

東洋大学大学院理工学研究科機能システム専攻

制御工学特論
講義資料

担当：山川聡子

目次

1. はじめに	1
2. 運動方程式	
3. システムの表現	
4. システムの時間応答	
5. 安定性	
6. 出力フィードバック制御	
7. 可制御性・可観測性	
8. レギュレータ	
9. オブザーバ	
10. サーボ系	
11. 簡単な非線形制御	
12. 移動ロボットを用いた演習	
授業の課題(レポート)	
付録	

1. はじめに

制御工学は、ロボットや機械を希望通りに動かすために不可欠な知識である。動的システムの特徴を捉え、解析し、動かす制御工学の考え方は、機械工学分野に限らず、経済、生体活動の解析や制御などにも広く応用できる。

近年、企業においても制御系設計の支援ツールとしてコンピュータのソフトウェアを用いることが多くなっている。そこで、本講義では、自動車産業などでも良く用いられている MATLAB、制御系設計に有効な数式処理ソフトである Mathematica を用いた演習を取り入れて、制御対象のモデリングと制御系の解析法、および代表的な制御系設計法について講義を行う。講義の終わりには、倒立振り子システムを対象として実際に制御則の設計を行う。

本講義では、古典制御との関連を加えながら、現代制御について説明を行う。古典制御の基礎的な知識、および、物理と行列の基礎的な知識は、各自復習のうで受講してほしい。

1-1 現代制御と古典制御

制御を人の手で行うのではなく、機械や電気回路、コンピュータなどを用いて自動的に行うことを自動制御という。自動制御は、18世紀の産業革命の時代、J.Watt の蒸気機関の回転数を一定に保つガバナという装置から始まったといわれている。その後、安定性などが数学的に解析され、制御工学といわれる分野が発達した。19世紀の中頃には、Bode や Nyquist などによって、周波数領域での解析が行われ、古典制御と呼ばれる解析手法が成熟した。1950年頃になると、制御対象が複雑になり、多入力多出力システムを制御するための現代制御理論が登場した。Kalman や Luenberger らによって示された現代制御は、最適性を取り入れた体系的な理論である。システムの内部状態という考え方が導入され、可制御性や可観測性なども定義され、制御系設計の条件もより明確に示されている。しかし、システム表現やコントローラ設計に行列とその方程式を用いること、制御対象のモデルを必要とすることなどから、経験的な古典制御に比べると産業分野での普及は滞っていた。1980年代になると、ポスト現代制御として、 H_{∞} 制御といわれる制御法が登場した。 H_{∞} 制御は、外乱やモデル化誤差に対してもシステム性能を保証するロバスト性を考えた制御法である。これは、古典制御の周波数領域での解析と、現代制御の多入力多出力を取り扱うための定式化など、二つの方法の性質を合わせ持っている。しかし、コントローラ設計には、現代制御と同等、もしくはそれ以上の数学的知識を必要とした。

1-2 制御系設計支援ソフト

1980年頃に、Mathematica や MATLAB といわれる数式処理ソフトが登場した。特に、1984年に製品化された MATLAB では制御系設計用のアプリケーションが多数提供された。このアプリケーションのおかげで、古典、現代制御のみならず、ロバスト制御などの複雑な制御系設計における繰り返し計算なども、非常に簡単に行うことができるようになった。また、DSP(Digital signal processor)とのインターフェースも提供されたため、シミュレーションから実験までが手軽にできる環境ができた。その結果、多くの企業が利用し、 H_{∞} 制御は自動車や航空機関連の企業でも実際に利用されるようになった。

以下では、Mathematica と MATLAB という二つの数式処理ソフトについて説明する。

0. 数式処理ソフト

数式処理ソフトは、その名の通り、数式を取り扱うことができるソフトウェアである。一般のコンピュータ言語との違いを有理数の取り扱いの例で示そう。

問い： $\frac{1}{6} \times 6$ はいくつか？

もちろん、答えは1である。さて、この計算を最も手近なコンピュータである電卓で行ってみよう。まず、 $1/6$ を計算すると、答えは0.1666666667と表示される。この時点で、 $1/6$ ではない。しかし、これに6をかけると1になる。この調子で、つぎのような計算をしてみよう。

$$\underbrace{\frac{1}{6} \times \frac{1}{6} \times \dots \times \frac{1}{6} \times \frac{1}{6} \times \frac{1}{6}}_{n \text{ 回}} \times \underbrace{6 \times 6 \times \dots \times 6 \times 6 \times 6}_{n \text{ 回}}$$

もちろん、自然数 n がいくつであっても厳密解は1である。電卓で計算した場合、 $n=3$ 回程度なら、1に戻るであろう。もっと n を大きくしたらどうなるか？これは、電卓の性能を測るのに良い実験でもある。性能の悪い電卓だと $n=10$ 程度で答えが0.999999となり、1にならない。

この現象は電卓やコンピュータの計算丸め誤差に起因する。コンピュータは、0と1の間を有限の数字で分割して取り扱うため、この誤差が生じてしまう。

これに対して、数式処理ソフトは、 $1 \div 6$ の答えを $\frac{1}{6}$ と返すことができるソフトである。つまり、 $1/6$ を「 $1 \div 6$ の計算を実行した結果の数値」ではなく、「1を6で割る値」として記憶している。同じように $1/a$ といったように、未知変数を用いた計算も可能である。さらに、数式処理ソフトの多くは様々な数学公式のライブラリを持っており、微分積分、方程式の求解なども可能である。

このような数式処理ソフトは1960年代後半から研究されてきており、以下で紹介するMathematicaやMATLAB以外にもReduceやMapleなど、古くから多くのソフトが開発されている。

1. MATLAB

matrix laboratory の略である。

行列、ベクトル計算、グラフなどのライブラリを持つ技術計算言語（環境）である。DSP用コードを作成するコンパイラなどが用意されており、コントローラ設計、シミュレーションから実装までを行うことができる。そのため、企業におけるコントローラ設計でも利用されている。

1970年後半 Cleve Moler(ニューメキシコ大)が開発。

1984年 John N. Little が商品化 Mathworks 社

↑制御工学が専門（制御用アプリケーションを作成→制御分野で普及）

2. Mathematica

多様な関数を持つ数式処理ソフトであり、ユーザーが作成したパッケージといわれる関数も多く公開されている。パッケージを集めたアプリケーションソフトも販売されている。MATLABが実装まで視野に入れた展開をしているのに対し、Mathematicaは大学や研究機関向けに豊富なアルゴリズムを取り揃えているという印象が強い。

1981年 Stephen Wolfram が前身の Symbolic manipulation program を商業リリース

↑理論物理、複雑系が専門(カリフォルニア工科大)

1988年 Mathematica 発売。Wolfram research 社

参考文献：

ランダウ=リフシツ：力学，東京図書(1974)... 訳本．剛体運動の基本．

増渕，川田：システムのモデリングと非線形制御，コロナ社(1996)... 様々なシステム例あり

北川，掘込，小川：自動制御工学，森北出版(2001)... 大学のテキストレベルの古典制御

吉川：古典制御論，昭晃堂(2004)... 上記よりも詳しい古典制御の本

美多，小郷著：システム制御理論入門，実教出版(1979)... ポイントをおさえた現代制御の本

児玉，須田著：システム制御のためのマトリクス理論，コロナ社(1978)... 詳しい行列の本

演習1 「数式処理ソフト Mathematica を使ってみよう」

Mathematica のカーネル

- ・起動後の最初の演算では「Kernel」を読み込みに少し時間がかかる。（作業領域の確保）
- ・計算したい式を入力，その後「Shift」＋「Enter」で実行。
- ・一度読み込んだら，その「Kernel」を shut down するまでは変数の値は記憶される。

