2021.9 Y. Minami

# Python-Control *Cheat Sheet*

```python
from control.matlab import *
import numpy as np
```

## 1 System representation

### Transfer Function

```python
P = tf([0, 4], [1, 2, 3])
```

$$P(s) = \frac{4}{s^2 + 2s + 3}$$

```python
s = tf('s')
P = 4/(s**2 + 2*s + 3)
```

### State-Space Equation

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ -4 & -5 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$

$$y(t) = x(t)$$

```python
A = [[0, 1],[-4, -5]]
B = [[0], [1]]

C = np.eye(2)
D = np.zeros([2, 1])
P = ss(A, B, C, D)
```

▶ Controllability and Reachability Matrix

```python
Vc = ctrb(P.A, P.B)
```

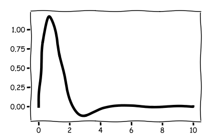
```python
Vo = obsv(P.A, P.C)
```

## 2 Time response

### Impulse response

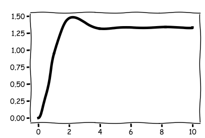
```python
T = np.arange(0, 10, 0.01)
y, t = impulse(P, T)
```

0, 0.01, 0.02,...,9.99



### Step response

```python
T = np.arange(0, 10, 0.01)
y, t = step(P, T)
```

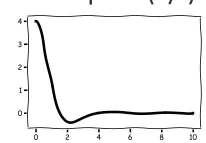


▶ Step response characteristics

Info = stepinfo(sys)

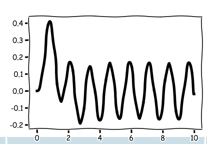
### Initial condition response

```python
T = np.arange(0, 10, 0.01)
X0 = [0, -1]
y, t = initial(P, T, X0)
```



### Forced response

```python
T = np.arange(0, 10, 0.01)
U = np.sin(5*T)
y, t, x0 = lsim(P, U, T)
```



## 3 Frequency response

### Bode diagram $P(j\omega) = \alpha(\omega)e^{j\beta(\omega)}$

```python
gain, phase, w = bode(P, logspace(-2,2))
```

$10^{-2} \sim 10^{2}$

Gain $20\log_{10}|\alpha(\omega)|$

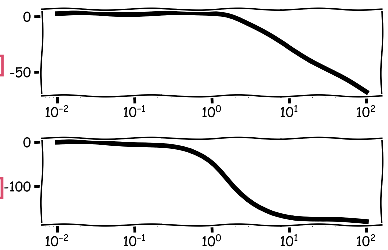
```python
20*np.log10(gain)
```

[dB]

Phase $\beta(\omega) \times \frac{180}{\pi}$

```python
phase*180/np.pi
```

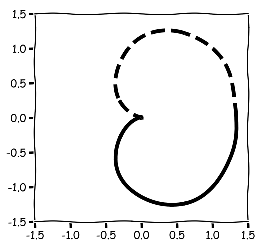
[deg]




▶ Frequency response at multiple angular frequencies
gain, phase, w = freqresp(sys, [omega])

### Nyquist diagram $P(j\omega) = x(\omega) + jy(\omega)$

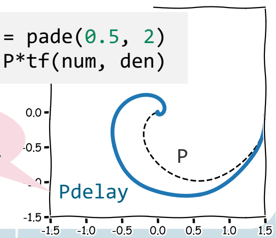
```python
x, y, w = nyquist(P, logspace(-2,2,1000))
```



### Pade approximation

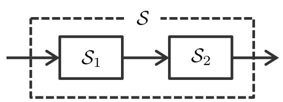
```python
num, den = pade(0.5, 2)
Pdelay = P*tf(num, den)
```

Delay = **0.5**[s]
**2nd order** appr.

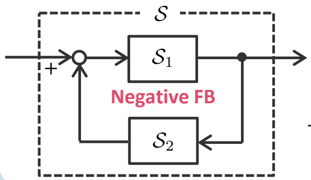



## 4 Block diagram

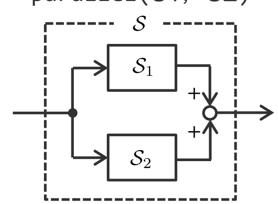
series(S1, S2)



feedback(S1, S2, sign=-1)




parallel(S1, S2)



▶ minreal(sys)

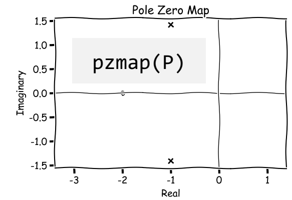
## 5 Stability and Robustness

### Pole & Zero

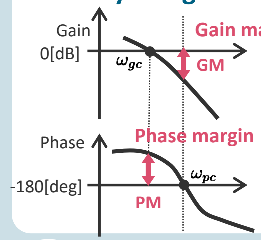
```python
P.pole()
```

```python
P.zero()
```

```python
pzmap(P)
```



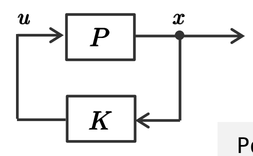

### Stability margin




```python
GM, PM, wpc, wgc = margin(P)
print('GM=', 20*np.log10(GM))
print('PM=', PM)
print('wpc=', wpc)
print('wgc=', wgc)
```

Phase crossover
Gain crossover

## 6 Controller design



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

$$K: u(t) = Fx(t)$$

### Pole placement

```python
Pole = [-1, -2]
F = -acker(P.A, P.B, Pole)
```

```python
F = -place(P.A, P.B, Pole)
```

### LQ optimal control

$$J = \int_0^\infty \{x^\top Q x + u^\top R u\}\, dt$$

```python
Q = [ [100, 0], [0, 1]]
R = 1
F, _, _ = lqr(P.A, P.B, Q, R)
F = -F
```

▶ solves the continuous-time algebraic Riccati equation
(X, L, G) = care(A, B, Q, R)

### Mixied sensitivity design

```python
from control import mixsyn
```

```python
K, _, gamma = mixsyn(Sys, w1=WS , w2=WU, w3=WT)
```

## 7 Digitalization

### Zero Order Hold

```python
ts = 0.2
Pd = c2d(P, ts, method='zoh')
```

### Tustin Transformation

```python
ts = 0.2
Pd = c2d(P, ts, method='tustin')
```