古典制御

No. 1234　都倉 佑悟

1　はじめに

本資料では古典制御の考え方を説明し, プログラミング (Python) を用いた古典制御の実装方法を紹介する.

2　制御対象

制御対象には図 1 に示す 1 リンクマニュピレータを用いる.

3　モデル化

制御対象を任意に制御する為には, 制御対象の動特性を知らなければならない. ここで動特性とは時間に依存する制御量の振る舞いであり, この動特性は微分方程式で記述される. 即ち, 制御対象を制御する為には制御対象を微分方程式で記述する必要がある. この制御対象を微分方程式で記述する過程をモデル化と呼び, モデル化された対象を数理モデル, または単にモデルと呼ぶ. ニュートンの運動方程式はこれに該当する.

次に図 1 をモデル化する. 出力は角度 $\theta(t)$ とし, 入力は原点に加わるトルク T, 原点まわりの慣性モーメント (Moment of Inertia: MoI) を I_o, ダンパ係数 (Dumping cofficient) を c とした時, 制御対象は式 1 の様に書ける.

$$I_o\ddot{\theta}(t) + c\dot{\theta}(t) = T \tag{1}$$

この微分方程式を解く事で, 時間領域での出力

$$\theta(t) = C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t} + \alpha \tag{2}$$

を得る. これより時間領域出力 $\theta(t)$ の動特性は得られたが, 更なる制御入力を加える場合, 式 1 に手を加え再び微分方程式を解く必要に迫られる. もちろん解けば問題ないのだが, その手順は煩雑であり完全微分方程式型など特殊な解放が必要になる事も少なくない. また制御を行いたいのは過渡状態でなく定常状態である事も多く, この場合時間領域解は冗長である.

これらの問題を解決する為に時間領域で考える事は一旦諦め, 周波数領域で考える事とする. この時間領域から周波数領域への写像をラプラス変換と呼ぶ.

4　ラプラス変換

ラプラス変換の定義や手法は各専門書に任せる事とし, 初期状態 $\theta(0) = 0,\ \dot{\theta}(0) = 0$ のもとで式 1 をラプラス変換する事で式 3 を得る.

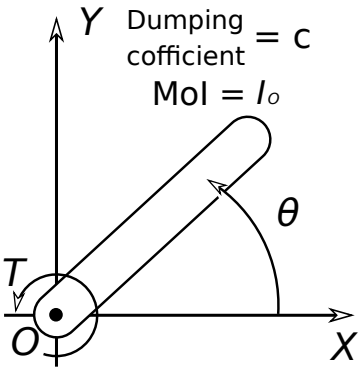

図 1　1 リンクマニュピレータ

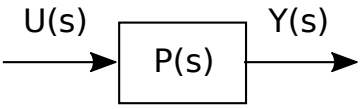

図 2　式 4 のブロック線図

$$s^2 I_o \Theta(s) + sc\Theta(s) = T \tag{3}$$

ここで出力, 入力をそれぞれ $\Theta(s) = Y(s), T = U(s)$ と置き式 3 を式 4 の様に書き換え, まとめる.

$$\begin{aligned} s^2 I_o Y(s) + scY(s) &= U(s) \\ Y(s)(s^2 I_o + sc) &= U(s) \\ Y(s) &= \frac{1}{s^2 I_o + sc} U(s) \end{aligned} \tag{4}$$

これより周波数領域での出力が時間領域に対して比較的容易に得られ, 入力と出力の関係が明瞭に分かる. また更なる入力を加える為には $U(s)$ を変えるだけでよく, 時間領域の様に微分方程式を解き直す必要がない. 式 4 のブロック線図を図 2 に示し, 次章では式 4 と最終値の定理を用いて定常応答を調べる.

5　定常応答

5.1　最終値の定理

定常応答とは時間を無限に飛ばした時の出力, 即ち $\lim_{t\to\infty} y(t)$ である. これは時間領域での出力 $y(t)$ に用いられ, 周波数領域での出力 $Y(s)$ と次式の関係がある.

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0} sY(s) \tag{5}$$

式 5 の関係を最終値の定理と呼び, 周波数領域へ写像された時間領域の定常応答を, 周波数領域のまま計算する事が出来る.