厳密値と近似値

- ・「1」や「-3」は整数として取り扱われ，厳密解が返される。「1.」や「-3.00」は実数として取り扱われ，近似値が返される。
- ・厳密な値を近似値にするとき → $N[x, n]$ で， n 桁までの x の近似値。
- ・厳密値で計算すると時間がかかる。

簡単な関数

- ・関数は大文字から始まる．引数は[]でくくる。
- ・平方根： $\text{Sqrt}[x]$
- 三角関数： $\text{Sin}[x]$ ， $\text{Cos}[x]$ ， $\text{Tan}[x]$ ， $\text{ArcSin}[x]$ など（角度は radian）
- 対数： $\text{Log}[x]$ （自然対数）
- 指数関数： $\text{Exp}[x]$
- ・微分： $\text{D}[a*t^2 + 1, t]$
- 不定積分： $\text{Integrate}[a*t^2 + 1, t]$
- ・ラプラス変換： $\text{LaplaceTransform}[e^{-t}, t, s]$ ，
- 逆ラプラス変換： $\text{InverseLaplaceTransform}[s+1, s, t]$
- ・方程式を解く： $\text{Solve}[5x^2+1=0, x]$

1-1. 次の計算を実行してみましょう。

2^{100} , $2.^{100}$, $N[2^{100}, 2]$

$\text{Pi}/2$, $\text{Pi}/2.$, $N[\text{Pi}/2]$ （ただし， Pi は円周率 π を表す）

$\text{I}*\text{I}$, I^3 , $(1+\text{I})^2$ （ただし， I は虚数単位を表す）

1-2. 方程式を解いてみましょう。

$\text{Solve}[x^3 + x^2 + 1 = 0, x]$

$\text{Solve}[x^5 + x^2 + 1 = 0, x]$ （5次以上は解の公式がないので，解が出ません）

$\text{NSolve}[x^5 + x^2 + 1 = 0, x]$ （5次以上でも，数値解は得られます）

$\text{DSolve}[x'[t] + x[t] = 1, x[t], t]$ （微分方程式の一般解）

$\text{DSolve}[\{x'[t] + x[t] = 1, x[0] = 0\}, x[t], t]$ （初期値がある場合）

2. 運動方程式

良い制御を行うためには、制御対象の運動学的、力学的特徴を理解しておく必要がある。この特徴は、客観的な表現を用いて記述し、他者と共有できるようにすることが望ましい。これは、その後の問題点の原因究明や制御法の改善にも役立つだろう。

工学的には、制御対象の運動の特徴は微分方程式を用いて表現されることが多い。その代表的なものとして運動方程式(*equation of motion*)があげられる。運動方程式として、最もポピュラーな表現は、ニュートンの第二法則(*Newton's second law*)であろう。質点の質量と加速度をかけたものは、質点に加わる力の総和に等しいというものである。質量を m 、時刻 t での位置を $x(t)$ 、質点にかかる力の総和を $F(t)$ とすれば、

$$m\ddot{x}(t) = F(t) \quad (2-1)$$

で表わされる。

さて、複数の質点間の相互作用がある場合など、系(=システム)が複雑になると、質点にかかる力 $F(t)$ が容易に導出できないことがある。特に、回転運動と並進運動が混ざっているときなどは、導出が煩雑になることが多い。このような場合に、ラグランジュ方程式(*Lagrange's equations*)を用いると、運動方程式が比較的容易に得られる。

そこで、本章ではラグランジュ方程式について説明する。なお、ニュートンの運動方程式とラグランジュ方程式から得られる運動方程式は等価である。

2-1 一般化座標 (*generalized coordinate*)

ある系の位置をすべて決めるために必要な独立な量を自由度(*degree of freedom*)と言う。例えば、3次元空間に1つの質点が浮いている系ならば、自由度は3である。また、系の位置を決めるのに十分な任意の量を一般化座標という。たとえば、3次元空間に固定した直交座標系 $O\text{-}xyz$ の座標は、前述した系の一般化座標のひとつである。一般化座標は無数の組み合わせが選択できるので、対象とする問題で扱いやすい座標を選べばよい。

一般化座標の値をすべて与えれば、ある時刻での系の位置(=座標)は確定する。しかし、この時刻以降に、位置がどのように変化するかは定まらない。力学的な問題においては、ある時刻での位置と速度がすべて与えられると、その時刻以降の位置変化が予測できる。これは、運動方程式を解く際に、初期位置と初期速度が与えられれば、解が一意に決まることで理解できるだろう。

2-2 最小作用の原理 (*principle of least action, Hamilton's principle*)

系の位置を表わす一般化座標を $\mathbf{q} = (q_1, q_2, \dots, q_n)$ 、その導関数(一般化速度)を $\dot{\mathbf{q}} = (\dot{q}_1, \dot{q}_2, \dots, \dot{q}_n)$ とおく。力学系の運動は、1-1節で述べたように \mathbf{q} と $\dot{\mathbf{q}}$ で決まるので、系の運動はある関数 $L(\mathbf{q}, \dot{\mathbf{q}}, t)$ で特徴付けられる。この関数 L をラグランジアン(*Lagrangian*)という。

ここで、時刻 t_1 で位置が \mathbf{q}_1 にあり、時刻 t_2 で \mathbf{q}_2 になるとする。このとき、系は、

$$S = \int_{t_1}^{t_2} L(\mathbf{q}, \dot{\mathbf{q}}, t) dt \quad (2-2)$$

が最小の値をとるように変化する。これを**最小作用の原理**といい、古典力学の基礎的な原理である。

最小作用の原理を満たすための \mathbf{q} と $\dot{\mathbf{q}}$ の条件を求めてみよう。

まず、 $\mathbf{q} = \mathbf{q}(t)$ が作用 S を最小にする解だとしよう。すなわち、 \mathbf{q} が $\mathbf{q}(t)$ から少しでもずれると、 S

は大きくなる． q_1 から q_2 に変化する途中で $q(t)$ から微小な値 $\delta q(t)$ だけずれたとする．このとき，作用 S は，

$$\delta S = \int_{t_1}^{t_2} L(q + \delta q, \dot{q} + \delta \dot{q}, t) dt - \int_{t_1}^{t_2} L(q, \dot{q}, t) dt \quad (2-3)$$

だけ増加する．第一項目を $\delta q(t) = \delta \dot{q}(t) = 0$ の周りでテイラー展開すると，

$$\begin{aligned} \delta S &= \int_{t_1}^{t_2} \left(L(q, \dot{q}, t) + \frac{\partial L(q, \dot{q}, t)}{\partial q} \delta q + \frac{\partial L(q, \dot{q}, t)}{\partial \dot{q}} \delta \dot{q} + \dots \right) dt - \int_{t_1}^{t_2} L(q, \dot{q}, t) dt \\ &= \int_{t_1}^{t_2} \left(\frac{\partial L(q, \dot{q}, t)}{\partial q} \delta q + \frac{\partial L(q, \dot{q}, t)}{\partial \dot{q}} \delta \dot{q} \right) dt + O(\delta^2) \end{aligned} \quad (2-4)$$

