

항공방산SW산학연구프로젝트

Research Project

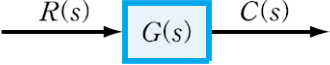
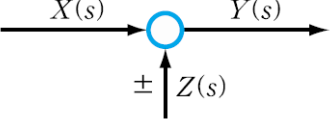
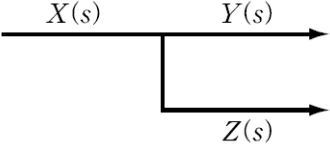
int**Ro**-Lab.

청주대학교 무인항공기학과
김상현 교수 (www.intro-lab.com)

전달함수

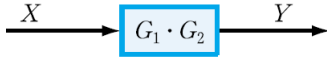
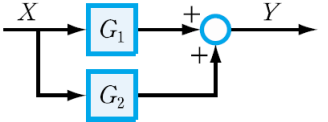
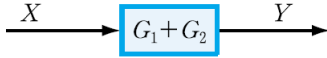
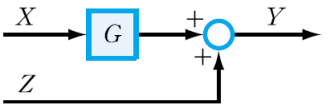
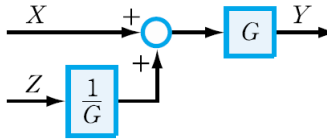
◆ 블록선도

● 표기법

종류	기호	연산
신호		화살표 방향으로만 전달
전달요소		$C(s) = G(s) R(s)$
가산점		$Y(s) = X(s) \pm Z(s)$
인출점		$X(s) = Y(s) = Z(s)$

전달함수

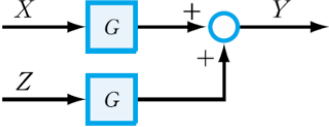
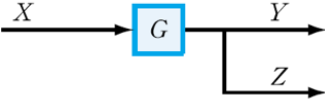
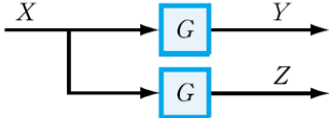
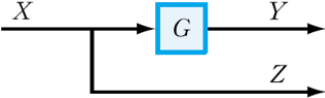
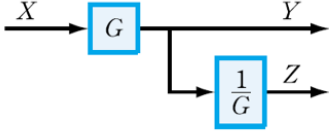
◆ 블록선도 ● 등가변환

번호	블록연산	블록선도	등가 블록선도
1	교환		
2	직렬결합		
3	병렬결합		
4	가산점을 앞으로 이동		

전달함수

◆ 블록선도

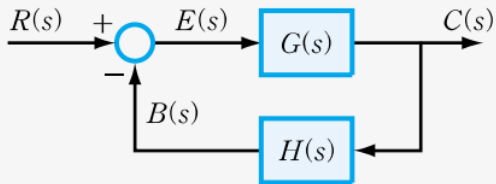
● 등가변환

번호	블록연산	블록선도	등가 블록선도
5	가산점을 뒤로 이동		
6	인출점을 앞으로 이동		
7	인출점을 뒤로 이동		

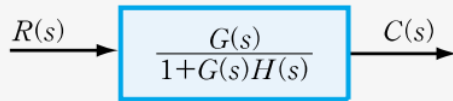
◆ 블록선도

- 등가변환

[그림 3-9(a)]와 같은 폐루프 제어시스템의 등가변환이 [그림 3-9(b)]와 같음을 보여라.



(a) 폐루프 제어시스템의 블록선도



(b) 등가변환 블록선도

[그림 3-9] [예제 3-5]의 등가변환

◆ 블록선도

● 등가변환

$$E(s) = R(s) - B(s)$$

$$C(s) = G(s) E(s)$$

$$B(s) = H(s) C(s)$$

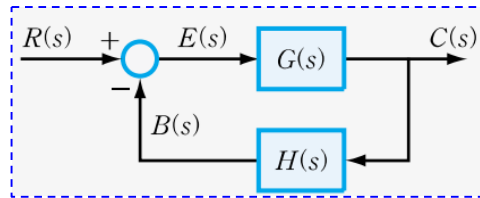
$$E(s) = R(s) - H(s) C(s)$$

$$C(s) = G(s) E(s) = G(s) \{R(s) - H(s) C(s)\}$$

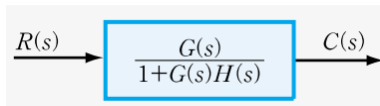
$$C(s) + G(s) H(s) C(s) = G(s) R(s)$$

$$\{1 + G(s) H(s)\} C(s) = G(s) R(s)$$

$$C(s) = \frac{G(s)}{1 + G(s) H(s)} R(s)$$

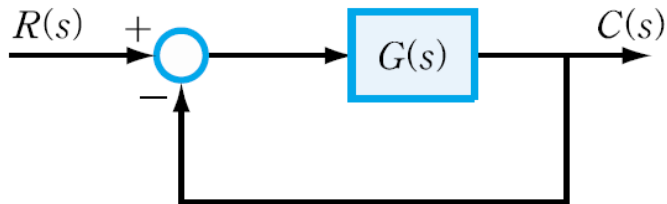


$$M(s) = \frac{C(s)}{R(s)} = \frac{G(s)}{1 + G(s) H(s)}$$



◆ 블록선도

● 단위귀환 제어시스템

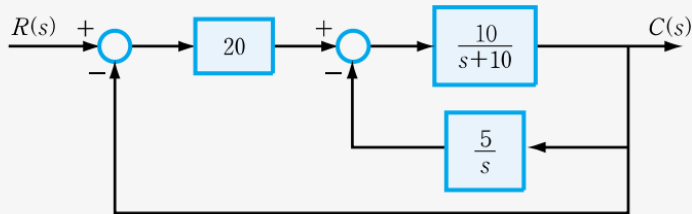


$$M(s) = \frac{G(s)}{1 + G(s)}$$

전달함수

◆ 블록선도 ● 등가변환

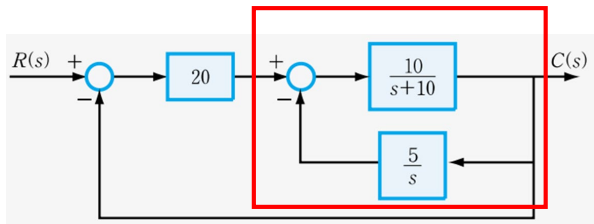
[그림 3-11]과 같이 복잡한 귀환 제어시스템의 전체 폐루프 전달함수를 구하라.



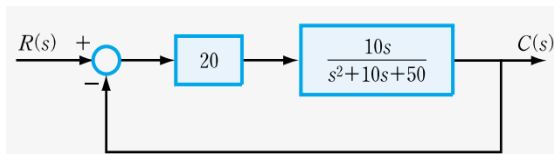
[그림 3-11] [예제 3-6]의 귀환 제어시스템

전달함수

◆ 블록선도 ● 등가변환



$$\frac{\frac{10}{s+10}}{1 + \frac{10}{s+10} \times \frac{5}{s}} = \frac{10s}{s^2 + 10s + 50}$$



[그림 3-12] 간략화된 [그림 3-11]의 귀환 제어시스템

$$\begin{aligned} M(s) &= \frac{\frac{200s}{s^2 + 10s + 50}}{1 + \frac{200s}{s^2 + 10s + 50}} \\ &= \frac{200s}{s^2 + 210s + 50} \end{aligned}$$

◆ 블록선도 파이썬 코드

● Series 연결

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

H1 = control.tf([3], [4, 1])
H2 = control.tf([5], [2, 1])

H = control.series(H1, H2)

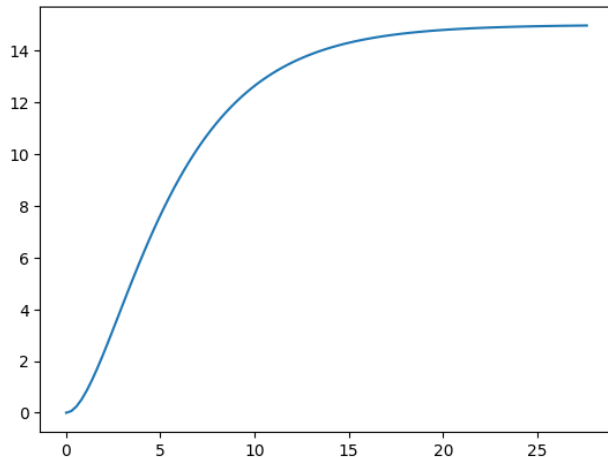
print(H)

t, y = control.step_response(H)
plt.plot(t,y)
```

<TransferFunction>: sys[8]
Inputs (1): ['u[0]']
Outputs (1): ['y[0]']

H1, H2 출력도 확인

$$\frac{15}{8s^2 + 6s + 1}$$



전달함수

◆ 블록선도 파이썬 코드

● Parallel 연결

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

