Научно-технологический университет «Сириус»
Научный центр информационных технологий и искусственного интеллекта
Направление «Математическая робототехника»

Отчёт

по заданию №3

Дисциплина: численные методы нелинейной и выпуклой оптимизации

Тема: управление

Выполнил

Студент группы М01МР-24 __________________А. С. Кондратьев

Преподаватель

Профессор, к.ф.-м.н. _______________________С. В. Гусев

Сириус

2025

Задача 1

Задана система второго порядка с передаточной функцией

$$G(s) = \frac{b_0}{s^2 + a_1 s + a_0}$$

Построим представление системы в виде

$$\dot{x} = Ax + Bu$$

$$y = Cx$$

выбрав в качестве вектора состояния

$$x(t) = \left[\int_0^t y(\tau)d\tau \quad y(t) \quad \dot{y}(t)\right]^T$$

тогда

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -a_0 & -a_1 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ b_0 \end{bmatrix}, C = [0 \quad 1 \quad 0]$$

Запишем ПИД-регулятор как линейный регулятор вида

$$u = Kx$$

Выход системы при подобранных вручную коэффициентах ПИД-регулятора ($K = [0 \quad 10 \quad 5]$) представлен на рисунке 1.

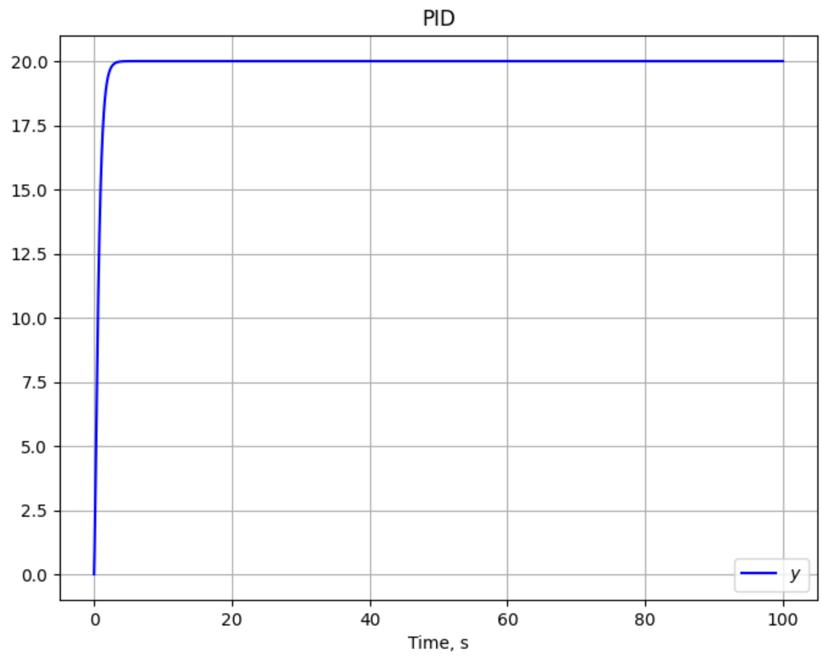

Рисунок 1 – Выход системы (ручной подбор коэффициентов)

Задача 2

Напишем программу для нахождения стабилизирующего решения уравнения Риккати

$$I + A^T P + PA - \frac{1}{r} PBB^T P = 0$$

с помощью программных пакетов при значениях $r = 0.01, 1, 100$. Значения коэффициентов ПИД-регулятора рассчитываются по формуле

$$K = \frac{1}{r} B^T P$$

Для поиска решения зададим значения коэффициентов для системы второго порядка

$$a_0 = -2, a_1 = 1, b_0 = 1$$

В качестве референсного сигнала выберем $ref = 20$ (константа). Результаты моделирования представлены на рисунке 2.