である． $O(\delta^2)$ は， $\delta q(t)$ と $\delta \dot{q}(t)$ の 2 次以上の項であり，一次の項より十分小さいので無視する． S が $q(t)$ で極小になるためには， $\delta S = 0$ が必要である．この条件は，

$$\int_{t_1}^{t_2} \left(\frac{\partial L}{\partial q} \delta q + \frac{\partial L}{\partial \dot{q}} \delta \dot{q} \right) dt = 0 \quad (2-5)$$

となる．ここで， $L(q, \dot{q}, t)$ は単に L と表わしている．ところで，(2-5) の左辺第二項目は，部分積分をすると，

$$\int_{t_1}^{t_2} \frac{\partial L}{\partial \dot{q}} \delta \dot{q} dt = \left[\frac{\partial L}{\partial \dot{q}} \delta q \right]_{t_1}^{t_2} - \int_{t_1}^{t_2} \frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}} \right) \delta q dt \quad (2-6)$$

となる．運動の最初と最後では与えられた位置にいるので， $\delta q(t_1) = \delta q(t_2) = 0$ であり，右辺第一項目は 0 になる．(2-6) を (2-5) に代入すると，

$$\int_{t_1}^{t_2} \left(\frac{\partial L}{\partial q} - \frac{d}{dt} \frac{\partial L}{\partial \dot{q}} \right) \delta q dt = 0 \quad (2-7)$$

となる．任意の変位 $\delta q(t)$ に対して (2-7) が成立する，つまり， $q(t)$ が作用 S を最小にする条件は，

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = 0 \quad (2-8)$$

と得られる．この微分方程式 (2-8) がラグランジュ方程式である．

2-3 ラグランジアン (Lagrangian)

ラグランジアン L が決まれば，それを微分することで運動方程式が得られる．では，ラグランジアンはどのように求めるのだろうか？

まず，何の力も働いていない様な 3 次元空間で拘束されずに運動している一つの質点を考えてみよう．基準となる場所や時間を変えても，運動の性質は変わらない．このような系を表わすラグランジアンは，位置と時間に依存しないので， $L(\dot{q})$ である．さらに，速度の向きが逆でも，運動の性質は変わらないから， $L(\dot{q}^2)$ と書き表せる．詳しい説明は参考文献¹⁾を参照してほしいが，結局，このときのラグランジアンは，定数 $\times \dot{q}^2$ となる．この定数として，一般には質量 m を用いて $m/2$ を選ぶ．複数の質点が独立に運動している場合は，これを足し合わせ，

$$L = \sum_i \frac{1}{2} m_i \dot{q}_i^2 \quad (2-9)$$

となる。すなわち、ラグランジアンは運動エネルギーの総和と等しい。

さて、つぎに、例えば、質点間に働く引力のように、場に相互作用が存在している場合を考えよう。このとき、ラグランジアンはこの相互作用の分だけ修正される。

$$L = \sum_i \frac{1}{2} m_i \dot{q}_i^2 - U(\mathbf{q}) \quad (2-10)$$

修正分の U は、質点の位置だけに依存するものであり、ポテンシャルエネルギーと呼ばれる。ポテンシャルエネルギーには、位置エネルギーやバネの弾性エネルギーなどがある。

これを一般的に書き直すと、力学系のラグランジアンは、運動エネルギー T とポテンシャル U を用いて、

$$L = T - U \quad (2-11)$$

となる。

さて、(2-11)式において、ラグランジアンは運動エネルギーとポテンシャルの差になっており、力学的エネルギーの総和とは異なる。なぜ、ラグランジアンはエネルギーの引き算なのか？と、不思議に思うかもしれない。そこで、上記の定義から、力学的エネルギー保存則を導出することで、この式が矛盾していないことを確認してみよう。

ラグランジアンの全微分は、 q と \dot{q} の偏微分を用いて、

$$\frac{dL}{dt} = \frac{\partial L}{\partial q} \frac{dq}{dt} + \frac{\partial L}{\partial \dot{q}} \frac{d\dot{q}}{dt} \quad (2-12)$$

と書き表せる²⁾。右辺第一項目に(2-8)を代入すると、

$$\begin{aligned} \frac{dL}{dt} &= \frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}} \right) \frac{dq}{dt} + \frac{\partial L}{\partial \dot{q}} \frac{d\dot{q}}{dt} \\ &= \frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}} \frac{dq}{dt} \right) \end{aligned} \quad (2-13)$$

となる。さらに、(2-13)の両辺をまとめると、

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}} \frac{dq}{dt} - L \right) = 0 \quad (2-14)$$

である。これは、

$$\frac{\partial L}{\partial \dot{q}} \frac{dq}{dt} - L = \text{const.} \quad (2-15)$$

を表わしている。ところで、ラグランジアンが(2-10)で表わされる場合、

$$\frac{\partial L}{\partial \dot{q}} \frac{dq}{dt} = 2 \sum_i \left(\frac{1}{2} m_i \dot{q}_i \frac{dq_i}{dt} \right) (= 2T) \quad (2-16)$$

である。(2-11)と(2-16)を(2-15)に代入すると、

$$T + U = \text{const.} \quad (2-17)$$

となり，力学的エネルギーの保存則が導かれた。

2-4 ラグランジュの運動方程式

前節までは，外部とのエネルギーのやりとりがない保存系(孤立系)のラグランジュの運動方程式について説明した．摩擦などによって系のエネルギーが散逸する場合や，外力が加えられる場合に対しては，より一般的に，ラグランジュの運動方程式が，

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} + \frac{\partial D}{\partial \dot{q}} = F \quad (2-18)$$

で与えられる³⁾．ここで， D は摩擦などの散逸エネルギーである．減衰係数 c_i のダンパの場合は，

$$D = \sum_i \frac{1}{2} c_i \dot{q}_i^2 \quad (2-19)$$

で計算される． F は一般化座標の方向に働く一般化力(外力やトルク)である．

(2-18)を一般化座標 $q=(q_1, q_2, \dots, q_n)$ の各要素ごとに計算すると，加速度と位置・速度や力との関係を表わす運動方程式が得られる．

まとめ：ラグランジュの運動方程式

一般化座標を $q=(q_1, q_2, \dots, q_n)$ とする．系の運動エネルギーの総和が T ，ポテンシャルエネルギーの総和が U ，散逸エネルギーの総和が D ，一般化力が $F=(F_1, F_2, \dots, F_n)$ であるとき，運動方程式は，

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} + \frac{\partial D}{\partial \dot{q}_i} = F_i \quad (i=1, 2, \dots, n)$$

で得られる．ただし， $L=T-U$ である．

2-5 質点の運動方程式

質量 m の物体に外力 F が加えられたときの鉛直方向の運動を考えよう．重力加速度は g とし，前節までの説明にしたがって，ラグランジュの運動方程式を求める．

地表面から測った質点 m の位置を x とする．運動エネルギーは，

$$L = \frac{1}{2} m \dot{x}^2 \quad (2-20)$$

である．ポテンシャルエネルギーは，

$$U = mgx \quad (2-21)$$

である．したがって，ラグランジアンは，

$$L = \frac{1}{2} m \dot{x}^2 - mgx \quad (2-22)$$

となり,

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{x}} = m\ddot{x} \quad (2-23)$$

$$\frac{\partial L}{\partial x} = -mg \quad (2-24)$$

より, ラグランジュの運動方程式は,

$$m\ddot{x} + mg + 0 = F \quad (2-25)$$

と得られる. なお, このとき, ニュートンの運動方程式(2-1)は,

$$m\ddot{x} = -mg + F \quad (2-26)$$

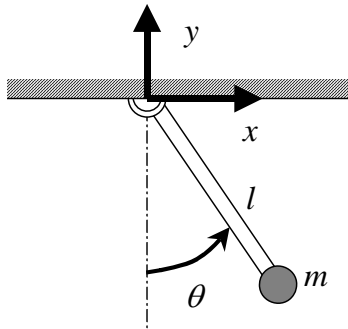
であり, 明らかにラグランジアンを用いて得られた運動方程式(2-25)と同じである.