```
H1 = control.tf([3], [4, 1])
```

```
H2 = control.tf([5], [2, 1])
```

```
H = control.parallel(H1, H2)
```

```
print(H)
```

```
t, y = control.step_response(H)
```

```
plt.plot(t,y)
```

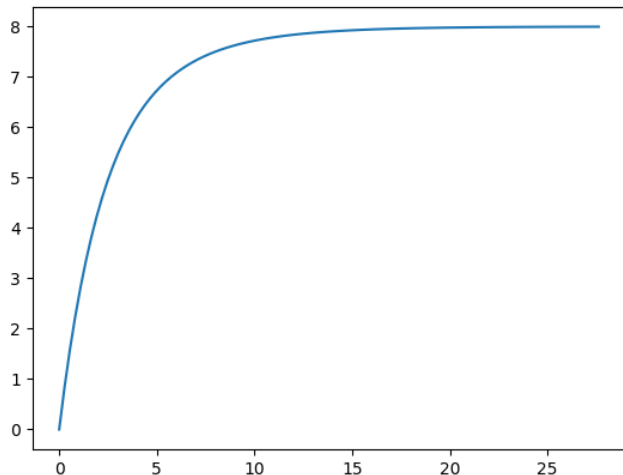
<TransferFunction>: sys[13]

Inputs (1): ['u[0]']

Outputs (1): ['y[0]']

H1, H2 출력도 확인

$$\frac{26s + 8}{8s^2 + 6s + 1}$$



◆ 블록선도 파이썬 코드

● Feedback 연결

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

```
H1 = control.tf([3], [4, 1])
```

```
H2 = control.tf([5], [2, 1])
```

```
H = control.feedback(H1, H2)
```

```
print(H)
```

```
t, y = control.step_response(H)
```

```
plt.plot(t,y)
```

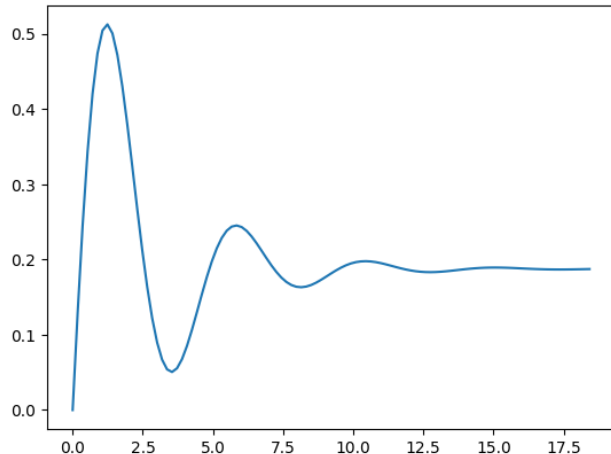
<TransferFunction>: sys[18]

Inputs (1): ['u[0]']

Outputs (1): ['y[0]']

H1, H2 출력도 확인

$$\frac{6s + 3}{8s^2 + 6s + 16}$$



◆ 상태방정식

- 복잡한 동적 시스템(Dynamic System)의 입력과 출력, 내부 상태 간의 관계를 1차 미분 행렬 방정식으로 정식화한 현대 제어 이론의 핵심 도구

◆ 상태방정식의 등장

- 1950년대 이전에 사용된 고전 제어 이론은 라플라스 변환(Laplace Transform)을 이용한 전달함수 모델을 주로 사용.
- 전달함수를 이용한 자동제어 기법의 단점
 - 초기 조건 무시: 라플라스 변환을 적용하려면 모든 초기 조건이 0이라고 가정. 그러나 실제 드론과 같이 움직이는 시스템에서는 이동하는 도중에 자동제어 필요하게 됨. 즉, 초기 조건이 0이 아닌 상황에서의 제어가 필수적임.
 - SISO(단일입력 단일출력) 시스템에만 국한: 입력과 출력이 각각 하나인 단순한 시스템에는 유리하지만, 드론과 같이 수많은 모터와 센서가 있는 MIMO(다중입력 다중출력) 형태의 시스템에서는 전달함수로 표현하기가 극도로 복잡해짐.
- 이를 해결하기 위해 루돌프 칼만 등이 중심이 되어 시스템의 내부 물리량을 행렬로 묶어 다루는 상태방정식 중심의 현대 제어이론을 정립

◆ 상태방정식의 정의

- 선형 시불변 시스템의 연속시간 상태 공간 모델로 정의됨.
- 상태 공간(State Space): 시간에 따라 변하는 시스템 내부의 상태를 변수로 잡고, 이를 좌표축으로 설정하는 n 차원의 수학적 공간을 의미. 예를 들어, 상태 변수를 위치와 속도로 정의하면 상태 공간의 x 축은 위치 좌표가 되고, y 축은 속도 좌표가 됨.
- 상태 변수(State Variable): 특정 시점에서 시스템의 물리적 상태를 완전하게 정의하는데 필요한 변수의 집합. 예를 들어, 달리는 자동차는 상태 변수로 위치와 속도를 사용할 수 있음.

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t)\end{aligned}$$

$x(t)$: 상태 변수

$y(t)$: 출력 변수

$u(t)$: 입력 변수

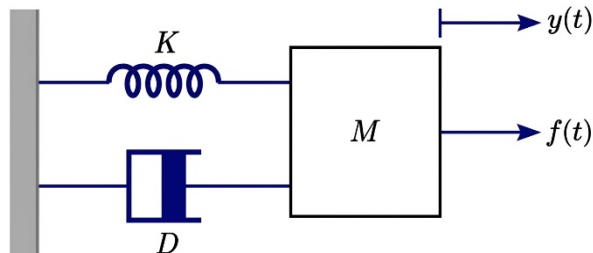
C : 출력 행렬

A : 시스템 행렬

B : 입력 행렬

◆ 상태방정식 모델링 예시

● 동적 시스템 형상



● 미분방정식

$$M \frac{d^2}{dt^2} y(t) + D \frac{d}{dt} y(t) + K y(t) = f(t)$$

◆ 상태방정식 모델링 예시

● 상태 변수 정의

$$y(t) = x_1(t) \quad \leftarrow \text{위치}$$

$$\dot{y}(t) = x_2(t) = \dot{x}_1(t) \quad \leftarrow \text{속도}$$



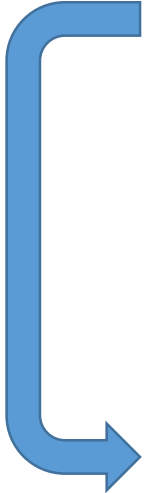
$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \quad \leftarrow \text{상태 변수 벡터}$$



$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} \quad \leftarrow \text{상태 변수 벡터 미분}$$

◆ 상태방정식 모델링 예시

- 미분방정식을 상태 변수로 재정의


$$M \frac{d^2}{dt^2} y(t) + D \frac{d}{dt} y(t) + Ky(t) = f(t)$$

$$y(t) = x_1(t)$$

$$\frac{d}{dt} y(t) = \dot{y}(t) = \dot{x}_1(t) = x_2(t)$$

$$\frac{d^2}{dt^2} y(t) = \ddot{y}(t) = \dot{x}_2(t) = -\frac{K}{M} x_1(t) - \frac{D}{M} x_2(t) + \frac{1}{M} f(t)$$


$$M \dot{x}_2(t) + Dx_2(t) + Kx_1(t) = f(t)$$

◆ 상태방정식 모델링 예시

- 상태 변수 벡터를 이용한 상태방정식

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{K}{M} & -\frac{D}{M} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M} \end{bmatrix} f(t)$$

$$y(t) = [1 \quad 0] \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$



$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}f(t)$$

$$y(t) = \mathbf{C}\mathbf{x}(t)$$

◆ 상태방정식과 전달함수 관계

- 상태방정식을 라플라스 변환

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t)$$



$$sX(s) = AX(s) + BU(s)$$

$$Y(s) = CX(s)$$



$$X(s) = (sI - A)^{-1}BU(s)$$



$$(sI - A)X(s) = BU(s)$$

$$Y(s) = CX(s)$$



$$Y(s) = C(sI - A)^{-1}BU(s)$$

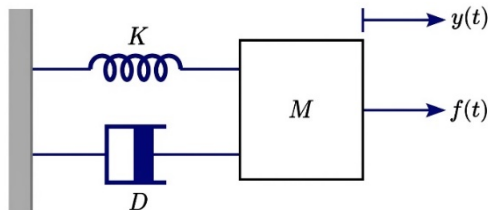


$$G(s) = \frac{Y(s)}{U(s)} = C(sI - A)^{-1}B$$

◆ 상태방정식 → 전달함수 예제

● 동적 시스템 정의

➢ $M = 1, K=3, D=2$



$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -3 & -2 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} f(t)$$



$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 1 \\ -3 & -2 \end{bmatrix}$$

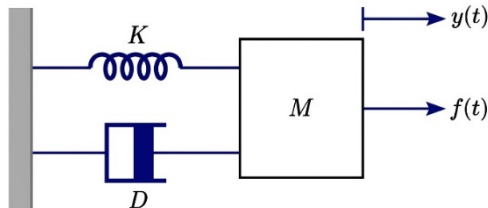
$$B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

◆ 상태방정식 → 전달함수 예제

● 동적 시스템 정의

➢ $M = 1, K=3, D=2$

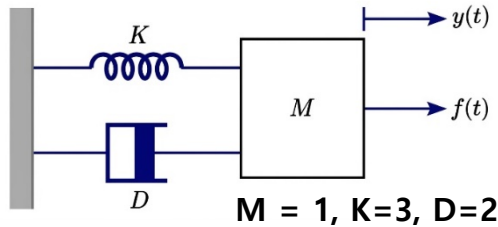
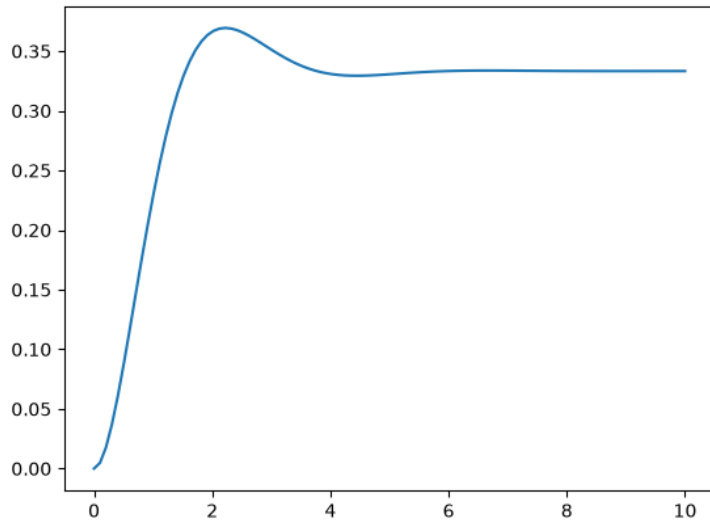


$$A = \begin{bmatrix} 0 & 1 \\ -3 & -2 \end{bmatrix}$$
$$B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$
$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$G(s) = \frac{Y(s)}{U(s)} = C(sI - A)^{-1}B = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} s & -1 \\ 3 & s+2 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$
$$= \frac{1}{s^2 + 2s + 3}$$

◆ 전달함수를 이용한 위치 변화 시뮬레이션

● 파이썬 코드