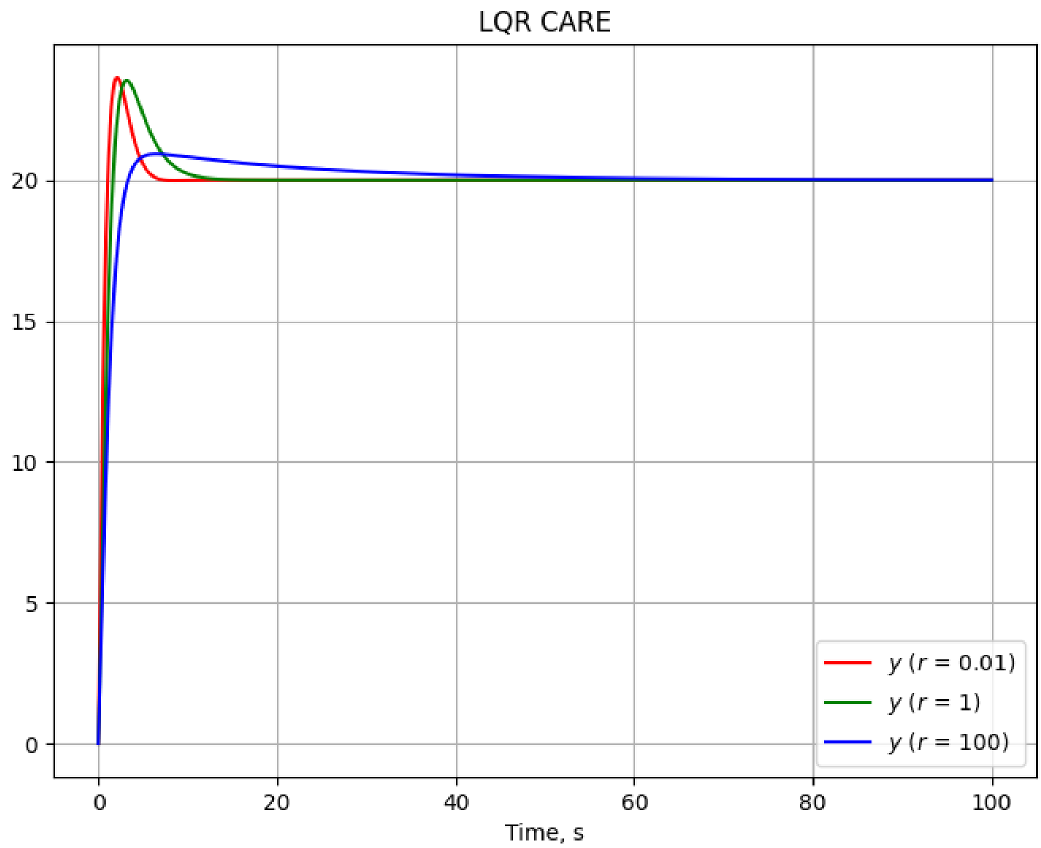

Рисунок 2 – Стабилизирующие решения уравнения Риккати

Найдем стабилизирующее решение, решая задачу полуопределенного программирования

$$\max \operatorname{tr}(P)$$

при

$$L(P) = \begin{bmatrix} PA + A^T P + Q & PB \\ B^T P & R \end{bmatrix} \geq 0$$

где $Q = I$, $R = 1$.

Разница между значениями выходного сигнала, полученного при использовании программных пакетов и при решении задачи SDP представлена на рисунке 3 и имеет порядок 10^{-7}.

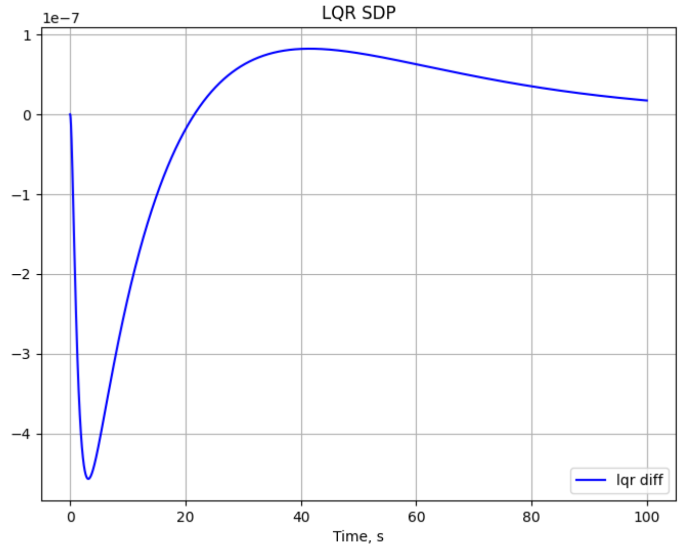

Рисунок 3 – Сравнение выходных сигналов, полученных разными методами

Задача 3

Используя решение неравенства Ляпунова, построим регулятор, минимизирующий число обусловленности матрицы функции Ляпунова и обеспечивающий неравенства $\operatorname{Re} p_i \leq -\gamma$, где p_i – полюсы передаточной функции замкнутой системы. Результаты работы регулятора представлены на рисунке 4.