参考文献:

1. ランダウリフシッツ: 力学, 東京図書(1974)
3. 小寺, 長谷川: 工学系学生のための常微分方程式, 森北出版(1996)
2. 増淵, 川田: システムのモデリングと非線形制御, コロナ社(1996)

演習 2 「運動方程式をたてる」

1. 運動方程式



図のように、長さ l で天井に取り付けられた質量 m の振り子がある。振り子は、平面内で左右に触れるとする。振り子の運動を表わす運動方程式を、下記の方法で求めなさい。なお、棒の質量は無視してよい。

1. ニュートンの運動方程式
2. ラグランジュの運動方程式

なお、図中の x, y, θ は参考として書いた例であり、座標系の取り方は自由とする。

2. Mathematica を使って微分をしてみましょう。

2-1. 次の関数を t で微分してみよう。

$$t^2, e^t, \log_e t, \frac{1}{t+1}, \sin(t), x(t), \frac{1}{2} m v(t)^2 \quad (m \text{ は定数})$$

2-2. $L = \frac{1}{2} m(\dot{x} + \dot{y})^2$ を \dot{x}, \dot{y} でそれぞれ偏微分してみよう。さらにこの答えを時間で微分してみよう。

Mathematica のコマンド

- 微分 : $D[f(t), t]$... $f(t)$ を t で微分する
- 関数 : $x[t]$ x は t の関数である
- 関数の微分 : $x'[t]$ x の t による一階微分

3. システムの表現

本講義で取り扱うのは、時不変で連続な線形システムである。時不変なシステムとは、質量や粘性係数などのシステムパラメータが時間によって変化しないシステムである。一方、連続システムであるとは、入力や出力といった信号が連続時間的に変化することを表す。線形システムでは、入力が2倍になれば、出力も2倍になるという線形関係が成立する。

連続時間システムの状態や応答の変化の様子は、多くの場合、微分方程式で表すことができる。

例：ばねダンパ系

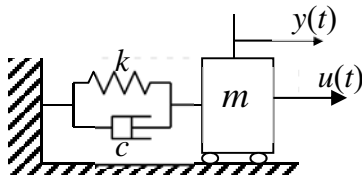


図 2-1 ばねダンパ系

質量が $m[\text{kg}]$ の物体が壁にばね定数 k のばね、粘性係数 c のダッシュポットで取り付けられている。このとき、物体に力 $u(t)$ を加えると、物体のつり合いからの位置 $y(t)$ は、運動方程式、

$$m\ddot{y}(t) + c\dot{y}(t) + ky(t) = u(t) \quad (3-1)$$

にしたがって変化する。

3-1 ラプラス変換 (Laplace transformation) と逆ラプラス変換 (inverse Laplace transformation)

システムの状態や応答の変化を表す微分方程式を解けば、応答や状態の様子を知ることができる。複雑なシステムを表すモデルは、高階の微分方程式になったり、連立微分方程式になったりする。一般に、これらの微分方程式は解くのが大変である。そこで、ラプラス変換を用いる方法がある。ラプラス変換は微分方程式を代数方程式に変換するので、解析やコントローラ設計が容易になるという利点もある。

ラプラス変換は、時間関数 $f(t)$ を複素関数 $F(s)$ に変換する。時間関数 $f(t)$ に対して、

$$F(s) = \int_0^{\infty} f(t)e^{-st} dt \quad (= \mathcal{L}(f(t))) \quad (3-2)$$

の右辺が存在するような複素数 s が存在するとき、 $F(s)$ を $f(t)$ のラプラス変換形という。ラプラス変換の変換記号として、 $\mathcal{L}(\cdot)$ を用いることにする。

定義(3-2)に従うと、関数の微分について、

$$\mathcal{L}\left(\frac{df(t)}{dt}\right) = sF(s) - f(0) \quad (3-3)$$

$$\mathcal{L}\left(\frac{d^2f(t)}{dt^2}\right) = s^2F(s) - sf(0) - \dot{f}(0) \quad (3-4)$$

$$\mathcal{L}\left(\frac{d^nf(t)}{dt^n}\right) = s^nF(s) - s^{n-1}f(0) - s^{n-2}\dot{f}(0) \cdots - f^{(n-1)}(0) \quad (3-5)$$

が成り立つ。

ラプラス変換の逆変換、すなわち、複素関数 $F(s)$ を時間関数 $f(t)$ に戻す変換を逆ラプラス変換という。その定義は、

$$f(t) = \frac{1}{2\pi j} \int_{c-j\infty}^{c+j\infty} F(s)e^{st} ds \quad (= \mathcal{L}^{-1}(F(s))) \quad (3-6)$$

である。cはある実数値であり、jは虚数単位である。cの条件や逆ラプラス変換の存在条件についての説明を省略するが、数学的に興味のある人は参考文献(吉川著：古典制御論など)を参照して欲しい。線形システムでよく用いる関数についてはラプラス変換が分かっているから、その逆の関係であることを理解していれば、基本的な線形システムの解析には対応出来るだろう。

3-2 伝達関数 (transfer function)

ラプラス変換を用いて運動方程式(3-1)を変換してみよう。簡単な例として、 $\dot{x}(0) = x(0) = 0$ として考えよう。 $Y(s) = \mathcal{L}(y(t))$, $U(s) = \mathcal{L}(u(t))$ とする。(3-3)(3-4)を用いると、

$$ms^2Y(s) + csY(s) + kY(s) = U(s) \quad (3-7)$$

となる。微分演算子がなくなり、 $U(s)$ と $Y(s)$ の関係が代数方程式で表されていることがわかる。

システムの入力と出力をラプラス変換した関数の比 $Y(s)/U(s)$ をシステムの伝達関数と呼ぶ。ただし、伝達関数ではシステムの信号の初期値をすべて0で考える。

たとえば、システム(3-1)の伝達関数は、(3-7)式より、

$$\frac{Y(s)}{U(s)} = \frac{1}{(ms^2 + cs + k)} \quad (3-8)$$

となる。

3-3 ブロック線図 (block diagram)

システムは、複数の構成要素(サブシステム)で構成されていることが多い。このとき、サブシステム間での信号の伝達を表すのに、ブロック線図がよく用いられる。

ブロック線図では、システムは四角で囲まれ、信号は矢印で表される。

例：フィードバック制御系

左図で表されるシステムは、出力 $Y(s)$ が入力信号に影響していることから、フィードバック制御系と呼ばれる。

このシステムにおいて、外部入力 $R(s)$ から出力 $Y(s)$ への伝達関数はどうなるだろうか？

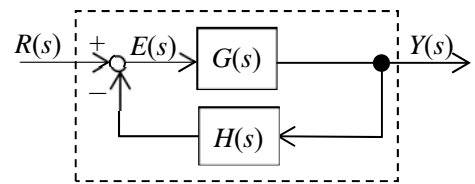


図 2-2 フィードバック制御系

ブロック線図から、次の関係が読み取れる。

$$Y(s) = G(s)E(s) \quad (3-9)$$

$$E(s) = R(s) - H(s)Y(s) \quad (3-10)$$

この代数方程式を連立させて $E(s)$ を消去すると、

$$\frac{Y(s)}{R(s)} = \frac{G(s)}{1 + H(s)G(s)} \quad (3-11)$$

という伝達関数が得られる.

