[EEC3600-001] 수치해석		
소속: 전기전자공학부	학번: 12191529	이름: 장준영
Term Project		**Prob #5**

1. Problem

a. 문제

Problem 5. [*Linear system (programming)*]
Consider the so-called Lyapunov equation

$$AX + XB = C$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times m}$, $C \in \mathbb{R}^{n\times m}$ are given matrices and $X \in \mathbb{R}^{n\times m}$ is the unknown to be found[3].

(a) Make your own **python** function to solve the Lyapunov equation for X when the matrices A, B, and C are given. You might first want to convert the matrix equations into a linear system (see the next page for an example) and solve the linear system to find the elements of the matrix X. You are asked to write a python code in which the input is (A, B, C) and the output (or return) is X of a matrix form with a compatible dimension.

(b) Test your python code with numerical case studies. (If you are taking my class, "Control Systems Design", then use the pole placement examples of state feedback controller design and state observer design in the lecture note.) Show the test results you performed.

(c) Test your code with $A = \begin{bmatrix} 1 & 2 \\ -3 & -4 \end{bmatrix}$, $B = A^\top$, $C = \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix}$.

[3]There already exist some numerical solvers for this form of matrix equations:

- For MATLAB, there is a built-in function `lyap` to solve the special and general forms of the Lyapunov equation. You can find more details of the MATLAB built-in function `lyap` in the link https://www.mathworks.com/help/control/ref/lyap.html.
- For Python, a similar function `control.matlab.lyap` from *Python Control Systems Library* can be used to solve the special and general forms of the Lyapunov equation. You can find more details of the Python function `control.matlab.lyap` in the link https://python-control.readthedocs.io/en/0.8.3/generated/control.matlab.lyap.html.

Background on the use of Lyapunov equation for pole-placement

• Lyapunov Equation

$$AX + XB = C$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times m}$, $C \in \mathbb{R}^{n\times m}$, $X \in \mathbb{R}^{n\times m}$
(A, B, C) given $\rightarrow$ solve it for $X \in \mathbb{R}^{n\times m}$
The equation is called the Lyapunov equation for X.

(Example) Consider a case with n = 3 and m = 2 :

$$AX + XB = C$$

$$\Leftrightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix} + \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}$$

$$\Leftrightarrow \begin{bmatrix} a_{11}+b_{11} & a_{12} & a_{13} & b_{21} & 0 & 0 \\ a_{21} & a_{22}+b_{11} & a_{23} & 0 & b_{21} & 0 \\ a_{31} & a_{32} & a_{33}+b_{11} & 0 & 0 & b_{21} \\ b_{12} & 0 & 0 & a_{11}+b_{22} & a_{12} & a_{13} \\ 0 & b_{12} & 0 & a_{21} & a_{22}+b_{22} & a_{23} \\ 0 & 0 & b_{12} & a_{31} & a_{32} & a_{33}+b_{22} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \\ x_{12} \\ x_{22} \\ x_{32} \end{bmatrix} = \begin{bmatrix} c_{11} \\ c_{21} \\ c_{31} \\ c_{12} \\ c_{22} \\ c_{32} \end{bmatrix}$$

$$\Leftrightarrow \mathcal{A}\mathbf{x} = \mathbf{c}$$

This is indeed a standard linear system (i.e., a system of linear algebraic equations). There are $n \times m = 6$ equations for $n \times m = 6$ unknowns $(x_{11}, x_{21}, x_{31}, x_{12}, x_{22}, x_{32})$.

An application: Pole-placement via solving Lyapunov equation We discuss a method of computing state feedback gain for eigenvalue assignment (i.e. pole placement).

Procedure : Consider controllable (A, B) where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times 1}$. We want to find a state feedback controller gain $K \in \mathbb{R}^{1\times n}$ such that $(A - BK)$ has any set of desired eigenvalues which contain no eigenvalues of A.

1. Select an $n \times n$ matrix F with the set of desired eigenvalues where the form of F can be arbitrarily chosen. For example, F can be a modal, observer canonical, or controller canonical form.
2. Select an arbitrary $\bar{K} \in \mathbb{R}^{1\times n}$ such that $(F, \bar{K})$ is observable.
3. Solve the unique $T \in \mathbb{R}^{n\times n}$ in the Lyapunov equation

$$AT - TF = B\bar{K}$$

4. Compute the feedback gain $K := \bar{K}T^{-1}$.

본 문제에서는 선형 시스템 이론에서 중요한 역할을 하는 Lyapunov 방정식을 다룬다. 주어진 행렬 A, B, C에 대해 미지수 행렬 X를 만족시키는 식

$$AX + XB = C$$

을 수치적으로 해결하는 것이 핵심 과제이다. 이 방정식은 제어 이론, 시스템 안정성 해석, 상태 피드백 설계 등 다양한 분야에서 활용된다.