```
1 import control as ct
2 import matplotlib.pyplot as plt
3
4 num = [1]
5 den = [1, 2, 3]
6
7 sys = ct.tf(num, den)
8
9 t, y = ct.step_response(sys, T=10)
10
11 plt.plot(t, y)
12 plt.show()
```

◆ 전달함수를 이용한 위치 변화 시뮬레이션

● 파이썬 코드

```
import control as ct
import matplotlib.pyplot as plt

num = [1]
den = [1, 2, 3]

sys = ct.tf(num, den)

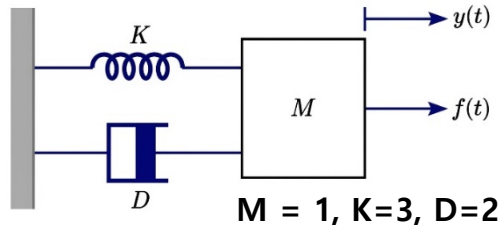
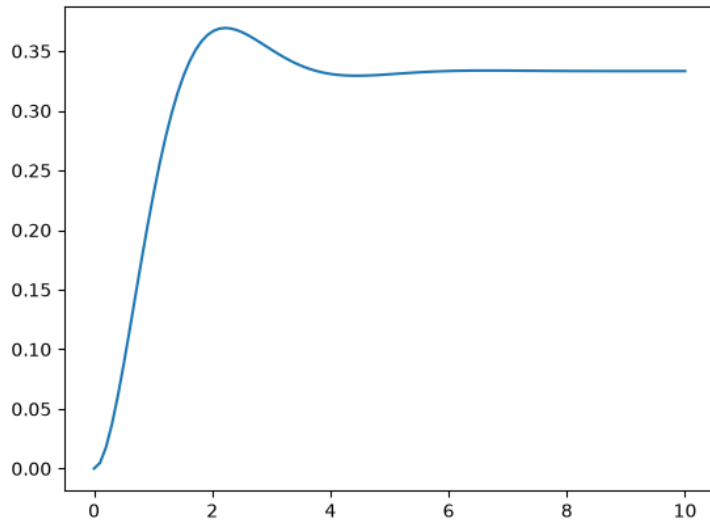
t, y = ct.step_response(sys, T=10)

plt.plot(t, y)
plt.show()
```

상태방정식

◆ 상태방정식을 이용한 위치 변화 시뮬레이션

● 파이썬 코드

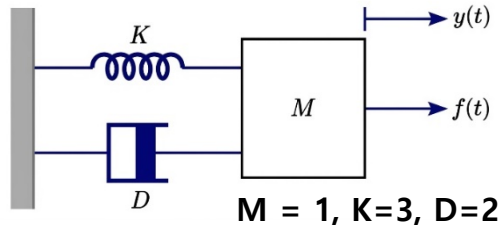
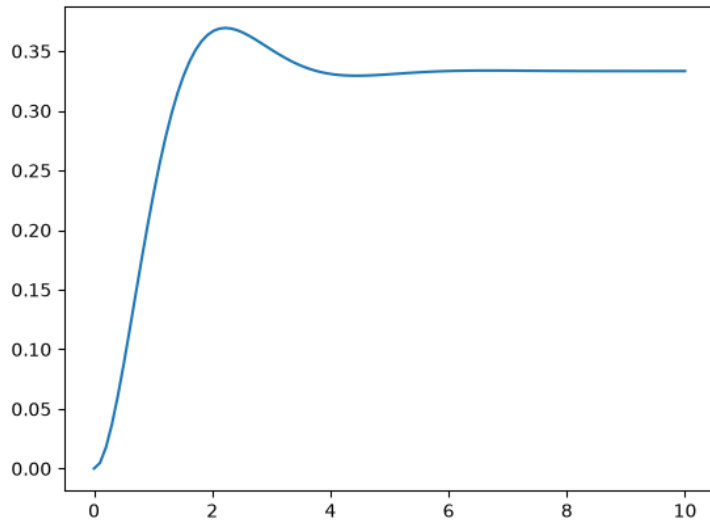


```
1 import control as ct
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 A = np.array([[ 0, 1],
6               [-3, -2]])
7 B = np.array([[0],
8               [1]])
9 C = np.array([1, 0])
10
11 sys = ct.ss(A, B, C, 0)
12
13 t, y = ct.step_response(sys, T=10)
14
15 plt.plot(t, y)
16 plt.show()
```

상태방정식

◆ 상태방정식을 이용한 위치 변화 시뮬레이션

● 파이썬 코드



```
1 import control as ct
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 A = np.array([[ 0, 1],
6               [-3, -2]])
7 B = np.array([[0],
8               [1]])
9 C = np.array([1, 0])
10
11 sys = ct.ss(A, B, C, 0)
12
13 t, y = ct.step_response(sys, T=10)
14
15 plt.plot(t, y)
16 plt.show()
```

◆ 상태방정식을 이용한 위치 변화 시뮬레이션

● 파이썬 코드

```
import control as ct
import matplotlib.pyplot as plt
import numpy as np

A = np.array([[ 0, 1],
              [-3, -2]])
B = np.array([[0],
              [1]])
C = np.array([1, 0])

sys = ct.ss(A, B, C, 0)

