Podstawy Sterowania Optymalnego
Labolatorium 7

Sterowanie układami nieliniowymi przy pomocy metody SDRE

Prowadzący: mgr inż. Krzysztof Hałas
Wykonał: Ryszard Napierała

7 stycznia 2022

Importy

```
from typing import Callable
import numpy as np
from numpy.linalg import inv, eig
from scipy import integrate, interpolate
import matplotlib.pyplot as plt
import sympy as sym
from dataclasses import dataclass
```

Zadanie 2

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*.**

```
def riccati(
        p: np.ndarray,
        t: float,
        A: np.ndarray,
        B: np.ndarray,
        Q: np.ndarray,
        R: np.ndarray) -> np.ndarray:
    p = p.reshape((2, 2))
    dp = p@A - p@(B@inv(R)@B.T)@p + A.T@p + Q
    return dp.flatten()

R = 0.5
```

```python
C = 0.5
L = 0.2

A = np.array([
    [0, 1],
    [-1/(L*C), -R/L]
])
B = np.array([[0, 1/L]]).T

Q = np.array([
    [5, 0],
    [0, 1]
])
R = np.array([[0.01]])

def solve_for_S(
        a: np.ndarray,
        b: np.ndarray,
        q: np.ndarray,
        r: np.ndarray) -> np.ndarray:

    s11, s12, s22 = sym.symbols('s11 s12 s22')
    A = sym.Matrix(a)
    B = sym.Matrix(b)
    Q = sym.Matrix(q)
    R = sym.Matrix(r)
    S = sym.Matrix([
        [s11, s12],
        [s12, s22]
    ])
    res = sym.solve(
        A.T*S+S*A-S*B*(R**(-1))*B.T*S+Q,
        (s11, s12, s22)
    )

    for s11, s12, s22 in res:
        S = np.array([
            [s11, s12],
            [s12, s22],
        ]).astype(float)
        K = inv(r)@b.T@S
        e1, e2 = eig(a - b@K)[0]
        if e1 < 0 and e2 < 0:
            return S


S = solve_for_S(A, B, Q, R)

t = np.linspace(5, 0, 100)
```

```python

res = integrate.odeint(
    riccati,
    S.flatten(),
    t,
    (A, B, Q, R))
```

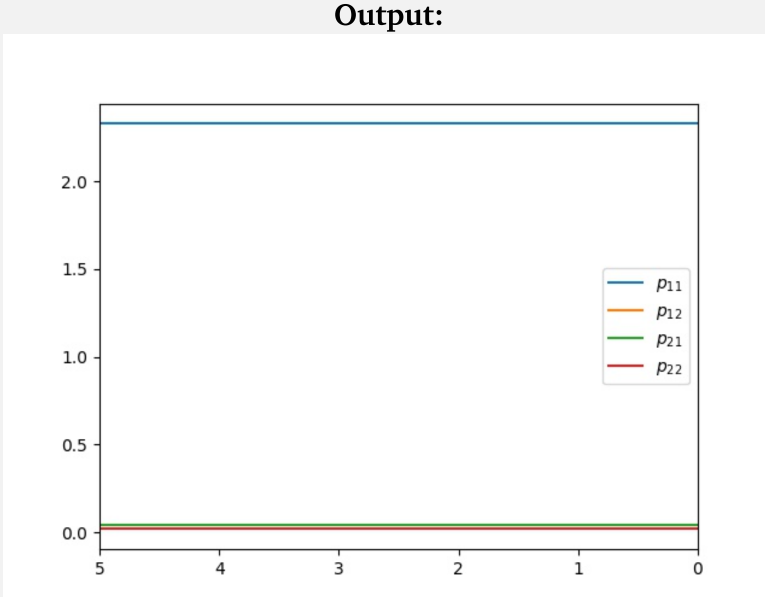
2. **Wykreślić przebieg elementów macierzy $P(t)$ w czasie. Zweryfikować poprawność wyni- ków poprzez porównanie z warunkiem krańcowym.**

```python
plt.plot(t, res)
plt.xlim(5, 0)
plt.legend(['$p_{11}$', '$p_{12}$', '$p_{21}$', '$p_{22}$'])
plt.show()
plt.close()
print(
    '(S==P) = ',
    np.all(np.isclose(S, res[-1].reshape((2,2))))
)
```

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.**

```
def model(
        x: np.ndarray,
        t: float,
        A: np.ndarray,
        B: np.ndarray,
        u: Callable) -> np.ndarray:
    x = x.reshape((2, 1))
    dx = A@x + B@u(t)
    return dx.flatten()
```

