Q: np.ndarray,
        R: np.ndarray) -> np.ndarray:
    P = x[:4].reshape((2, 2))
    J = x[-1:].reshape((1, 1))
    previous_t = x[-2]
    x = x[4:6].reshape((2, 1))
    A = a(x)
    dP = -P@A - P@B@inv(R)@B.T@P - A.T@P + Q
    p = P@x
    u = -inv(R)@B.T@p
    dx = A@x + B@u
    J += (x.T@Q@x + u.T@R@u)*(t - previous_t)
    return np.concatenate(
        (dP.flatten(), dx.flatten(), [t], J.flatten())
    )

def riccati_infinite_diff_with_J(
        x: np.ndarray,
        t: float,
        a: Callable,
        B: np.ndarray,
        Q: np.ndarray,
        R: np.ndarray) -> np.ndarray:
    J = x[-1:].reshape((1, 1))
    previous_t = x[-2]
    x = x[4:6].reshape((2, 1))
    A = a(x)
    P = solve_continuous_are(A, B, Q, R)
    u = -inv(R)@B.T@P@x
    dx = A@x + B@u
    J += (x.T@Q@x + u.T@R@u)*(t - previous_t)
    return np.concatenate(
        (P.flatten(), dx.flatten(), [t], J.flatten())
    )

def show_J(t: np.ndarray, res: np.ndarray, title:str) -> None:
    plt.plot(t, res[:, -1])
    plt.title(title)
    plt.xlim(t[0], t[-1])
    plt.legend(['$J$'])

J0 = [0]
t0 = [0]

res_fin = odeint(
```

```
    riccati_finite_diff_with_J,
    P0 + x0 + t0 + J0,
    t,
    (A_of_x, B, Q, R))

res_inf = odeint(
    riccati_infinite_diff_with_J,
    P0 + x0 + t0 + J0,
    t,
    (A_of_x, B, Q, R))
```

Output:



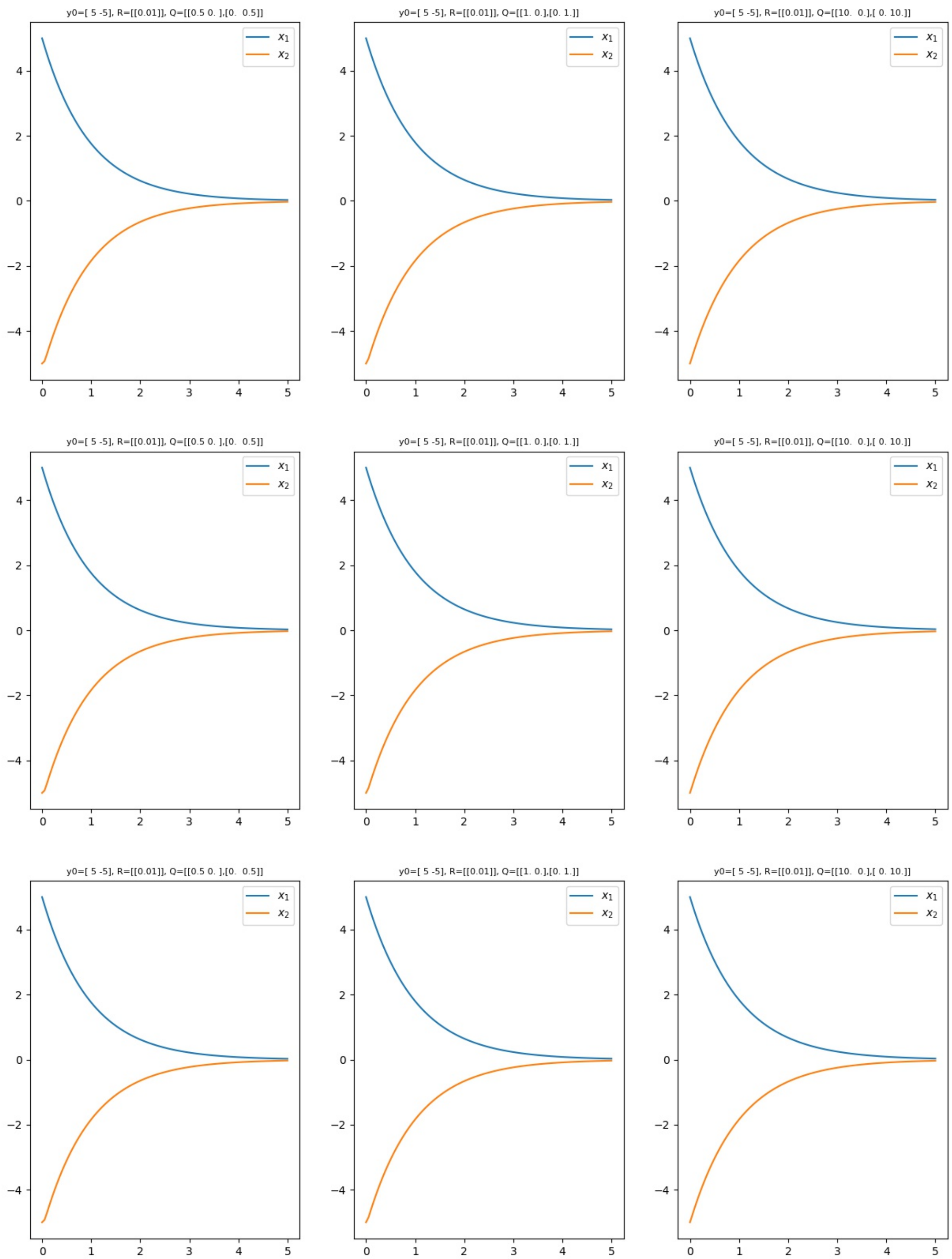
Czy macierze S, Q oraz R pozwalają dowolnie kształtować przebieg uchybu regula- cji? Czy istnieje jakaś zależność między doborem tych macierzy?

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.**

```
    show_J,
    [
        (t, res_fin, 'Finite Riccati'),
        (t, res_inf, 'Infinite Riccati')
    ],
    '$J$ function vs time',
    (2, 1)
)
# Wyznaczona wartość J odpowiada minimalnemu koszcie regulacji
# Została wyznaczona w czasie t \in [0, 5]

# 4.7
t = np.linspace(0, 2, RESOLUTION)
```

Output:



Czy macierze S, Q oraz R pozwalają dowolnie kształtować przebieg uchybu regula- cji? Czy istnieje jakaś zależność między doborem tych macierzy?

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