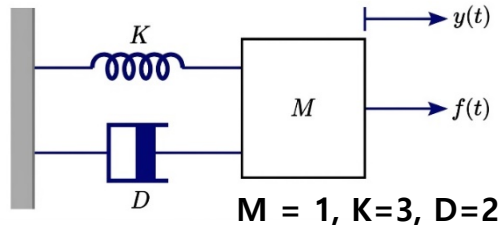
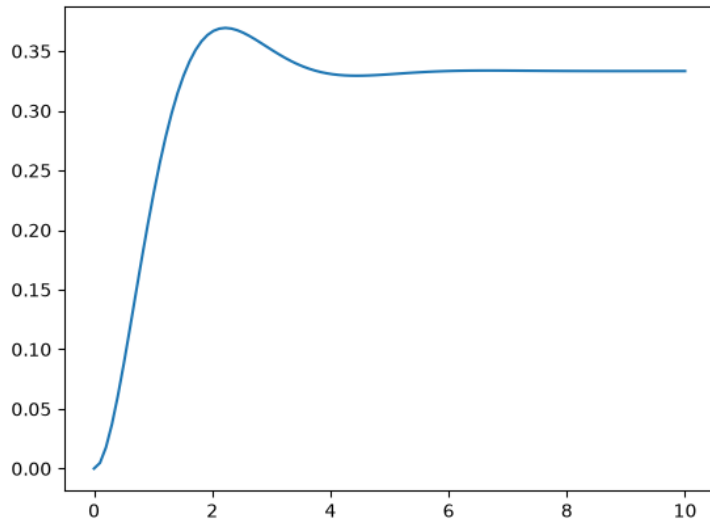
t, y = ct.step_response(sys, T=10)

plt.plot(t, y)
plt.show()
```

상태방정식

◆ 상태방정식을 이용한 위치 변화 시뮬레이션

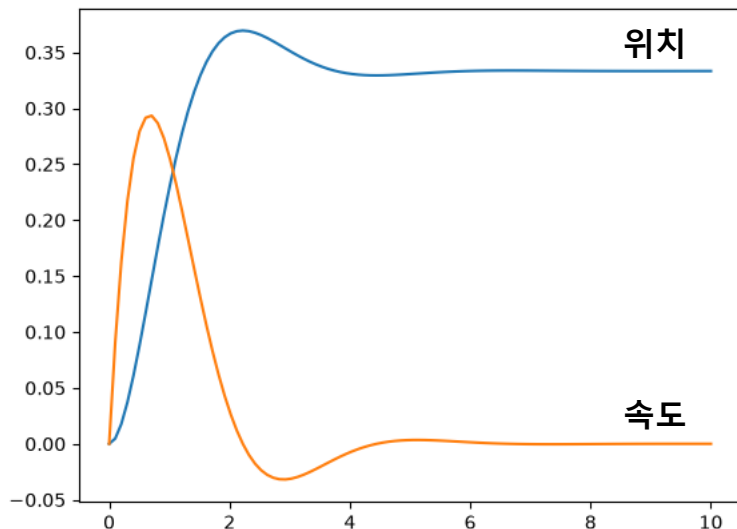
● 파이썬 코드



```
1 import control as ct
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 A = np.array([[ 0, 1],
6               [-3, -2]])
7 B = np.array([[0],
8               [1]])
9 C = np.array([1, 0])
10
11 sys = ct.ss(A, B, C, 0)
12
13 t, y = ct.step_response(sys, T=10)
14
15 plt.plot(t, y)
16 plt.show()
```

◆ 상태방정식을 이용한 위치 변화 시뮬레이션

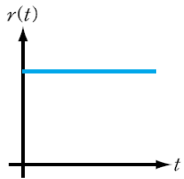
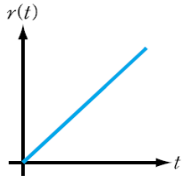
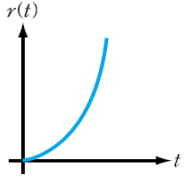
- 파이썬 코드: 위치, 속도 모두 출력 가능 → MIMO 구현 가능



```
1 import control as ct
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 A = np.array([[ 0, 1],
6               [-3, -2]])
7 B = np.array([[0],
8               [1]])
9 C = np.array([[1, 0],
10              [0, 1]])
11
12 sys = ct.ss(A, B, C, 0)
13
14 t, y = ct.step_response(sys, T=10)
15 print(y)
16 plt.plot(t, y[0,0,:], t, y[1,0,:])
17 plt.show()
```

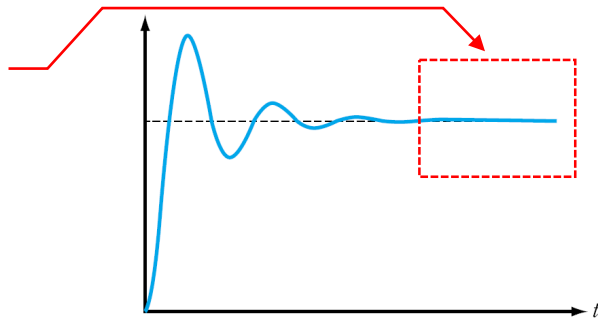
단위계단응답

◆ 입력 신호 종류

종류	신호 파형	시간함수	라플라스 변환
계단함수		$u_s(t)$	$\frac{1}{s}$
램프함수		$t u_s(t)$	$\frac{1}{s^2}$
포물선함수		$\frac{1}{2} t^2 u_s(t)$	$\frac{1}{s^3}$

◆ 정상상태(Steady-State)

- 정의: 출력이 일정한 값을 유지하는 상태



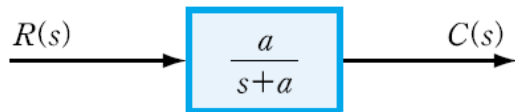
◆ 1차 지연시스템(1st-Order Lag System)

- 정의: 입력 신호의 변화에 대해 출력이 1차 미분방정식으로 표현되는 시스템
- 시정수: 계단 입력을 가했을 때, 출력이 최종값(정상상태 값)의 63.2%에 도달하는 데 걸리는 시간
- 상승시간: 시스템 응답이 최종값의 10%에서 90%까지 올라가는 데 걸리는 시간
 - 상승시간은 시정수의 약 2.2배.
- 정정시간: 출력이 최종 목표값의 허용 오차 범위 이내로 완전히 들어와 안정되는 데 걸리는 시간
 - 일반적으로 출력이 최종값의 98%에 도달하는 시간. 시정수의 약 3.9배.

단위계단응답

◆ 1차 지연시스템의 단위계단응답

● 1차 지연시스템



● 단위계단응답

$$C(s) = G(s) R(s) = \frac{a}{s+a} \frac{1}{s}$$

◆ 1차 지연시스템의 단위계단응답

● 시간에 대한 함수

$$\begin{aligned}
 c(t) &= \mathcal{L}^{-1} \left[\frac{a}{s+a} \frac{1}{s} \right] \\
 &= \mathcal{L}^{-1} \left[\frac{1}{s} - \frac{1}{s+a} \right] \\
 &= 1 - e^{-at}
 \end{aligned}$$

[표 2-2] 기본 함수의 라플라스 변환

$f(t)$	$F(s)$
c	$\frac{c}{s}$
t	$\frac{1}{s^2}$
t^2	$\frac{2}{s^3}$
t^n	$\frac{n!}{s^{n+1}}$
e^{-at}	$\frac{1}{s+a}$
$\sin \omega t$	$\frac{\omega}{s^2 + \omega^2}$
$\cos \omega t$	$\frac{s}{s^2 + \omega^2}$

단위계단응답

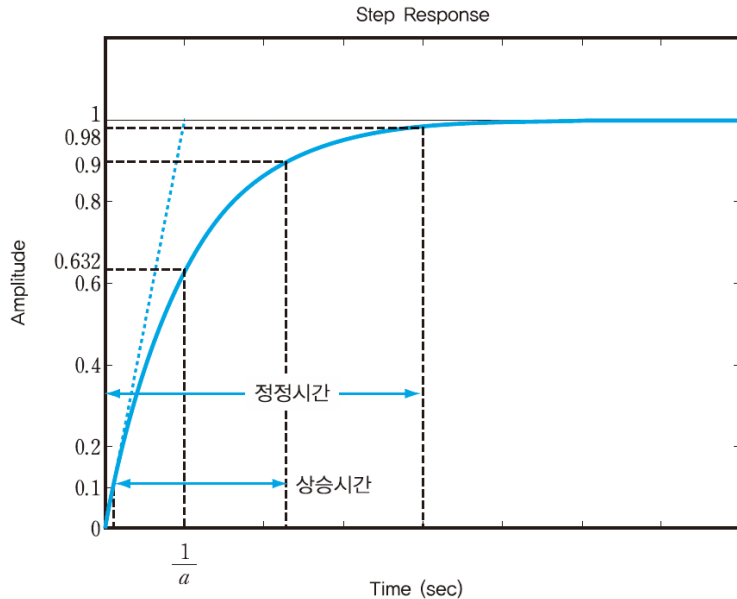
◆ 1차 지연시스템의 단위계단응답

$$c(t) = 1 - e^{-at}$$

■ 시정수 $t = \frac{1}{a}$

■ 상승시간 $T_r = 2.2 \times \frac{1}{a}$

■ 정정시간 $T_s \doteq 3.9 \times \frac{1}{a}$



◆ 1차 지연시스템의 단위계단응답

● 상승시간 유도

상승시간 : 출력 $c(t)$ 가 최종값의 10[%]가 되는 시간을 t_1 , 출력값의 90[%]가 되는 시간을 t_2 라고 하면

$$c(t_1) = 1 - e^{-at_1} = 0.1$$

$$e^{-at_1} = 0.9$$

$$-at_1 = \ln 0.9 \doteq -0.1$$

$$t_1 \doteq \frac{0.1}{a}$$

$$c(t_2) = 1 - e^{-at_2} = 0.9$$

$$e^{-at_2} = 0.1$$

$$-at_2 = \ln 0.1 \doteq -2.3$$

$$t_2 \doteq \frac{2.3}{a}$$

이므로, 상승시간은

$$T_r = t_2 - t_1 = \frac{2.3}{a} - \frac{0.1}{a} = 2.2 \times \frac{1}{a}$$

이다. 따라서 상승시간은 시정수의 약 2.2배다.

◆ 1차 지연시스템의 단위계단응답

● 정정시간 유도

정정시간 : 출력 $c(t)$ 가 최종값의 98[%]가 되는 시간을 t_s 라고 하면

$$c(t_s) = 1 - e^{-at_s} = 0.98$$

$$e^{-at_s} = 0.02$$

$$-at_s = \ln 0.02 \doteq -3.9$$

$$t_s \doteq 3.9 \times \frac{1}{a}$$

이다. 따라서 정정시간 T_s 는 시정수의 약 4배다.

◆ 1차 지연시스템의 단위계단응답

● 파이썬 코딩

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

num1 = np.array([1])
den1 = np.array([4, 1])

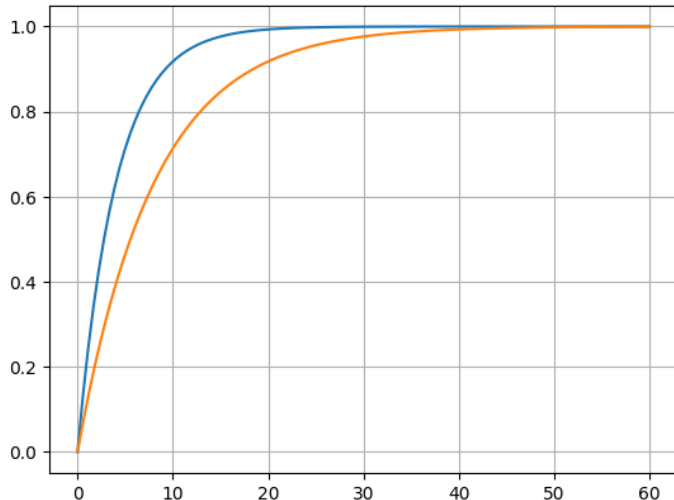
H1 = control.tf(num1, den1)
t1, y1 = control.step_response(H1, T=60)

num2 = np.array([1])
den2 = np.array([8, 1])

H2 = control.tf(num2, den2)
t2, y2 = control.step_response(H2, T=60)

plt.plot(t1, y1, t2, y2)
plt.grid()
```