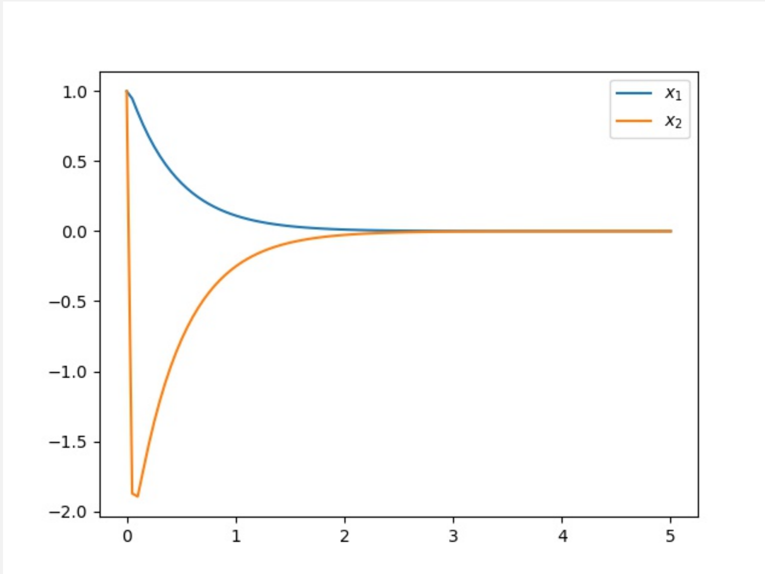
4. **Zmodyfikować funkcję *model(x,t)* tak, by wprowadzić do niej wyznaczone wcześniej wartości macierzy $P(t)$. Wykorzystać *interpolate.interp1d* w celu określenia wartości macierzy $P(t)$ w wybranej chwili czasu.**

```
def model(
        x: np.ndarray,
        t: float,
        A: np.ndarray,
        B: np.ndarray,
        R: np.ndarray,
        p: Callable) -> np.ndarray:
    x = x.reshape((2, 1))
    k = inv(R)@B.T@p(t)
    u = -k@x
    dx = A@x + B@u
    return dx.flatten()

i1d = interpolate.interp1d(
    t,
    res.T,
    fill_value=res.T[:, -1],
    bounds_error=False
)
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*.**

```
t = t[::-1]
res = integrate.odeint(model, [1, 1], t, (A, B, R, p))

plt.plot(t, res)
plt.legend(['$x_1$', '$x_2$'])
plt.show()
plt.close()
```

Output:



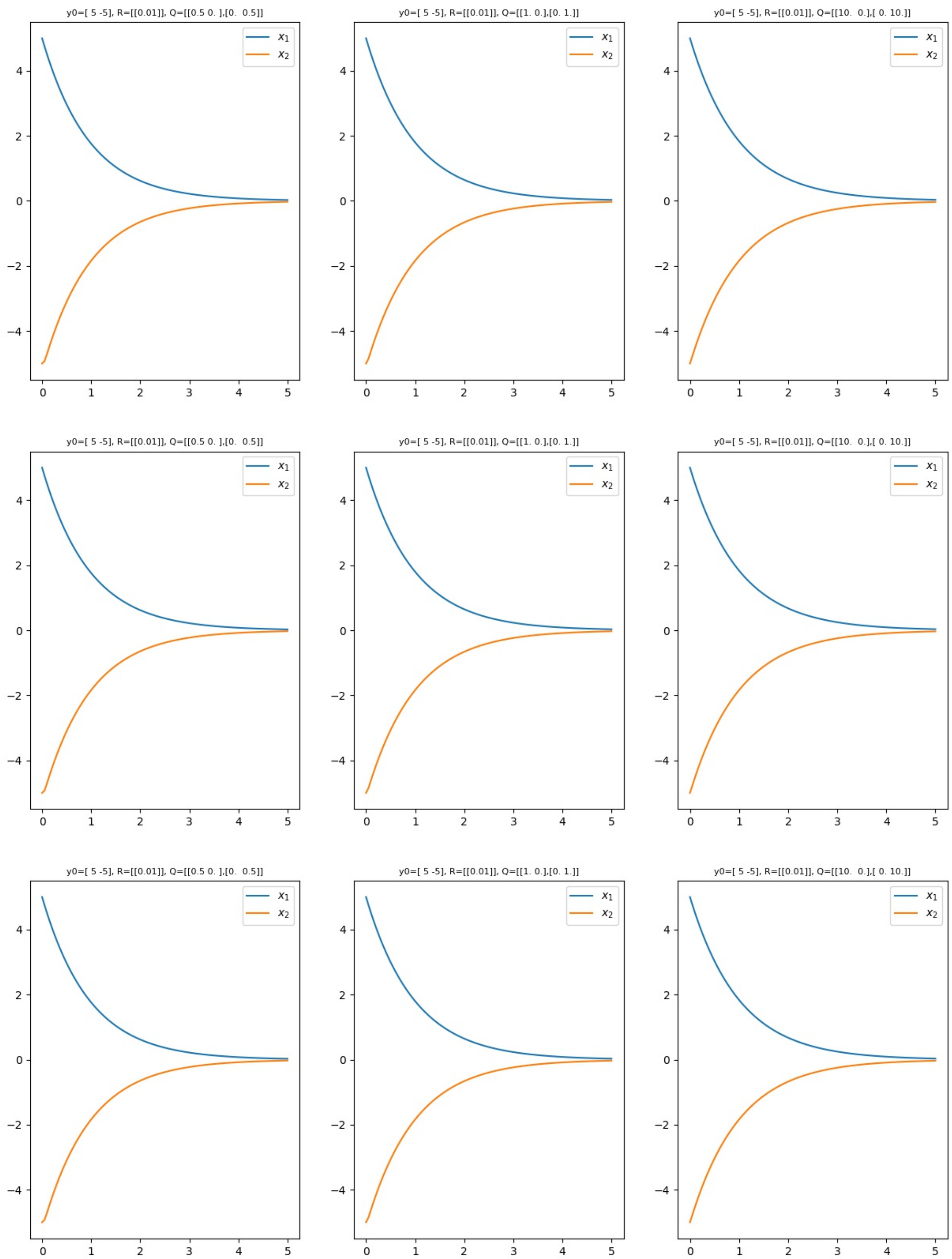
6. **Przeprowadzić symulację układu dla niezerowych warunków początkowych. Zbadać wpływ macierzy S, Q oraz R na przebieg odpowiedzi układu.**

