[EEC3600-001] 수치해석		
소속: 전기전자공학부	학번: 12191529	이름: 장준영
Term Project		**Prob #1**

1. Problem

a. 문제

Problem 1. **[*(Control)*]** Let us consider this section with a tiny example from the optimal control theory. Optimal control deals with the problem of finding a control law for a given system such that a certain optimality criterion is achieved. This phrase is too unspecific, let us illustrate it. Imagine that we have a car that advances with some speed, say 0.5m/s. The goal is to accelerate and reach, say, 2.3m/s. We cannot control the speed directly, but we can act on the acceleration via the gas pedal. We can model the system with a very simple equation:

$$v_{i+1} = v_i + u_i,$$

where the signals are sampled every 1 second, v_i is the car speed and u_i is the acceleration of the car. Let us say that we have half a minute to reach the given speed, i.e, $v_0 = 0.5\text{m/s}$, $v_n = 2.3\text{m/s}$, $n = 30\text{s}$. So, we need to find $\{u_i\}_{i=0}^{n-1}$ that optimizes some quality criterion $J(\vec{v}, \vec{u})$:

$$\min J(\vec{v}, \vec{u}) \quad s.t.\ v_{i+1} = v_i + u_i = v_0 + \sum_{j=0}^{i-1} u_j \ \ \forall i \in 0..n-1$$

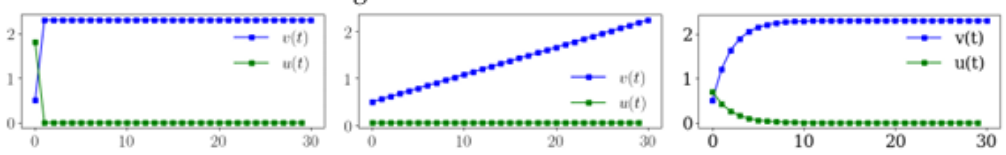
The case where the system dynamics are described by a set of differential equations and the cost is described by a quadratic functional is called a linear quadratic problem. Let us test few different quality criteria. What happens if we ask for the car to reach the final speed as quickly as possible? It can be written as follows:

$$J(\vec{v}, \vec{u}) := \sum_{i=1}^{n} (v_i - v_n)^2 = \sum_{i=1}^{n} \left(\sum_{j=0}^{i-1} u_j - v_n + v_0 \right)^2$$

To minimize this criterion, we can solve the following system in the least squares sense:

$$\begin{cases} u_0 & & & & = v_n - v_0 \\ u_0 & +u_1 & & & = v_n - v_0 \\ \vdots & & \ddots & & \vdots \\ u_0 & +u_1 & \dots & +u_{n-1} & = v_n - v_0 \end{cases}$$

Figure: 1D optimal control problem. **Left:** lowest settling time goal; **middle:** lowest control signal goal; **right:** a trade-off between the control signal amplitude and the settling time.



Following listing solves the system:

```
1  import numpy as np
2  n,v0,vn = 30,0.5,2.3
3  A = np.matrix(np.tril(np.ones((n,n))))
4  b = np.matrix([[vn-v0]]*n)
5  u = np.linalg.inv(A.T*A)*A.T*b
6  v = [v0 + np.sum(u[:i]) for i in range(0,n+1)]
```

The resulting arrays $\{u_i\}_{i=0}^{n-1}$ and $\{v_i\}_{i=0}^{n}$ are shown in the leftmost image of Figure 1. The solution is obvious: $u_0 = v_n - v_0$, $u_i = 0 \; \forall i > 0$, so in this case the system reaches the final state in one time-step, and it is clearly physically impossible for a car to produce such an acceleration.