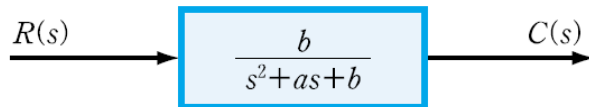
$$a = \frac{1}{4} = 0.25$$



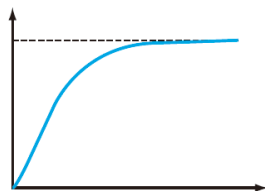
단위계단응답

◆ 2차 지연시스템의 단위계단응답

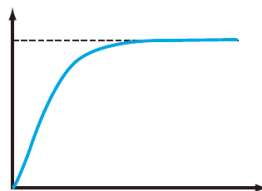
● 2차 지연시스템



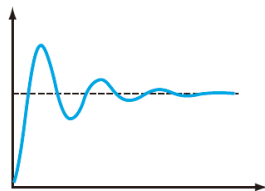
● 단위계단응답 종류



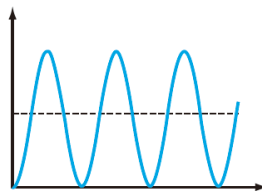
(a) 과제동응답



(b) 임계제동응답



(c) 부족제동응답



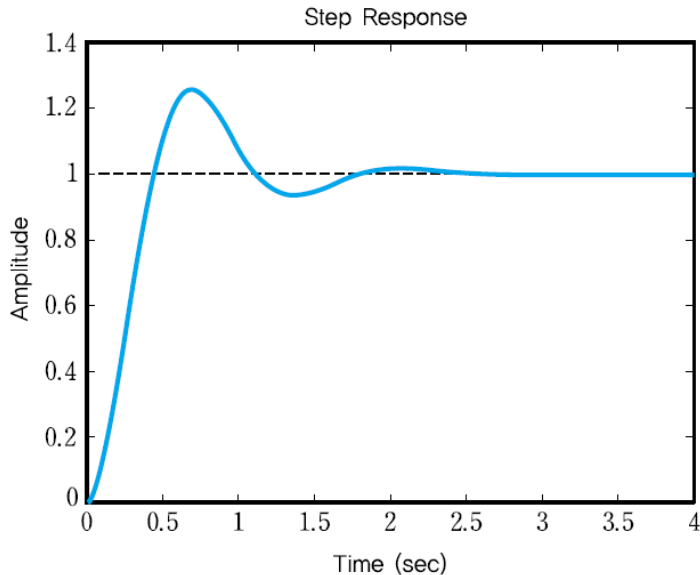
(d) 무제동응답

단위계단응답

◆ 2차 지연시스템의 단위계단응답

● 백분율 오버슈트

$$\text{백분율 오버슈트} = \frac{\text{최대 오버슈트}}{\text{최종값}} \times 100$$



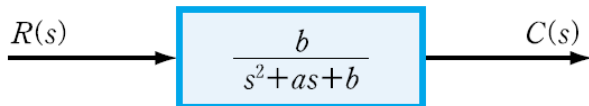
단위계단응답

◆ 2차 지연시스템의 단위계단응답

● 특성방정식의 표준형

$$s^2 + 2\zeta\omega_n s + \omega_n^2 = 0$$

● 예시



$$s^2 + as + b = 0 \quad \longrightarrow \quad s^2 + 2\zeta\omega_n s + \omega_n^2 = 0$$

$$s^2 + 4s + 16 = 0 \quad \longrightarrow \quad s^2 + 2 \times 0.5 \times 4 s + 4^2 = 0$$

$$\text{감쇠비 } \zeta = 0.5$$

$$\text{고유주파수 } \omega_n = 4$$

단위계단응답

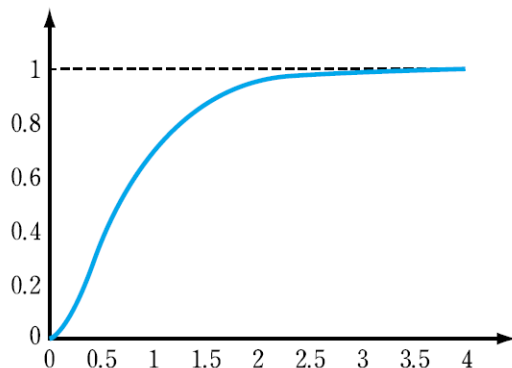
◆ 2차 지연시스템의 단위계단응답

● 과제동응답 예시

$$R(s) \rightarrow \boxed{\frac{6}{s^2+5s+6}} \rightarrow C(s) \Rightarrow \zeta = \frac{5}{2\sqrt{6}} = 1.02 \Rightarrow \zeta > 1$$

$$C(s) = \frac{6}{s^2+5s+6} \frac{1}{s} = \frac{1}{s} + \frac{-3}{s+2} + \frac{2}{s+3}$$