```
@dataclass
class Experiment:
    A: np.ndarray
    B: np.ndarray
    Q: np.ndarray
    R: np.ndarray
    t: np.ndarray

    def __post_init__(self):
        S = solve_for_S(
            self.A, self.B, self.Q, self.R
        )
        res = integrate.odeint(
            riccati,
            S.flatten(),
            self.t[::-1],
            (self.A, self.B, self.Q, self.R)
        )
        i1d = interpolate.interp1d(
            self.t,
            res.T,
            fill_value=res.T[:, -1],
            bounds_error=False
        )
        self.p = lambda x: i1d(x).reshape((2, 2))

    def model(
```

```python
            self,
            x: np.ndarray,
            t: float,
            A: np.ndarray,
            B: np.ndarray,
            R: np.ndarray,
            p: Callable) -> np.ndarray:

        x = x.reshape((2, 1))
        k = inv(R)@B.T@p(t)
        u = -k@x
        dx = A@x + B@u
        return dx.flatten()

    def run(self, y0: np.ndarray, t: np.ndarray) -> np.ndarray:
        return integrate.odeint(
            self.model,
            y0,
            t,
            (self.A, self.B, self.R, self.p)
        )

    def plot(self, y0: np.ndarray, ax: plt.Axes) -> None:
        res = self.run(y0, self.t)
        ax.plot(self.t, res)
        ax.set_title(
            f'y0={y0}, R={self.R}, Q=[{self.Q[0]},{self.Q[1]}]',
            fontsize=8
        )
        ax.legend(['$x_1$', '$x_2$'])

Rs = [np.array([[x]]) for x in [0.01, 0.5, 10]]
Qs = [np.eye(2)*x for x in [0.5, 1, 10]]

t = np.linspace(0, 5, 100)
experiments = [[]]*len(Rs)
for i, r in enumerate(Rs):
    for q in Qs:
        experiments[i].append(
            Experiment(A, B, q, r, t)
        )

fig, axes = plt.subplots(len(Rs), len(Qs))
for i in range(len(Rs)):
    for j in range(len(Qs)):
        experiments[i][j].plot(
            np.array([5, -5]), axes[i][j]
        )
```

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

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. **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**

```python
def model(
        x: np.ndarray,
        t: float,
        A: np.ndarray,
        B: np.ndarray,
        R: np.ndarray,
        Q: np.ndarray,
        p: Callable) -> np.ndarray:
    last_t = x[0]
    dj = x[1]
    x = x[2:].reshape((2, 1))

    k = inv(R)@B.T@p(t)
    u = -k@x
    dx = A@x + B@u

    dj += (x.T@Q@x + u.T@R@u)*(t - last_t)
    return np.append(
        np.array([t]),
        np.append(dj.flatten(), dx.flatten())
    )