Неравенство Ляпунова представлено формулой

$$AQ + BKQ + QA^T + QK^T B_T \leq -2\gamma Q$$

Задача оптимизации

$$\min \sigma$$

при

$$AQ + BKQ + QA^T + QK^TB_T \leq -2\gamma Q,$$
$$I \leq Q \leq \sigma I$$

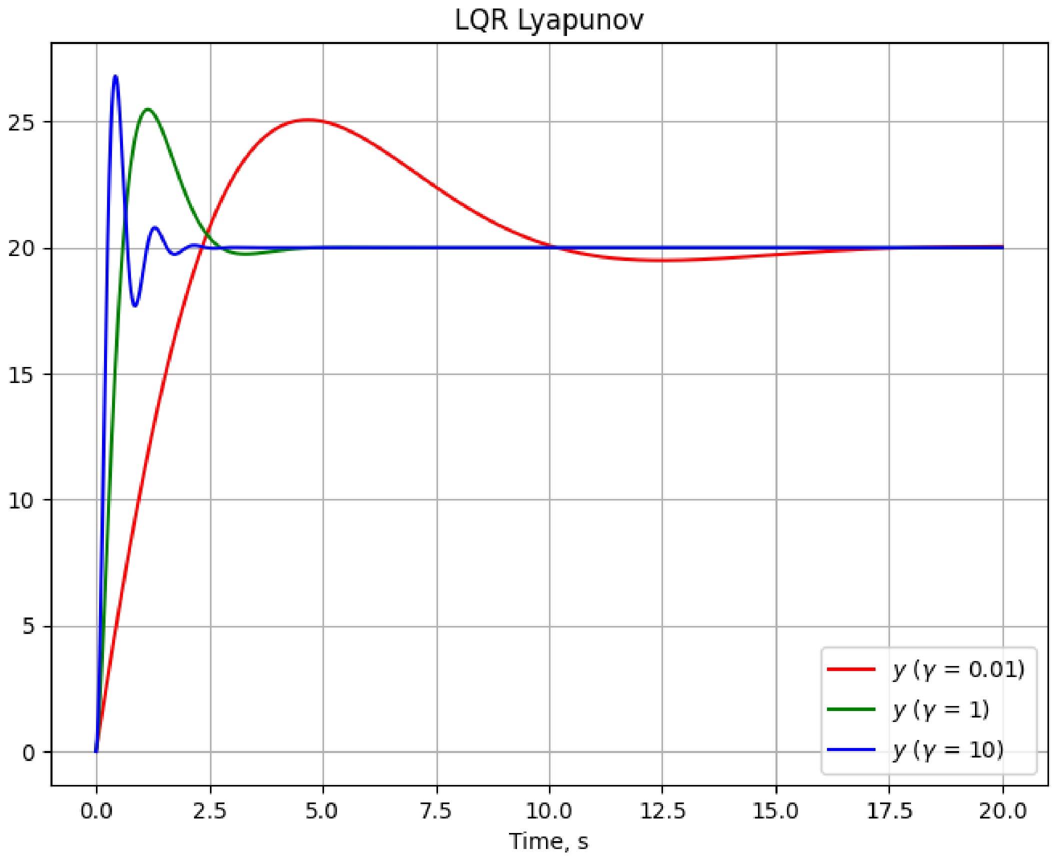

Рисунок 4 – Регулятор на основе неравенства Ляпунова

Заключение

В случае использования стабилизирующего решения уравнения Риккати выбор параметра r влияет на скорость сходимости выхода системы к референсному сигналу: при меньших значениях r сходимость быстрее. При этом поиск такого решения можно свести к задаче полуопределенного программирования.

При использовании неравенства Ляпунова параметр γ так же влияет на скорость сходимость выходного сигнала к желаемому значению: при меньших значениях γ система сходится медленнее. Кроме того, во время сходимости наблюдаются частые колебания вокруг референсного значения, в то время как при использовании стабилизирующего решения уравнения Риккати такое явление не наблюдалось.