$$c(t) = 1 - 3e^{-2t} + 2e^{-3t}$$



단위계단응답

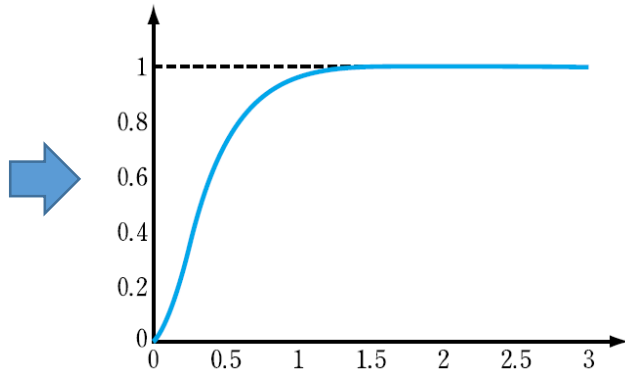
◆ 2차 지연시스템의 단위계단응답

● 임계제동응답 예시

$$R(s) \rightarrow \boxed{\frac{25}{s^2 + 10s + 25}} \rightarrow C(s) \Rightarrow \zeta = \frac{10}{2\sqrt{25}} = 1 \Rightarrow \zeta = 1$$

$$C(s) = \frac{25}{s^2 + 10s + 25} \times \frac{1}{s} = \frac{1}{s} + \frac{-5}{(s+5)^2} + \frac{-1}{s+5}$$

$$c(t) = 1 - 5te^{-5t} - e^{-5t}$$



단위계단응답

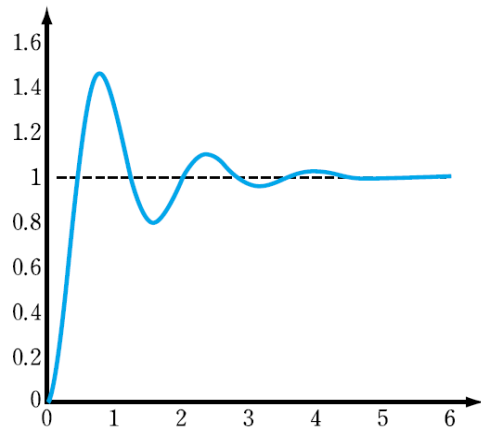
◆ 2차 지연시스템의 단위계단응답

● 부족제동응답 예시

$$R(s) \rightarrow \boxed{\frac{17}{s^2+2s+17}} \rightarrow C(s) \Rightarrow \zeta = \frac{2}{2\sqrt{17}} = 0.24 \Rightarrow \zeta < 1$$

$$\begin{aligned} C(s) &= \frac{17}{s^2+2s+17} \times \frac{1}{s} = \frac{1}{s} - \frac{s+2}{s^2+2s+17} \\ &= \frac{1}{s} - \frac{s+1}{(s+1)^2+4^2} - \frac{1}{(s+1)^2+4^2} \end{aligned}$$

$$\begin{aligned} c(t) &= 1 - e^{-t} \left(\cos 4t + \frac{1}{4} \sin 4t \right) \\ &= 1 - 1.03 e^{-t} \sin(4t + 76^\circ) \end{aligned}$$



단위계단응답

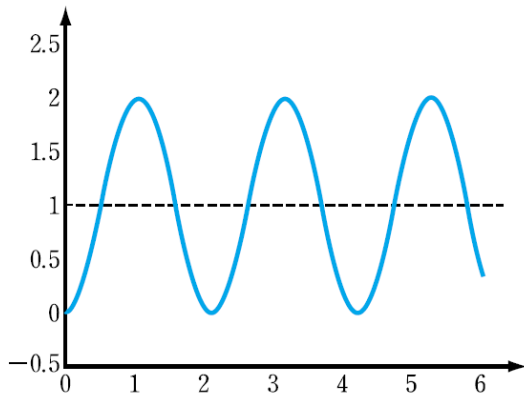
◆ 2차 지연시스템의 단위계단응답

● 무제동응답 예시



$$C(s) = \frac{9}{s^2+9} \times \frac{1}{s} = \frac{1}{s} - \frac{s}{s^2+9}$$

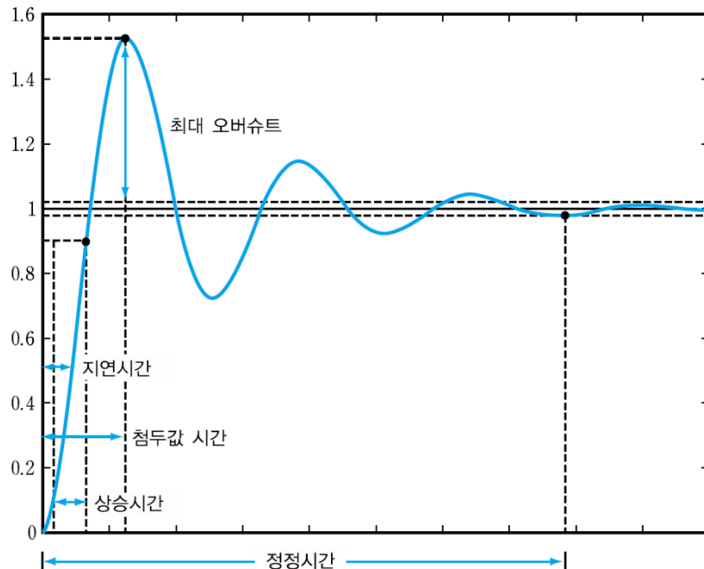
$$c(t) = 1 - \cos 3t$$



단위계단응답

◆ 2차 지연시스템의 단위계단응답

● 부족제동 2차 지연시스템 특성



◆ 2차 지연시스템의 단위계단응답

● 부족제동 2차 지연시스템 특성

전달함수가 다음과 같은 부족제동 2차제어시스템의 단위계단응답에 대한 침투값 시간과 백분율 오버슈트, 상승시간, 정정시간을 구하라.

$$G(s) = \frac{25}{s^2 + 4s + 25}$$

풀이

$$G(s) = \frac{25}{s^2 + 4s + 25} = \frac{5^2}{s^2 + 2 \times 0.4 \times 5s + 5^2}$$



$$\omega_n = 5, \zeta = 0.4$$

$$T_p = \frac{\pi}{\omega_n \sqrt{1-\zeta^2}} = \frac{\pi}{5 \sqrt{1-0.4^2}} \approx 0.69 [\text{sec}]$$

$$\% OS = e^{-(\zeta\pi/\sqrt{1-\zeta^2})} \times 100 = e^{-(0.4\pi/\sqrt{1-0.4^2})} \times 100 \approx 25.4 [\%]$$

$$T_r \approx \frac{0.8 + 2.5\zeta}{\omega_n} = \frac{0.8 + 2.5 \times 0.4}{5} \approx 0.36 [\text{sec}]$$

$$T_s \approx \frac{4}{\zeta\omega_n} = \frac{4}{0.4 \times 5} = 2 [\text{sec}]$$

단위계단응답

◆ 2차 자연시스템의 단위계단응답

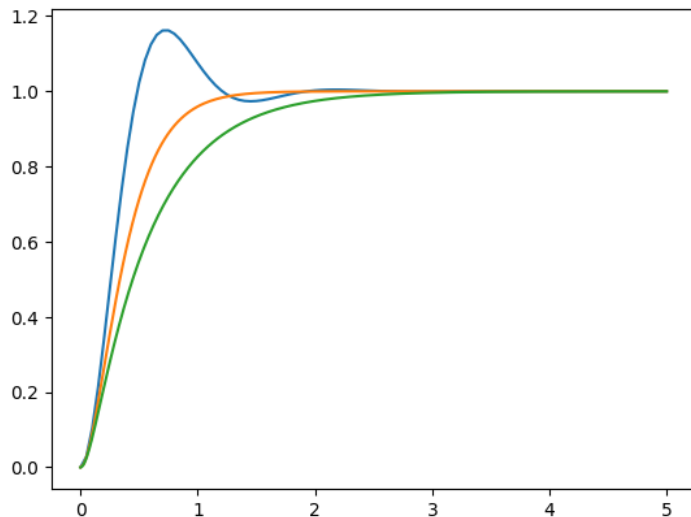
●파이썬 코딩

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

```
H1 = control.tf([25], [1, 5, 25])
H2 = control.tf([25], [1, 10, 25])
H3 = control.tf([25], [1, 15, 25])
```

```
t1, y1 = control.step_response(H1, T=5)
t2, y2 = control.step_response(H2, T=5)
t3, y3 = control.step_response(H3, T=5)
plt.plot(t1,y1, t2,y2, t3,y3)
```