5.2 定常応答を求める

1 リンクマニュピレータの時間領域での目標値には一定値が与えられる事が多い. そこで時間領域の入力も目標値と同じ一定値を入れてみる.

一定の時間領域入力を $T = u(t) = k\ (k = const)$ とした時, ラプラス変換より周波数領域入力 $U(s) = k/s\ (k = const)$ を得る. これを式 4 に代入すると周波数領域出力が次式となる.

$$Y(s) = \frac{1}{s^2 I_o + sc} \times \frac{k}{s} \tag{6}$$

式 6 の時間領域での定常応答は最終値の定理より

$$\begin{aligned} \lim_{t\to\infty} y(t) &= \lim_{s\to 0} sY(s) \\ \lim_{s\to 0} sY(s) &= \lim_{s\to 0} s\left(\frac{1}{s^2 I_o + sc} \times \frac{k}{s}\right) \\ &= \lim_{s\to 0} \frac{k}{s^2 I_o + sc} \\ &= \infty \end{aligned} \tag{7}$$

となり時間領域出力は発散することが分かる.

5.3 フィードバックループをつける

入力を一定値にすると発散してしまったので, フィードバックループを用いて逐次入力を変更出来る様に図 3 の様な制御系を構築する. ここで $R(s), E(s)$ は目標値, 偏差であり偏差は

$$E(s) = R(s) - Y(s) \tag{8}$$

で計算される. この時目標値と出力の値は

$$Y(s) = \frac{C(s)P(s)}{1 + C(s)P(s)} R(s) \tag{9}$$

となり, これより偏差 $E(s)$ は

$$E(s) = \frac{1}{1 + C(s)P(s)} R(s) \tag{10}$$

となる. この偏差が 0 となる時, 目標値と出力の値が一致するので, フィードバックループをつけた時は定常時の偏差が 0 になる様にすれば出力が発散せずうまく制御する事が出来る.

次に式 10 は最終値の定理より

$$\begin{aligned} \lim_{t\to\infty} e(t) &= \lim_{s\to 0} sE(s) \\ \lim_{s\to 0} sE(s) &= \lim_{s\to 0} s\left(\frac{1}{1 + C(s)P(s)} R(s)\right) \end{aligned} \tag{11}$$

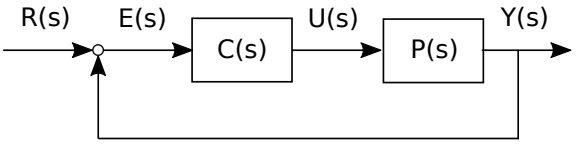

図 3 フィードバックループ

となり, ここで一定の目標値 $R(s) = k/s\ (k = const)$ を印加すると

$$\begin{aligned} \lim_{s\to 0} sE(s) &= \lim_{s\to 0} s\left(\frac{1}{1 + C(s)P(s)} \times \frac{k}{s}\right) \\ &= \lim_{s\to 0} \frac{k}{1 + C(s)P(s)} \end{aligned} \tag{12}$$

また $C(s) = 1$ とし, $P(s) = 1/(s^2 I_o + sc)$ なので

$$\begin{aligned} \lim_{s\to 0} C(s)P(s) &= \lim_{s\to 0} \frac{1}{s^2 I_o + sc} = \infty \\ \therefore \lim_{s\to 0} \frac{k}{1 + C(s)P(s)} &= \frac{k}{1 + \infty} = 0 \end{aligned} \tag{13}$$

となる. これよりフィードバックループをつけた場合, 定常時で偏差がなくなりうまく制御出来る事が分かる.

6 Python を用いた実装

#フィードバックループなし

#パラメータ

```
I_o = 1.0
c = 1.0
```

#関数

```
P = lambda s: 1/(I_o * s **2 + c * s)
U_step = lambda s: 1/s
final_value_step = lambda s: s * P(s) * U_step(s)
print([final_value_step(s) for s in (1,0.1,0.01,0.001)]
```

#フィードバックループあり

#パラメータ

```
I_o = 1.0
c = 1.0
```

#関数

```
C = lambda s: 1
P = lambda s: 1/(I_o * s **2 + c * s)
E = lambda s: 1/(1 + C(s)*P(s))
U_step = lambda s: 1/s
final_value_step = lambda s: s * E(s) * U_step(s)
print([final_value_step(s) for s in (1,0.1,0.01,0.001)]
```