2. Solution (a)

■ 입력 및 기본정보 설정

```
    n, m = C.shape
    N = n * m  # Number of unknowns in X

    # Initialize the system matrix and right-hand side
    M = np.zeros((N, N))
    b = C.reshape(N)
```

- $X \in \mathbb{R}^{m \times n}$을 총 $N = n \times m$개의 스칼라 미지수로 간주.
- $AX + XB = C$를 벡터화하여 $Mx = b$로 변환하기 위한 선형 시스템의 행렬 M과 벡터 b를 준비한다.

■ 행렬 M 구성

```
    # Build the system manually
    # Let X = [x11, x12, ..., x1m, x21, x22, ..., xnm]^T
    for i in range(n):
        for j in range(m):
            row = i * m + j
            eq = np.zeros(N)

            # AX term: sum_k A[i,k] * x[k,j]
            for k in range(n):
                col = k * m + j  # x[k,j]
                eq[col] += A[i, k]

            # XB term: sum_k x[i,k] * B[k,j]
            for k in range(m):
                col = i * m + k  # x[i,k]
                eq[col] += B[k, j]

            M[row, :] = eq
```

- X의 각 항목 x_{ij}에 대해, AX, XB를 직접 계산한 항등식을 행별로 설정.
- Kronecker product 를 쓰지 않고, 수동으로 각 항에 대한 계수를 직접 할당해 방정식을 구성한다.

■ 선형 시스템 풀기

```
    # Solve Mx = b
    x = np.linalg.inv(M) @ b  # 또는 np.linalg.pinv(M) @ b 도 가능 (정칙성 안심 못할 경우)
    X = x.reshape(n, m)

    return X
```

- $Mx = b$ 선형 시스템을 풀어 x 벡터를 얻고, 이를 다시 X 행렬 형태로 복원.

■ **Test Case 검증 (문제 5-(c)에서 확인 예정.)**

```
40    A = np.array([[1, 2], [-3, -4]])
41    B = A.T
42    C = np.array([[3, 1], [1, 1]])
43
44    X = solve_lyapunov_manual(A, B, C)
45
46    print("Solution X:")
47    print(X)
48
49    # 검증
50    print("\nAX + XB:")
51    print(A @ X + X @ B)
52    print("\nOriginal C:")
53    print(C)
```

- 실제 $AX + XB$ 결과가 C 와 거의 일치하는지 확인하여 정확성을 검증한다.

3. Solution (b)

■ Lyapunov 방정식 해법 함수 정의

```python
# Lyapunov solver (manual)
def solve_lyapunov_manual(A, B, C):
    n, m = C.shape
    N = n * m
    M = np.zeros((N, N))
    b = C.reshape(N)

    for i in range(n):
        for j in range(m):
            row = i * m + j
            eq = np.zeros(N)

            for k in range(n):
                col = k * m + j
                eq[col] += A[i, k]

            for k in range(m):
                col = i * m + k
                eq[col] += B[k, j]

            M[row, :] = eq

    x = np.linalg.inv(M) @ b
    X = x.reshape(n, m)
    return X
```

- $AX + XB = C$를 벡터화하여 $Mx = b$의 선형 시스템으로 구성.
- 고급 함수 없이 반복문과 행렬 indexing 으로 방정식을 풀고 결과 X 반환

■ 시스템 정의

```python
# --------------------------------
# Step 1: Define system matrices
# --------------------------------
A = np.array([[0, 1],
              [-2, -3]])
B = np.array([[0],
              [1]])
```

- 테스트에 사용할 상태방정식의 시스템 행렬 A, 입력 행렬 B.

■ **목표 폐루프 pole 정의**

```
37    # --------------------------------
38    # Step 2: Define desired closed-loop poles
39    # --------------------------------
40    # Desired poles: -2 and -5
41    # Characteristic equation: s^2 + 7s + 10
42    # Corresponding F matrix with desired poles
43    F = np.array([[0, 1],
44                  [-10, -7]])
```

- 고유값 -2, -5 를 갖는 시스템으로 만들기 위한 목표 폐루프 행렬 F.