このようにサブシステムが伝達関数で表わされていると、システム全体の伝達関数は代数的な計算によって求めることができる.

3-4 伝達関数から状態方程式へ

古典制御で扱うシステムの入力は1つ、出力も1つ(*single-input-single-output*, SISO)であった. また、システムの入出力比を表す伝達関数にもとづいて解析・設計を行うため、システム内部の状態は考慮していない.

しかし、科学技術や工業の発展の中で、複数の入力や出力を持つ複雑なシステムの制御問題を考える必要が生じた. 現代制御論はこのような多入力多出力(*multiple-input-multiple-output*, MIMO)システムを取り扱うのに適した理論である. 現代制御論ではシステムのモデルを状態方程式(*state equation*)で表す. 状態方程式とはシステムの状態を定義し、それぞれの状態の時間変化を表す一階の微分方程式を連立させたモデルである. 状態方程式を用いることで、システム内部の状態を考慮した解析や制御系設計を行うことができる.

例：伝達関数から状態方程式へ
つぎの伝達関数を考えよう.

$$\frac{Y(s)}{U(s)} = \frac{c}{s^2 + as + b} \quad (3-12)$$

これを変形し、初期値を0として逆ラプラス変換を行うと、

$$(s^2 + as + b)Y(s) = U(s)$$

$$\ddot{y}(t) + a\dot{y}(t) + by(t) = u(t) \quad (3-13)$$

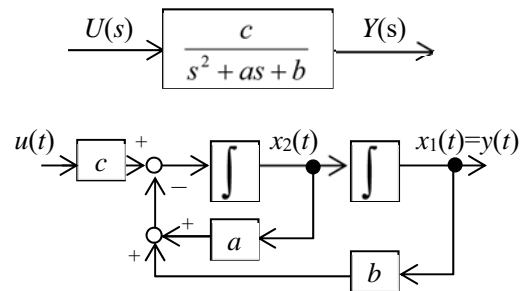


図 3-1 伝達関数と状態空間表現

という2階の微分方程式になる. ここで、つぎのように状態ベクトル \mathbf{x} を定義しよう. なお、以下では、 $x(t)$ が時間関数であることが明らかな場合は、単に x と表記する.

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y \\ \dot{y} \end{pmatrix} \quad (3-14)$$

このとき、(3-13)式はつぎのように書き直すことができる.

$$\dot{x}_2 + ax_2 + bx_1 = cu \quad (3-15)$$

また、明らかに、

$$\dot{x}_1 = x_2 \quad (3-16)$$

であるから、(3-15)(3-16)式をあわせて行列表現を用いると、

$$\dot{\mathbf{x}} = \begin{pmatrix} 0 & 1 \\ -b & -a \end{pmatrix} \mathbf{x} + \begin{pmatrix} 0 \\ c \end{pmatrix} u$$

$$y = \begin{pmatrix} 1 & 0 \end{pmatrix} \mathbf{x} \quad (3-17)$$

が得られる。(3-17)式を状態空間表現という。

一般に、状態方程式は係数行列に A, B, C を用いて、

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x}\end{aligned}\tag{3-18}$$

と書き表す。状態数が n 、入力数が m 、出力数が p である場合、係数行列のサイズは、 A は $n \times n$ 、 B は $n \times m$ 、 C は $p \times n$ となる。状態方程式(3-18)で表現することで、状態数 n や入力数 m 、出力数 p がいくつであっても、統一的にシステムを取り扱うことができる。

3-5 状態方程式と伝達関数行列

今度は、逆に状態方程式(3-18)から、伝達関数行列を求めてみよう。スカラー関数のときと同様に、(3-18)式の両辺をラプラス変換すると、

$$\begin{aligned}s\mathbf{X}(s) &= \mathbf{A}\mathbf{X}(s) + \mathbf{B}\mathbf{U}(s) \\ \mathbf{Y}(s) &= \mathbf{C}\mathbf{X}(s)\end{aligned}\tag{3-19}$$

となる。ただし、 $\mathbf{X}(s)$ は $\mathbf{x}(t)$ の各要素をラプラス変換したベクトルである。 A が行列であることに注意して、(3-19)式をまとめると、

$$\mathbf{Y}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s)\tag{3-20}$$

となる。ここで、 \mathbf{I} は A と同じサイズの単位行列である。 $\mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}$ は伝達関数行列と呼ばれる。

3-6 相似変換

3-4 節の例で示したように、伝達関数行列から状態方程式を導くことを、実現(*realization*)という。一つの伝達関数を実現するための状態の選び方は一意ではなく、得られる状態方程式は無数に存在する。例えば、以下に示すような相似の関係にある状態方程式である。

任意の正則行列 T を考える。(3-18)式の両辺に左から T をかける。

$$T\dot{\mathbf{x}} = T\mathbf{A}\mathbf{x} + T\mathbf{B}\mathbf{u}\tag{3-21}$$

$T^{-1}T = \mathbf{I}$ であるから、

$$T\dot{\mathbf{x}} = T\mathbf{A}T^{-1}T\mathbf{x} + T\mathbf{B}\mathbf{u}\tag{3-22}$$

と変形できる。ここで、正則行列 T を用いて座標変換した状態を

$$\tilde{\mathbf{x}} = T\mathbf{x}\tag{3-23}$$

とおく。すると、(3-22)式は、

$$\dot{\tilde{\mathbf{x}}} = (T\mathbf{A}T^{-1})\tilde{\mathbf{x}} + (T\mathbf{B})\mathbf{u}\tag{3-24}$$

となり、同様に出力も

$$\mathbf{y} = (\mathbf{C}T^{-1})\tilde{\mathbf{x}}\tag{3-25}$$

と表現できる．ところで， \mathbf{x} と $\tilde{\mathbf{x}}$ は一対一の関係(3-23)にある．例えば， $\tilde{\mathbf{x}}=0$ となるのは， $\mathbf{x}=0$ のときのみである．よって，

$$\begin{aligned}\dot{\tilde{\mathbf{x}}} &= \tilde{\mathbf{A}}\tilde{\mathbf{x}} + \tilde{\mathbf{B}}\mathbf{u} \\ \mathbf{y} &= \tilde{\mathbf{C}}\tilde{\mathbf{x}} \\ \tilde{\mathbf{A}} &= \mathbf{T}\mathbf{A}\mathbf{T}^{-1}, \tilde{\mathbf{B}} = \mathbf{T}\mathbf{B}, \tilde{\mathbf{C}} = \mathbf{C}\mathbf{T}^{-1}\end{aligned}\tag{3-26}$$

