現代制御

No. 1234　都倉 佑悟

1　はじめに

本資料では現代制御の考え方を説明し, プログラミング (Python) を用いた現代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　状態空間表現

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) のように書ける.

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

ここで新たな変数を状態変数 $x_1(t) = \theta(t), x_2(t) = \dot{\theta}(t)$ と定義し, $T = u(t)$ とすると式 (1) は

$$I_o\dot{x_2}(t) + cx_2(t) = u(t) \tag{2}$$

となり, これを $\dot{x_2}$ について解くことで次式を得る.

$$\dot{x_2}(t) = -\frac{c}{I_o}x_2 + \frac{1}{I_o}u(t) \tag{3}$$

また $x_1(t) = \theta(t), x_2(t) = \dot{\theta}(t)$ より, $\dot{x_1}(t) = x_2(t)$ となるので, 連立方程式を用いて

$$\begin{cases} \dot{x_1}(t) = x_2(t) \\ \dot{x_2}(t) = -\dfrac{c}{I_o}x_2(t) + \dfrac{1}{I_o}u(t) \end{cases} \tag{4}$$

とまとめることが出来, 更に状態ベクトルを $\boldsymbol{x} = [x_1(t)\ x_2(t)]^{\mathrm{T}}$とすると$\dot{\boldsymbol{x}} = [\dot{x_1}(t)\ \dot{x_2}(t)]^{\mathrm{T}}$ となり, これらと行列を用いることで式 (4) が次のように書ける.

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{c}{I_o} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{I_o} \end{bmatrix} u(t) \tag{5}$$

式 (5) は状態ベクトル $\boldsymbol{x} = [x_1(t)\ \ x_2(t)]^{\mathrm{T}}, \dot{\boldsymbol{x}} = [\dot{x_1}(t)\ \dot{x_2}(t)]^{\mathrm{T}}$ を用いることで

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{c}{I_o} \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 0 \\ \frac{1}{I_o} \end{bmatrix} u(t) \tag{6}$$

となり,$\boldsymbol{x}$ に付随する行列を $\boldsymbol{A}$,$u(t)$ に付随する行列を $\boldsymbol{B}$ とすると次式, 状態方程式を得る.

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}u(t) \tag{7}$$

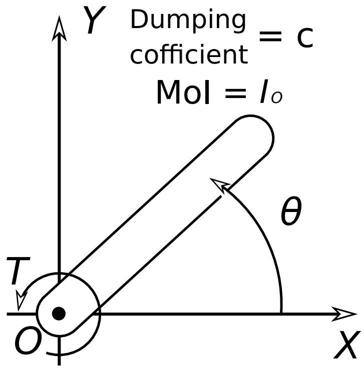

図 1　1 リンクマニュピレータ

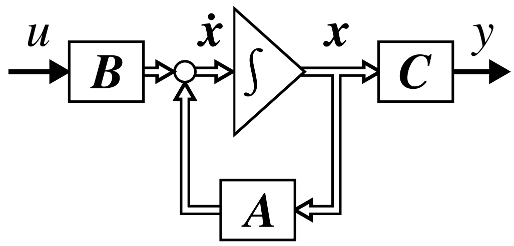

図 2　ブロック線図

また, 出力を $y(t) = \theta(t) = x_1(t)$ とすると, これも状態ベクトル $\boldsymbol{x} = [x_1(t)\ x_2(t)]^{\mathrm{T}}$ と行列を用いて

$$y(t) = [\ 1 \quad 0\ ]\boldsymbol{x}(t) \tag{8}$$

と書け, 最後に状態ベクトル $\boldsymbol{x}$ に付随する行列を $\boldsymbol{C}$ とすることで, 次式のように示される出力方程式を得る.

$$y(t) = \boldsymbol{C}\boldsymbol{x}(t) \tag{9}$$

式 (7) と式 (9) をまとめると

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}u(t) \\ y(t) = \boldsymbol{C}\boldsymbol{x}(t) \end{cases} \tag{10}$$

上記式 (10) を得, 式 (10) を状態空間表現と呼ぶ. 式 (10) をブロック線図に描いた物を図 2 に示す.

4　状態空間

状態変数を座標軸とする空間を状態空間と呼ぶ. 例えば状態ベクトルを $\boldsymbol{x}(t) = [x_1(t)\ \ x_2(t)]^{\mathrm{T}}$ とした時,$x_1(t), x_2(t)$ が状態変数に対応し x_1を座標平面の x 軸に,x_2を座標平面の y 軸とした 図 3 が状態空間となる.

もちろん状態変数を増やせば増やす程状態空間の座標軸も増えていく.

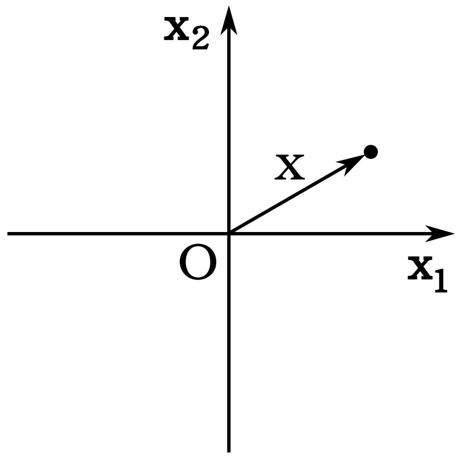

図 3　状態空間

5　Python を用いた実装

```
# 行列計算ライブラリ、numpy をインポート
import numpy as np
# グラフ描画ライブラリ、matplotlib をインポート。
# 実際にインポートするのは matplotlib を簡単に使える
# 補助ライブラリ、matplotlib.pyplot
import matplotlib.pyplot as plt

# 目的: システムのパラメータを定義する(スカラ)
# c: float
c  = 0.1
# I0: float
I0 = 1.0

# 目的: システムのパラメータを定義する(行列とベクトル)
# A: np.ndarray(2, 2)
A = np.array([[0,     1],
              [0, -c/I0]])
# B: np.ndarray(2,)
B = np.array([0, 1/I0])
# C: np.ndarray(2,)
C = np.array([1, 0])

# 目的: 状態 x、入力 u についての状態の微分 dx/dt を計算する
# func_dxdt: (np.ndarray(2,), float) -> np.ndarray(2,)
def func_dxdt(x, u):
    return A @ x + B * u

# 目的: 状態 x についての出力 y を計算する
# func_y: (np.ndarray(2,)) -> float
def func_y(x):
    return C @ x

# 目的: 状態 x についての入力 u を計算する
# func_y: (np.ndarray(2,)) -> float
def func_u(x):
    return 1

if __name__ == "__main__":
    # x の初期値
    x = np.array([0, 0])
    # シミュレーションの時系列データ
    ts = np.linspace(0, 10, 1001)
    # 時間間隔。ts から導出される
    dt = ts[1] - ts[0]

    # x, y, u の時系列データを保存する
    # x_log: np.ndarray(1001, 2)
    x_log = np.zeros((1001, 2))
    # y_log: np.ndarray(1001, 2)
    y_log = np.zeros((1001, 1))
    # u_log: np.ndarray(1001, 2)
    u_log = np.zeros((1001, 1))

    # オイラー法で時系列を計算する
    for i in range(1001):
        y = func_y(x)
        u = func_u(x)
        x_log[i] = x
        y_log[i] = y
        u_log[i] = u
        x = x + func_dxdt(x, u) * dt

    # 結果をプロットする
    plt.figure()
    plt.plot(ts, x_log)
    plt.figure()
    plt.plot(ts, y_log)
    plt.figure()
    plt.plot(ts, u_log)
    plt.show()
```