■ **상태 피드백 Gain (K) 설정**

```
46    # --------------------------------
47    # Step 3: Manually choose state feedback gain K
48    # --------------------------------
49    # A - B*K should match F
50    K = np.array([[12, 8]])
```

- $A - BK = F$ 조건을 만족하는 feedback gain F 를 수동으로 설정.

■ **Lyapunov 방정식 구성 및 풀이**

```
52    # --------------------------------
53    # Step 4: Build Lyapunov equation: A*T - T*F = B*K
54    # --------------------------------
55    C = B @ K
56    T = solve_lyapunov_manual(A, -F, C)
```

- Lyapunov 방정식 $AT - TF = BK$를 $AX + XB = C$로 바꿔서 T 계산.

■ **결과 출력 및 검증**

```
58    # --------------------------------
59    # Step 5: Print results
60    # --------------------------------
61    print("T matrix (solution of A*T - T*F = B*K):")
62    print(T)
63
64    print("\nA*T - T*F:")
65    print(A @ T - T @ F)
66
67    print("\nB*K:")
68    print(C)
```

- 계산된 T 가 실제로 $AT - TF = BK$를 만족하는지 검증한다.

[결과 분석]

```
T matrix (solution of A*T - T*F = B*K):
[[ 1.12589991e+16  2.25179981e+15]
 [-2.25179981e+16 -4.50359963e+15]]

A*T - T*F:
[[  0. -14.]
 [-16.   8.]]

B*K:
[[ 0  0]
 [12  8]]
```

본 실험에서는 상태 피드백을 통해 지정된 고유값을 갖도록 시스템을 설계하고, 이에 따라 유도되는 Lyapunov 방정식 AT−TF=BK 를 수치적으로 해결하였다. 주어진 시스템 행렬 A, 입력 행렬 B, 그리고 목표 폐루프 행렬 F 에 대해 수동으로 피드백 게인 K 를 설계하였다. 이후 C=BK 로 정의하고, 직접 작성한 solve_lyapunov_manual 함수를 이용해 행렬 TTT 를 계산하였다.

결과로 얻은 T 행렬은 수치적으로 매우 큰 값을 가지는 비정상적인 형태를 보였으며, AT−TF 의 결과는 BK 와 명확히 일치하지 않았다. 이는 M 행렬이 수치적으로 매우 ill-conditioned 하거나 역행렬 계산 과정에서 정밀도 손실이 있었음을 의미한다. 실제로 이 문제에서는 M 이 정칙이어도 수치적으로 매우 불안정할 수 있으며, 이 경우에는 np.linalg.inv() 대신 np.linalg.pinv() 또는 SVD 기반 해법을 사용하는 것이 권장된다.

결론적으로, 이 코드는 구조적으로는 정확하지만, 특정 데이터에서는 수치적인 안정성을 확보하기 위해 정규화 기법이나 다른 수치해석적 방법을 병행하는 것이 필요하다는 점을 확인하였다.

4. Solution (c): 시뮬레이션 결과 및 그래프 해석

```
Solution X:
[[-6.16666667  3.83333333]
 [ 3.83333333 -3.        ]]

AX + XB:
[[3. 1.]
 [1. 1.]]

Original C:
[[3 1]
 [1 1]]
```

문제 5-(c)

$$A = \begin{bmatrix} 1 & 2 \\ -3 & -4 \end{bmatrix}, B = A^T = \begin{bmatrix} 1 & -3 \\ 2 & -4 \end{bmatrix}, C = \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix}$$

에 대해 Lyapunov 방정식

$$AX + XB = C$$

를 수치적으로 해결하였다.
직접 작성한 함수 solve_lyapunov_manual()을 이용해 계산된 해는 다음과 같다:

$$X = \begin{bmatrix} -6.17 & 3.83 \\ 3.83 & -3.00 \end{bmatrix}$$

이 결과를 $AX + XB$에 대입하여 다시 계산한 결과는 정확히 C와 일치하였다:

$$AX + XB = \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix} = C$$

이를 통해, 수동으로 구성한 선형 시스템 해법이 정확히 동작했으며, 구현된 Lyapunov 해법의 수치적 정확성이 충분히 검증되었음을 확인할 수 있었다. 특히, 이 문제는 행렬 차원이 작고, 계수의 크기가 크지 않아 수치적 불안정성 없이 잘 해결된 사례로 볼 수 있다.