と(3-18)式は相似の関係にあるといい，このような変換を相似変換，もしくは等価変換という．(3-26)式の伝達関数行列を求めてみると，

$$\begin{aligned}\mathbf{Y}(s) &= \tilde{\mathbf{C}}(s\mathbf{I} - \tilde{\mathbf{A}})^{-1}\tilde{\mathbf{B}}\mathbf{U}(s) \\ &= \mathbf{C}\mathbf{T}^{-1}(s\mathbf{I} - \mathbf{T}\mathbf{A}\mathbf{T}^{-1})^{-1}\mathbf{T}\mathbf{B}\mathbf{U}(s) \\ &= \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s)\end{aligned}\tag{3-27}$$

であり，状態方程式(3-26)は伝達関数(3-20)の実現である．

演習3「ラプラス変換，逆ラプラス変換をおこなう」（Mathematica の場合）

3-1. ラプラス変換してみましょう．

ステップ関数(3, $a = \text{一定値}$)，ランプ関数 ($t, 3t$)，

三角関数 ($\sin 2t, \cos wt$)，指数関数 (e^t, e^{-at})，その他 ($te^{-at}, e^{-at} \sin 2t$)

注： e^{-at} は，Exp[-at]と入力する．掛け算は，* を使うか，空白をあければよい．

3-2. 逆ラプラス変換してみましょう．

1, $\frac{1}{s}$, $\frac{K}{s+T}$, $\frac{w}{s+w^2}$, $\frac{1}{s^2+s+1}$ など

注： s^2 は s^2 と入力する．また，Mathematica では大文字と小文字は区別される．

3-3. 次のシステムの状態方程式を求めなさい．

$$1) Y(s) = \frac{1}{s^3 + s^2 + s + 2} U(s)$$

$$2) Y(s) = \frac{s+1}{s^2 + s + 1} U(s)$$

3-4. 状態方程式(3-17)について $G(s) = C(sI - A)^{-1}B$ を計算して，元の伝達関数(3-12)が求められることを確かめなさい．

4. システムの時間応答

微分方程式で表わされたシステムにある入力加わった場合の応答を調べるためには、その微分方程式を解けばよい。例えば、システム(3-1)に一定の入力が加わった場合、二階の非同次微分方程式(*nonhomogeneous differential equation*)になる。この解は、 $u = 0$ とした場合の同次微分方程式(*homogeneous differential equation*)の一般解(*general solution*)と、非同次微分方程式のある特殊解(*particular solution*)の和として得ることができる。しかし、高次のシステムになると微分方程式を直接解くのは困難であるので、ラプラス変換を用いて解く方法がある。ラプラス変換を行って伝達関数としてシステムを表わし、部分分数展開などの代数的な計算を行った後、逆ラプラス変換することで解を得る。

状態方程式は一階の微分方程式であるので、その応答は微分方程式の解として直接得ることができる。

4-1 状態方程式の一般解

状態空間表現を用いると高次のシステムも一般的に状態方程式(3-18)で表わされる。このうちの状態の遷移を表わす方程式(狭義の意味での状態方程式),

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \quad (4-1)$$

の解 $\mathbf{x}(t)$ はつぎのように得られる。

$$\mathbf{x}(t) = e^{\mathbf{A}t} \mathbf{x}(0) + \int_0^t e^{\mathbf{A}(t-\tau)} \mathbf{B}\mathbf{u}(\tau) d\tau \quad (4-2)$$

ただし、 $\mathbf{x}(0)$ は状態の初期値であり、 $e^{\mathbf{A}t}$ はつぎの式で定義される指数関数行列である。

$$e^{\mathbf{A}t} = \mathbf{I} + \frac{\mathbf{A}t}{1!} + \frac{(\mathbf{A}t)^2}{2!} + \frac{(\mathbf{A}t)^3}{3!} + \cdots + \frac{(\mathbf{A}t)^n}{n!} + \cdots \quad (4-3)$$

(4-3)式で定義された指数関数行列には、つぎのような性質が成り立つ。

$$1) \quad e^{\mathbf{A}0} = \mathbf{I} \quad (4-4)$$

$$2) \quad (e^{\mathbf{A}t})^{-1} = e^{-\mathbf{A}t} \quad (4-5)$$

$$3) \quad e^{\mathbf{A}t} = \mathcal{L}^{-1}((s\mathbf{I} - \mathbf{A})^{-1}) \quad (4-6)$$

$$4) \quad \frac{d}{dt} e^{\mathbf{A}t} = \mathbf{A}e^{\mathbf{A}t} \quad (4-7)$$

$$5) \quad \mathbf{A} \int_0^t e^{\mathbf{A}\tau} d\tau = e^{\mathbf{A}t} - \mathbf{I} \quad (4-8)$$

(4-2)式の証明：

まず、(4-1)式において $\mathbf{u} = \mathbf{0}$ とした同次微分方程式の一般解を求める。任意のベクトル \mathbf{c} を用いて、 $\mathbf{x} = e^{\mathbf{A}t} \mathbf{c}$ を仮定する。これを(4-1)式に代入すると、(4-7)式より、

$$\begin{aligned} \frac{d}{dt}(e^{At}c) &= Ae^{At}c \\ &= A(e^{At}c) \end{aligned} \tag{4-9}$$

であるから、 $x=e^{At}c$ が(4-1)式の解であることが確認できる。

つぎに、 $u \neq 0$ のときの特解を求める。特解を $x = e^{At}z(t)$ と仮定する。これが(4-1)の解であるためには、

$$\begin{aligned} Ae^{At}z + e^{At}\dot{z} &= Ae^{At}z + Bu \\ e^{At}\dot{z} &= Bu \end{aligned} \tag{4-10}$$

より、

$$\begin{aligned} \dot{z} &= e^{-At}Bu \\ z &= \int_0^t e^{-A\tau}Bu(\tau)d\tau \end{aligned} \tag{4-11}$$

である。したがって、もとの微分方程式の解は同次微分方程式の一般解 $e^{At}c$ と非同次微分方程式の特解の和、

$$x(t) = e^{At}c + e^{At} \int_0^t e^{-A\tau}Bu(\tau)d\tau \tag{4-12}$$

である。ここで状態の初期条件 $x(0)$ を満たすように c を求めると、(4-4)式より、 $c = x(0)$ と定まり、(4-2)式が得られる。

4-2 システムの極と時間応答

伝達関数の分母をシステムの特性多項式と呼ぶ。そして、特性多項式=0 とした s の方程式をシステムの特性方程式(*characteristic equation*)と呼ぶ。この特性方程式の解をシステムの極(*pole*)と呼ぶ。極はシステムの応答を知るために重要な値であり、伝達関数の次数と同じだけ存在する。

伝達関数で表現されたシステムの過渡応答は、極の値によって変わる。まず、例を見てみよう。

例：一次系の過渡応答

伝達関数 $G(s) = \frac{1}{s+a}$ の極は、 $s=-a$ であり、ステップ応答は、 $y(t) = 1 - e^{-at}$ である。 $a=1, 5$ のときのステップ応答のグラフを図 5-1 に示す。 $a=5$, すなわち、極が $s=-5$ であるときのほうが、極が -1 のときよりも速く定常値 1 に収束する。