$\zeta = ?$



단위계단응답

◆ 2차 지연시스템의 단위계단응답

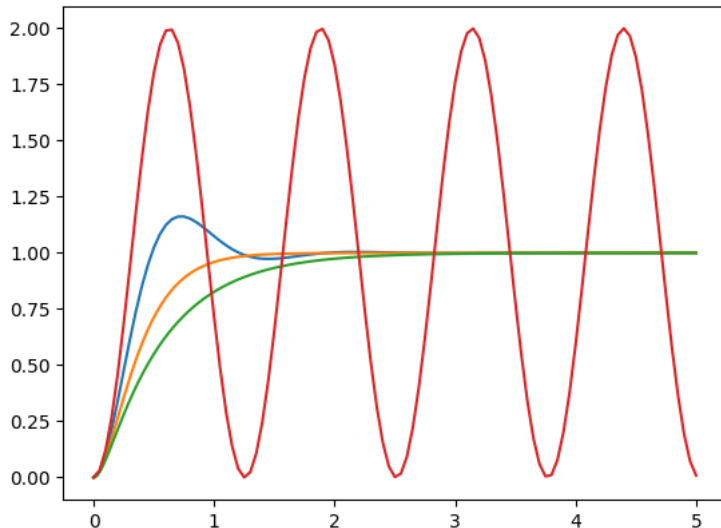
●파이썬 코딩

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

```
H1 = control.tf([25], [1, 5, 25])
H2 = control.tf([25], [1, 10, 25])
H3 = control.tf([25], [1, 15, 25])
H4 = control.tf([25], [1, 0, 25])
```

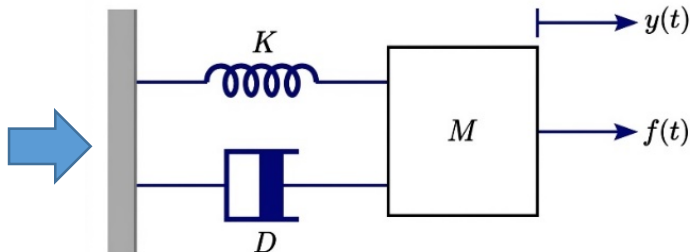
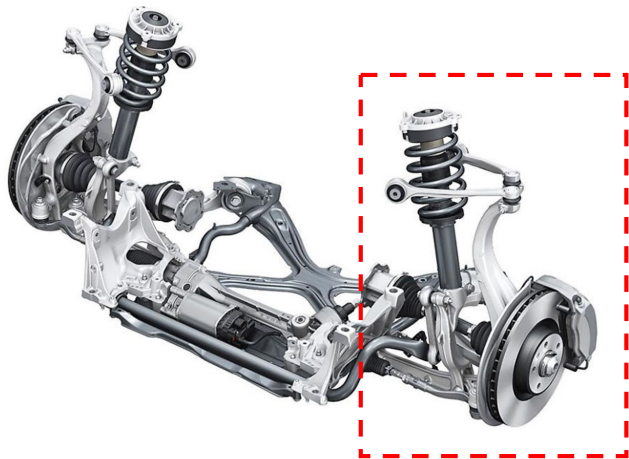
```
t1, y1 = control.step_response(H1, T=5)
t2, y2 = control.step_response(H2, T=5)
t3, y3 = control.step_response(H3, T=5)
t4, y4 = control.step_response(H4, T=5)
plt.plot(t1,y1, t2,y2, t3,y3, t4,y4)
```

$\zeta = ?$



◆ 2차 지연시스템의 단위계단응답

● 스프링-댐퍼 시스템 설계



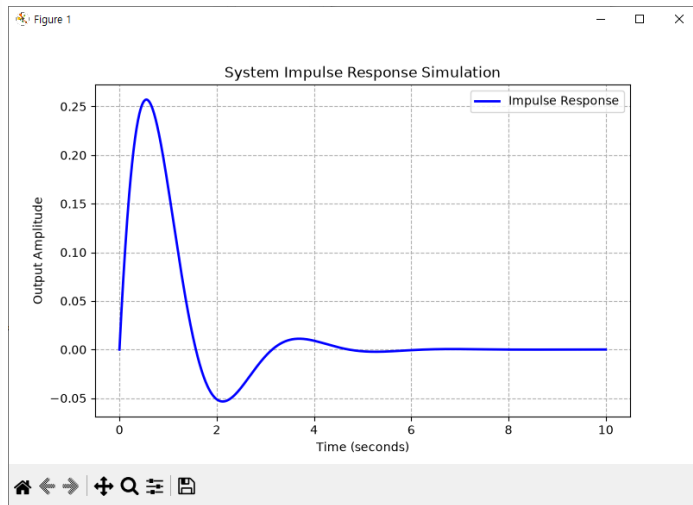
$$M \frac{d^2}{dt^2} y(t) + D \frac{d}{dt} y(t) + K y(t) = f(t)$$

단위계단응답

◆ 2차 지연시스템의 단위계단응답

● 스프링-댐퍼 시스템 설계 → 임펄스 입력 → 일종의 방지턱 충격

```
1 import control as ct
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 # 1. 전달 함수(Transfer Function) 정의
6 #  $G(s) = 1 / (s^2 + 2s + 5)$ 
7 num = [1]          # 분자 계수
8 den = [1, 2, 5]    # 분모 계수 ( $s^2 + 2s + 5$ )
9 sys = ct.tf(num, den)
10
11 # 2. 시뮬레이션 시간 배열 생성 (0초부터 10초까지 1000개 구간)
12 time = np.linspace(0, 10, 1000)
13
14 # 3. 임펄스 응답 계산
15 # 결과로 시간(t)과 시스템의 출력 값(y)을 반환합니다.
16 t, y = ct.impulse_response(sys, T=time)
17
18 # 4. 결과 시각화 (Graph)
19 plt.figure(figsize=(8, 5))
20 plt.plot(t, y, label='Impulse Response', color='blue', linewidth=2)
21 plt.title('System Impulse Response Simulation')
22 plt.xlabel('Time (seconds)')
23 plt.ylabel('Output Amplitude')
24 plt.grid(True, linestyle='--')
25 plt.legend()
26 plt.show()
```



◆ 2차 지연시스템의 단위계단응답

● 스프링-댐퍼 시스템 설계 → 임펄스 입력 → 일종의 방지턱 충격

```
import control as ct
import matplotlib.pyplot as plt
import numpy as np

# 1. 전달 함수(Transfer Function) 정의
#  $G(s) = 1 / (s^2 + 2s + 5)$ 
num = [1]      # 분자 계수
den = [1, 2, 5] # 분모 계수 ( $s^2 + 2s + 5$ )
sys = ct.tf(num, den)

# 2. 시뮬레이션 시간 배열 생성 (0초부터 10초까지 1000개 구간)
time = np.linspace(0, 10, 1000)

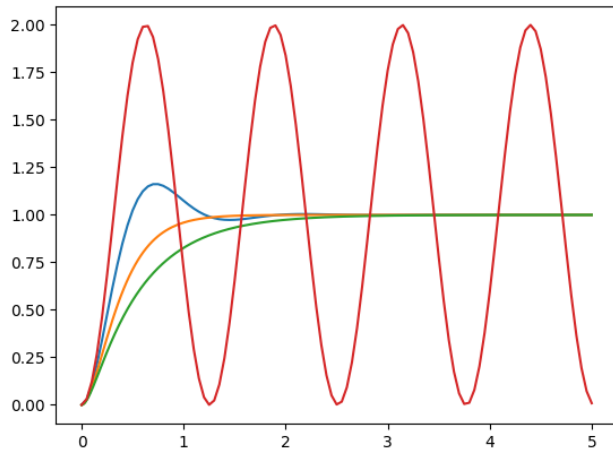
# 3. 임펄스 응답 계산
# 결과로 시간(t)과 시스템의 출력 값(y)을 반환합니다.
t, y = ct.impulse_response(sys, T=time)

# 4. 결과 시각화 (Graph)
plt.figure(figsize=(8, 5))
plt.plot(t, y, label='Impulse Response', color='blue', linewidth=2)
plt.title('System Impulse Response Simulation')
plt.xlabel('Time (seconds)')
plt.ylabel('Output Amplitude')
plt.grid(True, linestyle='--')
plt.legend()
plt.show()
```

단위계단응답

◆ 2차 자연시스템의 단위계단응답

● 스프링-댐퍼 시스템 설계 → 임펄스 입력 → 일종의 방지턱 충격



왼쪽의 단위계단응답처럼 임펄스 응답에 대한 그래프 출력 하시오.