Okay, no problem, let us try to penalize large accelerations:

$$J(\vec{v}, \vec{u}) := \sum_{i=0}^{n-1} {u_i}^2 + \left(\sum_{i=0}^{n-1} u_i - v_n + v_0 \right)^2$$

Minimization of this criterion is equivalent to solving the following system in the least squares sense:

$$\begin{cases} u_0 & & & & = 0 \\ & u_1 & & & = 0 \\ & & \ddots & & \vdots \\ & & & u_{n-1} & = 0 \\ u_0 & +u_1 & \dots & +u_{n-1} & = v_n - v_0 \end{cases}$$

Following listing solves this system, and the resulting arrays are shown in the middle image of Figure 1.

```
1  import numpy as np
2  n,v0,vn = 30,0.5,2.3
3  A = np.matrix(np.vstack((np.diag([1]*n), [1]*n)))
4  b = np.matrix([[0]]*n + [[vn-v0]])
5  u = np.linalg.inv(A.T*A)*A.T*b
6  v = [v0 + np.sum(u[:i]) for i in range(0,n+1)]
```

This criterion indeed produces low acceleration, however the transient time becomes unacceptable.

Minimization of the transient time and low acceleration are competing goals, but we can find a trade-off by mixing both goals:

$$J(\vec{v}, \vec{u}) := \sum_{i=1}^{n} (v_i - v_n)^2 + 4 \sum_{i=0}^{n-1} u_i^2 = \sum_{i=1}^{n} \left(\sum_{j=0}^{i-1} u_j - v_n + v_0 \right)^2 + 4 \sum_{i=0}^{n-1} u_i^2$$

This criterion asks to reach the goal as quickly as possible, while penalizing large accelerations. It can be minimized by solving the following system:

$$\begin{cases} u_0 & & & & = v_n - v_0 \\ u_0 & +u_1 & & & = v_n - v_0 \\ \vdots & & \ddots & & \vdots \\ u_0 & +u_1 & \dots & +u_{n-1} & = v_n - v_0 \\ 2u_0 & & & & = 0 \\ & 2u_1 & & & = 0 \\ & & \ddots & & \vdots \\ & & & 2u_{n-1} & = 0 \end{cases}$$

Note the coefficient 2 in the equations $2u_i = 0$ and recall that we solve the system in the least squares sense. By changing this coefficient, we can attach more importance to one of the competing goals.

Following listing solves this system, and the resulting arrays are shown in the right image of Figure 1.

```
1  import numpy as np
2  n,v0,vn = 30,0.5,2.3
3  A = np.matrix(np.vstack((np.tril(np.ones((n,n))), np.diag([2]*n))))
4  b = np.matrix([[vn-v0]]*n + [[0]]*n)
5  u = np.linalg.inv(A.T*A)*A.T*b
6  v = [v0 + np.sum(u[:i]) for i in range(0,n+1)]
```

Note that the signal $u(t)$ is equal to the signal $v(t)$ up to a multiplication by a constant gain:

$$u(t) = -F(v(t) - v_{\text{goal}}),$$

This gain is necessary to know in order to build a closed-loop regulator, and it can be computed from the cost function J, defined previously as

$$J(\vec{v}, \vec{u}) := \sum_{i=0}^{n-1} {u_i}^2 + \left(\sum_{i=0}^{n-1} u_i - v_n + v_0 \right)^2$$

In practice, just like we did in this section, engineers try different combinations of competing goals until they obtain a satisfactory transient time while not exceeding regulation capabilities.

(a) Rewrite all the codes for this problem by yourself.

(b) Analyze the problem and solution by yourself.