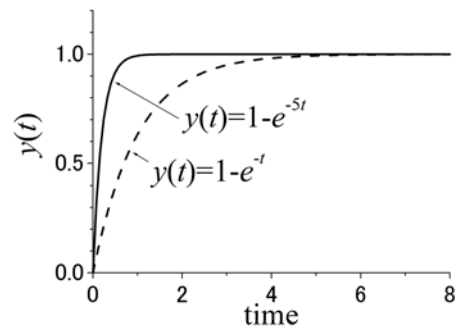


図 4-1 一次系のステップ応答

2次以上のシステムでは、極に複素数を含むことがある。極を複素平面にプロットしたときの位置と、過渡応答の特徴との関係をまとめておこう。

まず、実部が負である極は複素平面の左半平面にプロットされる。上記の例で示したように、複素平面の左側にあるほど(−∞に近づくほど)、収束は速い。また、複素数の極は、必ず共役複素数として現れる。共役複素数は、実軸対称にプロットされる。実軸から離れるほど、過渡応答は振動

的になる．安定限界である純虚数の極は，虚軸上にプロットされる．このときはステップ応答はある振幅で振動を続ける．

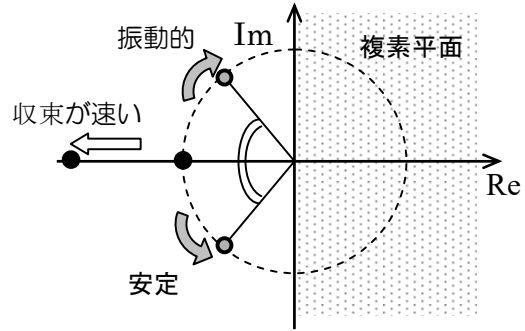


図 4-2 複素平面での極の配置

4-3 特性方程式と極とモード

状態方程式(3-18)で表わされるシステムの初期値応答を見てみよう．初期値 $\mathbf{x}(0)$ ，入力 $\mathbf{u}(t)=0$ としてラプラス変換すると，

$$\begin{aligned} s\mathbf{X}(s) - \mathbf{x}(0) &= A\mathbf{X}(s) \\ \mathbf{X}(s) &= (sI - A)^{-1} \mathbf{x}(0) \end{aligned} \quad (4-13)$$

となる．このシステム(3-18)の特性方程式はどうなるだろうか？ $(sI - A)$ の逆行列は，

$$(sI - A)^{-1} = \frac{1}{|sI - A|} \begin{pmatrix} \tilde{a}_{11}(s) & \cdots & \tilde{a}_{1n}(s) \\ \vdots & \ddots & \vdots \\ \tilde{a}_{n1}(s) & \cdots & \tilde{a}_{nm}(s) \end{pmatrix} \quad (4-14)$$

という形である．ただし， $\tilde{a}_{ij}(s)$ は s の $n-1$ 次以下の多項式である．(4-14)の右辺の行列を余因子行列といい， $\text{adj}(sI - A)$ と書く．(4-14)式の分母は $|sI - A|$ であるから，特性方程式は，

$$|sI - A| = 0 \quad (4-15)$$

である．ここで，システムの n 個の極を $p_1 \cdots p_n$ とおく．すなわち， $|sI - A| = (s - p_1)(s - p_2) \cdots (s - p_n)$ である．このとき，行列 $(sI - A)^{-1}$ の全ての要素は，

$$\frac{\tilde{a}_{ij}(s)}{(s - p_1)(s - p_2) \cdots (s - p_n)} = \frac{c_{ij1}}{(s - p_1)} + \frac{c_{ij2}}{(s - p_2)} + \cdots + \frac{c_{ijn}}{(s - p_n)} \quad (4-16)$$

と部分分数展開できる． c_{ijk} は定数である．(4-13)式の右辺は， $(sI - A)^{-1}$ に係数ベクトル $\mathbf{x}(0)$ をかけたものなので， $\mathbf{X}(s)$ は(4-16)の線形和となる．したがって， $\mathbf{X}(s)$ を逆ラプラス変換して得られる $\mathbf{x}(t)$ は，(4-16)を逆ラプラス変換して得られる $e^{p_1 t}$ ， $e^{p_2 t}$ ， \cdots ， $e^{p_n t}$ の線形和になる．

この $e^{p_1 t}$ ， $e^{p_2 t}$ ， \cdots ， $e^{p_n t}$ をシステム固有のモード(mode)と呼ぶ．モードは極の値で決まることから，システムの応答は極の値に大きく左右されることがわかる．個々のモードのふるまいは，図 4-2 で示した傾向に従う．たとえば，一つ(一対)でも原点に近い極があると，そのモードのために応答の収束は遅くなる．

同様に，入力，

$$\mathbf{U}(s) = \frac{1}{q(s)} \begin{pmatrix} r_1(s) \\ r_2(s) \\ \vdots \\ r_m(s) \end{pmatrix} \quad (4-17)$$

に対する応答を考えてみよう． $q(s)$ ，および $r_i(s)$ ， $i=1,2,\dots,m$ は s の任意の多項式とする．このときの応答は，

$$\mathbf{X}(s) = \frac{1}{|s\mathbf{I} - \mathbf{A}|q(s)} \begin{pmatrix} \tilde{a}_{11}(s) & \cdots & \tilde{a}_{1n}(s) \\ \vdots & \ddots & \vdots \\ \tilde{a}_{n1}(s) & \cdots & \tilde{a}_{nn}(s) \end{pmatrix} \mathbf{B} \begin{pmatrix} r_1(s) \\ \vdots \\ r_m(s) \end{pmatrix}$$

$$\mathbf{Y}(s) = \frac{1}{|s\mathbf{I} - \mathbf{A}|q(s)} \mathbf{C} \begin{pmatrix} \tilde{a}_{11}(s) & \cdots & \tilde{a}_{1n}(s) \\ \vdots & \ddots & \vdots \\ \tilde{a}_{n1}(s) & \cdots & \tilde{a}_{nn}(s) \end{pmatrix} \mathbf{B} \begin{pmatrix} r_1(s) \\ \vdots \\ r_m(s) \end{pmatrix}$$

という形になる．分母は $|s\mathbf{I} - \mathbf{A}|$ と入力分母 $q(s)$ のかけ算あり，分子は係数行列と $\text{adj}(s\mathbf{I} - \mathbf{A})$ のかけ算である．したがって，出力 $\mathbf{Y}(s)$ の各要素は，(4-16)同様に，

$$Y_i(s) = \frac{c'_{i1}}{(s-p_1)} + \frac{c'_{i2}}{(s-p_2)} + \cdots + \frac{c'_{in}}{(s-p_n)} + \frac{d_i(s)}{q(s)}, \quad i = 1, \dots, p \quad (4-18)$$

と部分分数展開できる．(4-18)右辺 n 項目まではシステム固有のモードであり， $n+1$ 項目は入力モードである．

例えば，安定なシステムに入力として正弦波を加えると，安定なシステム固有のモードは 0 に収束して入力モードのみが残り，定常応答は正弦波状の振動となる．

演習4「システムの時間応答をみる」（MATLAB/Simulink の場合）

4-1. 行列の指数関数について、次の計算を試みましょう。ただし、 $A = \begin{pmatrix} 0 & 1 \\ -1 & -1 \end{pmatrix}$ とします。