res = integrate.odeint(
    model,
    [0, 0, 1, 1],
    np.linspace(0, 5, 100),
    (A, B, R, Q, p)
)
x1 = res.T[2:, 0]
J = res.T[1, -1] + x1.T@S@x1
print('J = ', J)
```

Output:

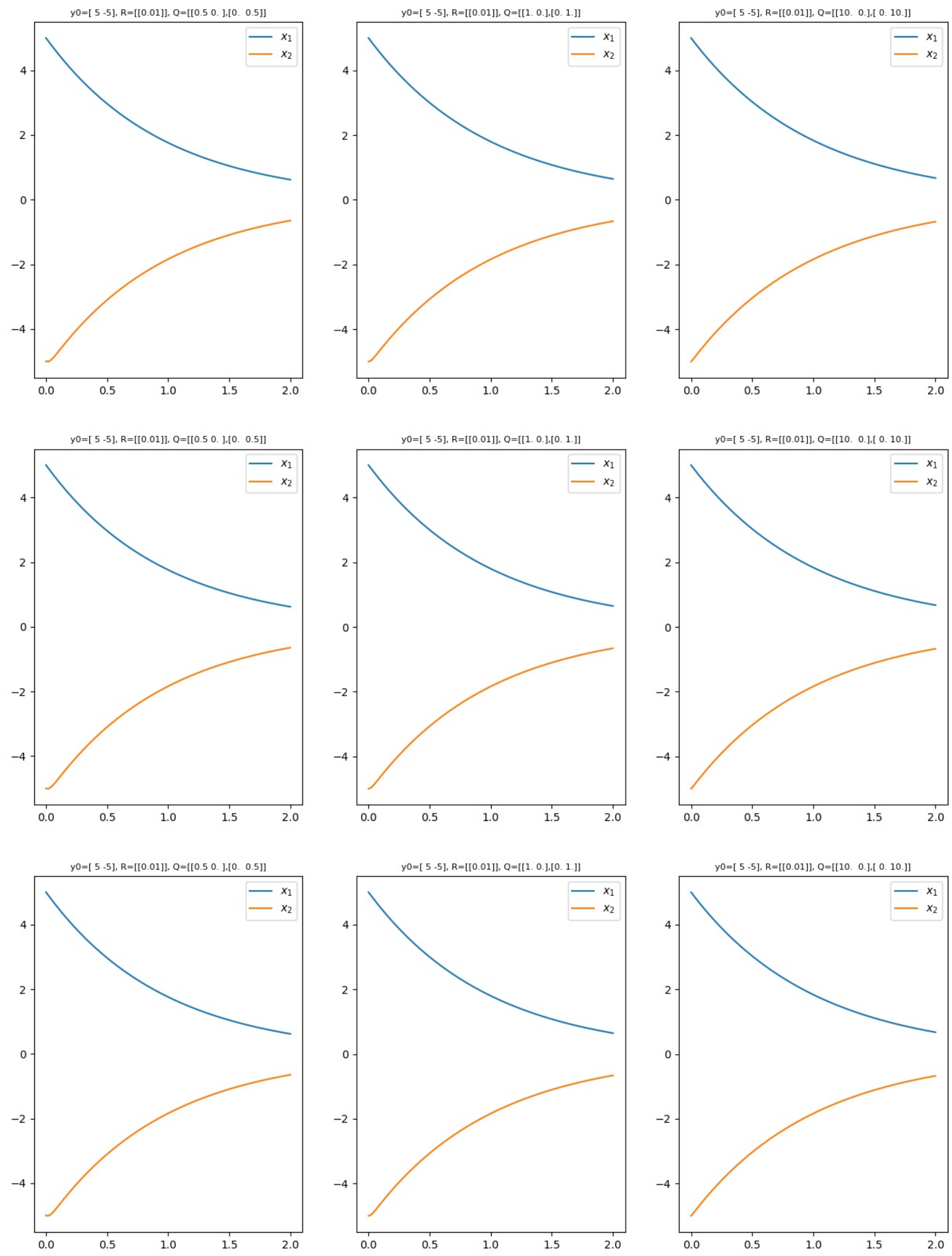
```
J =  48.035868144125565
```

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.**

```python
class Experiment2(Experiment):

    def plot(
            self,
            y0: np.ndarray,
            ax: plt.Axes,
            t: np.ndarray) -> None:

        res = self.run(y0, t)
        ax.plot(t, res)
        ax.set_title(
            f'y0={y0}, R={self.R}, Q=[{self.Q[0]},{self.Q[1]}]',
            fontsize=8
        )
        ax.legend(['$x_1$', '$x_2$'])

t = np.linspace(0, 5, 100)
experiments = [[]]*len(Rs)
for i, r in enumerate(Rs):
    for q in Qs:
        experiments[i].append(
            Experiment2(A, B, q, r, t)
        )

fig, axes = plt.subplots(len(Rs), len(Qs))
for i in range(len(Rs)):
    for j in range(len(Qs)):
        experiments[i][j].plot(
            np.array([5, -5]),
            axes[i][j],
            np.linspace(0, 2, 100)
        )

fig.set_size_inches(15, 20)
plt.show()
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.