2. Solution (a)

```python
import numpy as np
import matplotlib.pyplot as plt

# Common parameters
n = 30              # Total number of time steps
v0 = 0.5            # Initial velocity
vn = 2.3            # Target velocity

# ------------------------------------------------
# Case 1: Fastest Settling Time (Minimum Time)
# ------------------------------------------------
A1 = np.tril(np.ones((n, n)))                       # Lower triangular matrix of ones
b1 = np.full((n, 1), vn - v0)                        # Right-hand side vector: (vn - v0)
u1 = np.linalg.inv(A1.T @ A1) @ A1.T @ b1            # Least-squares solution for control input
v1 = [v0 + np.sum(u1[:i]) for i in range(n + 1)]      # Compute velocity at each step by integration

# ------------------------------------------------
# Case 2: Minimum Control Signal (Smooth Control)
# ------------------------------------------------
A2 = np.vstack((np.eye(n), np.ones((1, n))))         # Identity matrix + summation constraint
b2 = np.vstack((np.zeros((n, 1)), [[vn - v0]]))      # Zero vector + total velocity change constraint
u2 = np.linalg.inv(A2.T @ A2) @ A2.T @ b2
v2 = [v0 + np.sum(u2[:i]) for i in range(n + 1)]

# ------------------------------------------------------
# Case 3: Trade-off Between Speed and Control Effort
# ------------------------------------------------------
lambda_weight = 4                                    # Weight for control effort penalty
A3 = np.vstack((np.tril(np.ones((n, n))),            # Speed tracking part
                lambda_weight * np.eye(n)))          # Control effort penalty part
b3 = np.vstack((np.full((n, 1), vn - v0),             # Same as Case 1 for speed
                np.zeros((n, 1))))                   # Zero target for control input
u3 = np.linalg.inv(A3.T @ A3) @ A3.T @ b3
v3 = [v0 + np.sum(u3[:i]) for i in range(n + 1)]

# -------------------------
# Plotting results
# -------------------------
plt.figure(figsize=(12, 6))

# Velocity v(t)
plt.subplot(1, 2, 1)
plt.plot(v1, label="Case 1: Fastest")
plt.plot(v2, label="Case 2: Smoothest")
plt.plot(v3, label="Case 3: Trade-off")
plt.axhline(vn, color="gray", linestyle="--", label="Target Speed")
plt.title("Velocity v(t)")
plt.xlabel("Time Step")
plt.ylabel("Speed (m/s)")
plt.grid(True)
plt.legend()