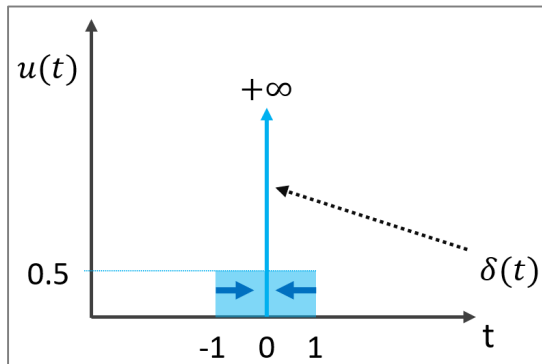
◆ 참고

● 임펄스 입력

임펄스 함수(디랙 델타 함수, $\delta(t)$)의 라플라스 변환 값은 1입니다.

이 성질 때문에 제어공학에서 임펄스 응답은 시스템의 고유한 특성(전달함수)을 그대로 보여주는 중요한 지표가 됩니다.

$$\mathcal{L}\{\delta(t)\} = 1$$



◆ 참고

● 임펄스 입력

📌 수학적 증명 및 정의

임펄스 함수는 $t=0$ 인 순간에 폭이 무한히 좁고 높이가 무한히 높으며, 전체 적분 면적은 항상 1이 되는 가상의 함수입니다. 라플라스 변환의 정의식을 통해 증명하면 다음과 같습니다.

1. 라플라스 변환의 기본 정의식

$$\mathcal{L}\{\delta(t)\} = \int_{0-}^{\infty} \delta(t)e^{-st}dt$$

(이때 적분 하한을 0-로 잡는 이유는 $t=0$ 에서 일어나는 임펄스 신호를 완전히 포함하기 위함입니다.)

2. 디랙 델타 함수의 체 거르기 성질 (Sifting Property)

델타 함수와 어떤 함수 $f(t)$ 를 곱해서 적분하면, 델타 함수가 무한대가 되는 순간($t=0$)의 함수 값만 속 빠져나옵니다.

$$\int_{-\infty}^{\infty} \delta(t-a)f(t)dt = f(a)$$

3. 최종 계산

위의 성질을 대입하면 $t=0$ 일 때의 e^{-st} 값이 결과가 됩니다.

$$\int_{0-}^{\infty} \delta(t)e^{-st}dt = e^{-s \cdot 0} = e^0 = 1$$

◆ 참고

● 임펄스 입력

1. 직사각형 직관을 통한 적분 이해 (면적 = 1)

임펄스 함수는 수학적으로 폭이 매우 좁고 높이가 매우 높은 직사각형의 **극한**으로 정의됩니다.

- 폭이 ϵ 이고, 높이가 $\frac{1}{\epsilon}$ 인 직사각형 함수 $d_{\epsilon}(t)$ 를 가정해 봅시다.
- 이 직사각형의 면적은 $t = 0$ 근처에서 항상 $\epsilon \times \frac{1}{\epsilon} = 1$ 입니다.

이제 폭 ϵ 을 0에 무한히 가깝게 만드는 극한($\epsilon \rightarrow 0$)을 취하면, 폭은 0이 되고 높이는 무한대(∞)가 되지만 **면적은 여전히 1로 유지**됩니다. 이것이 디랙 델타 함수 $\delta(t)$ 입니다.

따라서 임펄스 함수를 전 구간에 대해 적분하면 그 결과는 항상 **1**이 됩니다.

$$\int_{-\infty}^{\infty} \delta(t) dt = 1$$

◆ 참고

● 임펄스 입력

2. 체 거르기 성질(Sifting Property)의 세부 원리

질문하신 라플라스 변환에서 사용된 $\int_{-\infty}^{\infty} \delta(t)f(t) dt = f(0)$ 의 원리를 직사각형 극한으로 다시 쪼개어 설명해 드리겠습니다.

어떤 연속 함수 $f(t)$ 와 앞서 말한 폭이 ϵ 인 임펄스 근사 함수를 곱한 뒤 적분한다고 가정해 봅시다.

$$\int_{-\infty}^{\infty} d_{\epsilon}(t)f(t) dt = \int_0^{\epsilon} \frac{1}{\epsilon} f(t) dt$$

여기서 ϵ 이 0에 부근으로 아주 작아지면, 구간 $[0, \epsilon]$ 사이에서 연속 함수 $f(t)$ 의 값은 변하지 않고 거의 고정된 상수값 $f(0)$ 에 수렴하게 됩니다. 변수가 아닌 상수가 된 $f(0)$ 를 적분 기호 밖으로 빼내면 다음과 같습니다.

$$\approx f(0) \cdot \int_0^{\epsilon} \frac{1}{\epsilon} dt = f(0) \cdot \left(\frac{1}{\epsilon} \cdot \epsilon \right) = f(0) \cdot 1 = f(0)$$

즉, 임펄스 함수가 $t = 0$ 에서 무한대를 찍는 그 찰나의 순간에 $f(t)$ 함수에서 $t = 0$ 일 때의 딱 한 점 ($f(0)$)만 '체로 걸러내듯' 쓱 뽑아내고, 나머지 구간은 임펄스 값이 0이므로 전부 사라지는 것입니다.

◆ 참고

● 임펄스 입력

💡 제어공학에서 가지는 의미

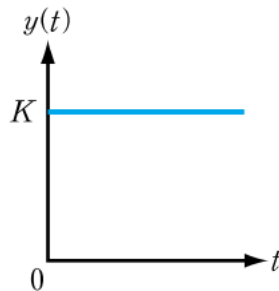
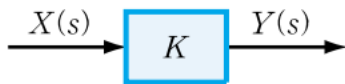
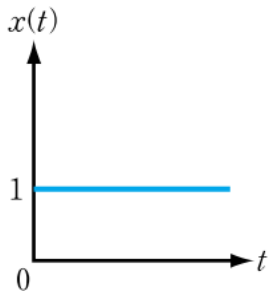
이 변환 결과는 제어 시스템을 분석할 때 엄청난 편리함을 줍니다.

- **전달함수와의 관계:** 시스템의 전달함수를 $G(s)$, 입력을 $X(s)$, 출력을 $Y(s)$ 라고 할 때 $Y(s) = G(s)X(s)$ 입니다. 이때 입력이 임펄스 신호라면 $X(s) = 1$ 이 되므로 **출력 자체가 곧 전달함수($Y(s) = G(s)$)**가 됩니다.
- **임펄스 응답:** 따라서 출력인 $Y(s)$ 를 다시 시간 영역으로 역라플라스 변환한 $y(t)$ 는 시스템의 순수한 고유 특성을 나타내며, 이를 **임펄스 응답(Impulse Response)**이라고 부릅니다.

◆ 비례요소

- 목표값과 현재값의 차이(오차)에 비례하여 제어 신호를 얼마나 강하게 보낼지 결정
- 현재 오차에 대응하여 빠르게 움직임 → 속도 담당

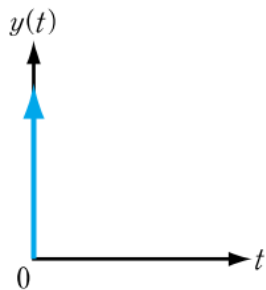
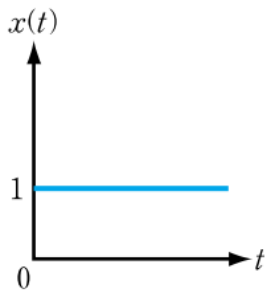
$$y(t) = K_p x(t) \quad \Rightarrow \quad Y(s) = K_p X(s) \quad \Rightarrow \quad G(s) = \frac{Y(s)}{X(s)} = K_p$$



◆ 미분요소

- 오차의 변화 속도(변화율)을 감지하여 앞으로 발생할 오차를 예측하고 제어
- 미래 오차를 예측하여 안정적으로 제동함 → 브레이크 담당

$$y(t) = K_d \frac{d}{dt} x(t) \quad \Rightarrow \quad Y(s) = K_d s X(s) \quad \Rightarrow \quad G(s) = \frac{Y(s)}{X(s)} = K_d s$$



◆ 적분요소

- 과거부터 누적된 오차의 합을 계산하여 시스템의 정상상태 오차를 완전히 제거
- 과거 오차를 누적하여 정확하게 맞춤 → 오차 제거 담당

$$y(t) = K_i \int_0^t x(t) dt \quad \Rightarrow \quad Y(s) = \frac{K_i}{s} X(s) \quad \Rightarrow \quad G(s) = \frac{Y(s)}{X(s)} = \frac{K_i}{s}$$