Приложение №1

```
import cvxpy as cp
import matplotlib.pyplot as plt
import numpy as np
import scipy

from scipy.optimize import minimize

t1 = 100.0
dt = 0.001

a0 = -2
a1 = 1
b0 = 1

def sim(K, x0, A, B, C):
    yarr = []
    tarr = []

    t = 0
    x = x0.copy()

    while t < t1:
        u = -K @ x

        dx = A @ x + B @ u

        x = x + dx * dt

        y = C @ x

        yarr.append(y)
        tarr.append(t)

        t += dt

    yarr = np.array(yarr)
    tarr = np.array(tarr)

    return yarr, tarr

def main():
```

```
A = np.array([[0, 1, 0],
              [0, 0, 1],
              [0, -a0, -a1]])

B = np.array([[0, 0, b0]]).T

C = np.array([[0, 1, 0]])

x0 = np.array([[0, -20, 10]]).T

# LQR scipy
_, ax0 = plt.subplots(1, 1, figsize=(8, 6), sharex=True)
_, ax1 = plt.subplots(1, 1, figsize=(8, 6), sharex=True)
_, ax2 = plt.subplots(1, 1, figsize=(8, 6), sharex=True)
_, ax3 = plt.subplots(1, 1, figsize=(8, 6), sharex=True)
ax0.set_xlabel('Time, s')
ax1.set_xlabel('Time, s')
ax2.set_xlabel('Time, s')
ax3.set_xlabel('Time, s')
ax0.grid(True)
ax1.grid(True)
ax2.grid(True)
ax3.grid(True)
ax0.set_title('PID')
ax1.set_title('LQR CARE')
ax2.set_title('LQR SDP')
ax3.set_title('LQR Lyapunov')

K_pid = np.array([[0, 10, 5]])
yarr_pid, tarr_pid = sim(K_pid, x0, A, B, C)
ax0.plot(tarr_pid, yarr_pid[:, 0, 0], f'b-', label=f'$y$')

Q = np.eye(A.shape[0])
r = (0.01, 1, 100)
R1 = np.eye(B.shape[1]) * r[0]
R2 = np.eye(B.shape[1]) * r[1]
R3 = np.eye(B.shape[1]) * r[2]
for rv, R, c in zip(r, (R1, R2, R3), ('r', 'g', 'b')):
    P = scipy.linalg.solve_continuous_are(A, B, Q, R)
    K_lqr = np.linalg.inv(R) @ B.T @ P
    yarr_care, tarr_care = sim(K_lqr, x0, A, B, C)
   ax1.plot(tarr_care, yarr_care[:, 0, 0], f'{c}-', label=f'$y$ ($r$ = {rv})')
```

```
    # LQR SDP cvxpy
    R = np.eye(B.shape[1]) * 100
    P = cp.Variable(A.shape, symmetric=True)
    L = cp.vstack([cp.hstack([P @ A + A.T @ P + Q, P @ B]),
                   cp.hstack([B.T @ P, R])])
    problem = cp.Problem(cp.Maximize(cp.trace(P)), [L >> 0])
    problem.solve(verbose=False)
    K_sdp = np.linalg.inv(R) @ B.T @ P.value
    yarr_sdp, tarr_sdp = sim(K_sdp, x0, A, B, C)
    yarr_sdp = np.array(yarr_sdp)
    tarr_sdp = np.array(tarr_sdp)
    yarr_care = np.array(yarr_care)
    tarr_care = np.array(tarr_care)
    ax2.plot(tarr_sdp, yarr_sdp[:, 0, 0] - yarr_care[:, 0, 0], f'b-', label=f'lqr
diff')

    # Lyapunov ineq.
    Gamma = [1]

    def objective(x):
        sigma = x[0]
        return sigma

    def unpack_variables(x):
        sigma = x[0]
        Q = x[1:10].reshape(A.shape[0], A.shape[0])
        L = x[10:].reshape(B.shape[1], Q.shape[0])
        return sigma, Q, L

    def constraint1(x):
        sigma, Q, _ = unpack_variables(x)
        return -np.linalg.eigvals(sigma * np.eye(Q.shape[0]) - Q)

    def constraint2(x):
        _, Q, _ = unpack_variables(x)
        return -np.linalg.eigvals(Q - np.eye(Q.shape[0]))

    sigma0 = 10
    Q0 = np.eye(A.shape[0]).flatten()
    L0 = np.zeros((B.shape[1], Q0.shape[0] // A.shape[0])).flatten()
    state0 = np.concatenate(([sigma0], Q0, L0))
```

```
    for gamma, color in zip(Gamma, ('b')):
        def constraint3(x):
            _, Q, L = unpack_variables(x)
            M = A @ Q + B @ L + Q @ A.T + L.T @ B.T + 2 * gamma * Q
            return -np.linalg.eigvals(M)

        constraints = [
            {'type': 'ineq', 'fun': constraint1},
            {'type': 'ineq', 'fun': constraint2},
            {'type': 'ineq', 'fun': constraint3},
        ]

              result   =   minimize(objective,   state0,   constraints=constraints,
method='SLSQP')
        _, Q, L = unpack_variables(result.x)
        K_lyap = (L @ np.linalg.inv(Q))

        yarr_lyap, tarr_lyap = sim(K_lyap, x0, A, B, C)
           ax3.plot(tarr_lyap[:20000],  yarr_lyap[:20000,  0,  0],  f'{color}-',
label=f'$y$ ($\gamma$ = {gamma})')

    ax0.legend(loc='lower right')
    ax1.legend(loc='lower right')
    ax2.legend(loc='lower right')
    ax3.legend(loc='lower right')
    plt.show()

if __name__ == '__main__':
    main()
```