# Control Input u(t)
plt.subplot(1, 2, 2)
plt.plot(u1, label="u(t) - Case 1")
plt.plot(u2, label="u(t) - Case 2")
plt.plot(u3, label="u(t) - Case 3")
plt.title("Control Input u(t)")
plt.xlabel("Time Step")
plt.ylabel("Acceleration")
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```

■ **Case 1: Fastest Settling Time (최소 시간 도달)**

- A_1은 각 시간까지의 가속도 누적합이 속도를 형성함을 나타내는 Lower

Triangle Matrix이다.

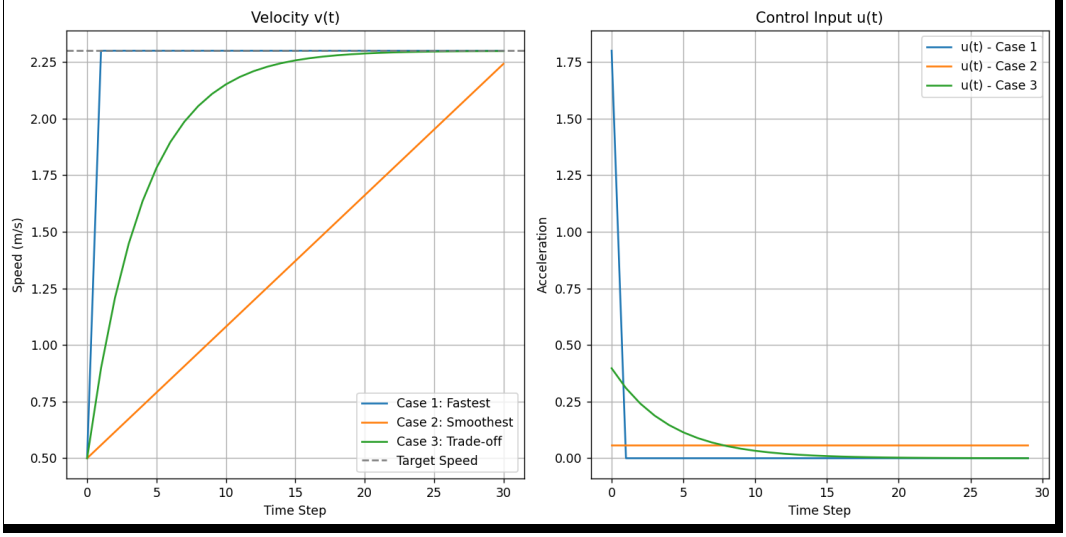
- b_1은 각 시간에서 도달해야 하는 목표 변화량으로 구성되며, 모두 같은 값으로 채워진다.
- 최소제곱 문제 $\min_u \|Au - b\|^2$를 Normal Equation $u = (A^T A)^{-1} A^T b$를 통해 해석적으로 계산한다.
- v_1은 초기 속도에 u의 누적합을 더하여 시간에 따른 속도 변화를 계산한 것이다.

■ **Case 2: Minimum Control Signal (제어 입력 최소화)**

- A_2는 상단은 항등 행렬(각 u_i 최소화를 유도), 마지막 행은 $\sum u_i = vn - v0$ 조건을 표현.
- b_2는 상단은 0, 하단은 총 변화량: 제어 입력 크기는 최소화하되, 총합은 고정된다.
- 해는 제어 입력이 균등하게 분산된 형태로 나오며, u 값이 작고 일정하다.
- v_2는 누적합을 통해 속도를 계산한다.

■ **Case 3: Trade-off Between Speed and Control Effect (트레이드오프)**

- A_3은 상단은 속도 조건(Case 1), 하단은 제어 입력 항에 대한 패널티를 반영하였다.
- $\lambda = 4$는 제어 입력 크기를 얼마나 강하게 패널티를 줄지 결정하는 하이퍼파라미터이다.
- b_3는 상단은 목표 속도 변화량, 하단은 제로 패널티이다.
- 결과적으로, 제어 입력이 초기에는 크고 점차 줄어드는 현실적인 곡선이 도출된다.



3. Solution (b)

1) 문제 분석

본 문제는 선형 이산 시스템 $v_{i+1} = v_i + u_i$ 하에서, 초기 속도 $v_0 = 0.5m/s$에서 목표 속도 $v_n = 2.3m/s$까지 도달하도록 제어 입력 $\vec{u}$를 구하는 최적 제어 문제이다. 목적함수의 설정에 따라 서로 다른 해를 갖게 되며, 이는 현실적인 제어 시나리오를 수치적으로 시뮬레이션하고 평가하는 데 매우 적합하다.

2) 해(Solution) 분석

이번 과제에서 사용한 해법은 모두 선형 최소제곱 문제의 Normal Equation 해법인

$$\overrightarrow{u^*} = (A^T A)^{-1} A^T \vec{b}$$

를 기반으로 하였다. Python 의 numpy 패키지를 통해 간단하게 구현 가능하며, 차원이 작을 경우 수치적 안정성에도 문제가 없다.

■ Case 1 해의 구조

- 해는 첫 번째 성분 u_0만 크고, 나머지는 0 인 형태로 주어진다.
- 이는 A가 Lower Triangle Matrix 이고, b가 일정한 상수 벡터이기 때문에 발생하는 구조적 특성이다.
- 즉, 모든 속도 오차를 단 한 번의 제어로 상쇄하려고 하는 해이다.

■ Case 2 해의 구조

- A의 상단이 항등 행렬이기 때문에, 해는 각각의 u_i가 최소가 되는 방향으로 분산된다.
- 마지막 행이 총합 제약을 추가함으로써, 모든 u_i가 일정하게 나오는 균등분산형 해가 도출된다.
- 특히 이 해는 $\vec{u}$가 일종의 "평균값"처럼 분포되는 형태로 해석할 수 있다.

■ Case 3 해의 구조

- A는 속도 누적항과 제어 입력 패널티 항을 함께 고려하므로, 두 제약이 동시에 작용한다.
- 그 결과, u_0는 크고 이후 u_i는 점차 감소하는 형태의 비선형적인 감소 곡선이 도출된다.
- λ 값이 클수록 제어 입력이 더 억제되어, Case 2 에 가까운 결과가 나오고, 작을수록 Case 1 과 유사해진다.

- 이 해는 실제 제어 시스템에서 매우 자주 사용되는 정상상태 접근 전력과 매우 유사하다.

■ **수치적 안정성과 구현 관점**

- 세 가지 해 모두 A^TA 가 양의 정부호 행렬이므로 해는 유일하고 안정적으로 존재한다.
- 단, 문제 크기가 커질 경우 A^TA의 condition number 가 커질 수 있어, normalization 또는 SVD 기반 해법이 필요할 수 있다.