- 1) e^{At} を計算しましょう。また、 $t=0$ とすると、いくつになりますか？
- 2) e^{At} と $I + \frac{At}{1!} + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \dots + \frac{(At)^n}{n!} + \dots$ を比較しなさい。ただし、 $t=0.1$ とします。
誤差が 5% 以下になるのは、 n がいくつのときでしょうか？
- 3) $(e^{At})^{-1}$ と e^{-At} が等しいことを確かめなさい。
- 4) $\frac{d}{dt}e^{At}$ と Ae^{At} が等しいことを確かめなさい。
- 5) e^{At} と $(sI - A)^{-1}$ の逆ラプラス変換した関数が等しいことを確かめなさい。

4-2. $\ddot{y} + \dot{y} + y = u$ で表されるシステムの応答 $y(t)$ をつぎの 3 種類の方法で求めなさい。ただし、 $u(t)=1$, $\dot{y}(0) = y(0) = 0$ とする。

- i) 二階の微分方程式を解く
- ii) $u(s)$ から $y(s)$ の伝達関数を求めてから、逆ラプラス変換を用いる
- iii) 状態方程式にしてから求める

4-3. 極とステップ応答

二次系 $\frac{1}{s^2 + as + b}$ の極は、 a と b の値によって、異なる 2 つの実数解、重解、共役複素解のいずれかになる。定数 a, b をつぎのように変えて、それぞれのステップ応答のグラフを描いて確認しなさい。

- 1) 実数解 : $a=4, b=1$
- 2) 重解 : $a=2, b=1$
- 3) 共役複素解 : $a=1, b=1$

4-4. 4-3 の応答を MATLAB を用いて計算してみましょう。

5. システムの安定性(stability)

制御系に要求される最も基本的な性質は安定なことである。システムが不安定であると、状態や出力の値が発散してしまう。これは、振動が徐々に大きくなっていく現象や可動範囲の超過などを表し、システムの故障や事故を引き起こす。

「安定」の直感的なイメージは、外部からの入力が無くなったときに、システムの状態がある一定値(=平衡点)に落ち着くということである。たとえば、やじろべえを指で弾くと振動するが、しばらく時間が経つと再び最初のバランスをとった姿勢に戻る。これが安定である。

安定性の厳密な定義は複数存在する。入出力安定性、内部安定性、リアプノフ安定性などである。ただし、古典制御で対象としている1入力1出力の線形システムでは、これらの定義はいずれも等価な条件となり、以下で述べる極の値に帰着する。

5-1 極 (pole) と安定性

例:1次系の極と安定性

図4-1で表される一次系のインパルス応答を調べよう。

インパルス入力をラプラス変換した $U(s)=1$ を代入してから、逆ラプラス変換すると、出力(応答)は、 $y(t)=be^{-at}$ となる。この定常値は、

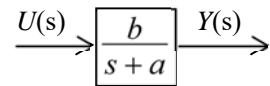


図4-1 一次系

$$\lim_{t \rightarrow \infty} y(t) = 0 \quad , a > 0$$

$$\lim_{t \rightarrow \infty} y(t) = \infty \quad , a < 0$$

であり、 a が正であればインパルス応答は0に収束する。すなわちシステムは安定である。ところで、伝達関数 $G(s) = \frac{b}{s+a}$ の極は $s=-a$ である。つまり、極が負であるときにシステムは安定となる。

4-3節で述べたように、高次のシステムの応答は、各極に対応したモードの線形和である。したがって、1つでも不安定なモードがあると応答は発散する。逆に、全てのモードがそれぞれ0に収束するならば、和も0に収束する。すなわち、システムが安定であるための必要十分条件は、

$$\text{システムが安定} \Leftrightarrow \text{全ての極の実部が負}$$

と表される。特に実部が0のときは、安定限界と呼ぶこともある。

5-2 状態空間表現における安定性

3章で示したように、一般的な状態方程式、

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{Ax} + \mathbf{Bu} \\ \mathbf{y} &= \mathbf{Cx} \end{aligned} \tag{3-18}$$

の伝達関数行列は、

$$\mathbf{Y}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1} \mathbf{B}\mathbf{U}(s) \tag{3-20}$$

である。この伝達関数行列の特性方程式は、

$$|sI - A| = 0 \quad (4-15)$$

である。したがって、この特性方程式の解、すなわちシステムの極のすべての実部が負であれば、システム(3-18)は安定となる。

ところで、(4-15)の解は、行列 A の固有値と等しい。このことから、システム(3-18)が安定であるための必要十分条件は、 A の固有値の全ての実部が負である、と言い換えることができる。

5-3 安定判別法 (*stability criterion*)

システムの安定性を調べるには、全ての極の実部の符号を調べればよい。しかし、5次以上の方程式の一般解が存在しないことなどから、特性方程式の次数があがると、極を求めることは難しくなる。そこで、方程式を解くことなく、安定性を調べる方法として、ラウスの安定判別法やフルビッツの判別法がある。これらは制御のテキストなどには必ず掲載されている手法であるので、ここでは省略する。

一方、近年では、数式処理ソフトウェアの発達により、方程式の数値解を求めることが容易になった。したがって、高次のシステムでも直接、極を計算して安定性を調べるのが可能である。

演習5 「システムの極と時間応答をみる」（Mathematica の場合）

1. 伝達関数がつぎのように与えられるシステムの極を求めてみましょう.

$$\frac{1}{s+2}, \frac{1}{s^2+2}, \frac{1}{s^2+s+2}, \frac{1}{s^3+s^2+s+2}, \frac{1}{s^4+s^3+s^2+s+2}, \frac{1}{s^5+s^4+s^3+s^2+s+2},$$

$$\frac{1}{s+a}, \frac{1}{s^2+as+b}$$

2. 安定判別

特性方程式が $s^3+s^2+s-1=0$ の場合, 安定か不安定か? (極の実部の符号を調べなさい)

特性方程式が $s^5+s^4+s^3+s^2+s+1=0$ の場合はどうか?

3. $\dot{x} = Ax$ というシステムの係数行列が,

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

であるとき, 極を求めなさい. ($|sI - A| = 0$ を解いて求める)

Mathematica のコマンド

有理関数の分母を抜き出す

`Denominator[a(s)/b(s)]`

方程式を解く

`Solve[b(s)=0, s]`... s の方程式 $b(s)=0$ を s について解く

`NSolve[b(s)=0, s]`... s の方程式 $b(s)=0$ を s について数値的に解く

`FindRoot[b(s)=0, {s, s0}]`... s の方程式 $b(s)=0$ について $s=s_0$ の近傍から解を探索する
(一つの解を見つけるだけ. 一度に全ての解は得られない)

実部と虚部

`Re[x]`... 変数 x の実数部分を取り出す

`Im[x]`... 変数 x の虚数部分を取り出す

行列

`Det[A]`... A の行列式を計算する

`Inverse[A]`... A の逆行列を計算する

`IdentityMatrix[n]`... $n \times n$ の単位行列

6. 出力フィードバック制御

この章では、1入力1出力系に対する出力フィードバック制御について復習しよう。

6-1 フィードバック制御による安定化

全ての極の実部が負であればシステムは安定である。不安定な極を持つシステムに対して、何らかの制御を行って極を安定な値に変更することを安定化(*stabilization*)という。

例 6-1: 一次系のフィードバック制御

図 6-1 の上図で表される一次系,

$$\frac{Y(s)}{R(s)} = \frac{1}{s - 0.5} \quad (6-1)$$

の極は+0.5 であり、不安定である。このシステムに対して、図 6-1 の下図のようにフィードバック制御をほどこす。つまり、出力 $y(t)$ を用いて制御対象への制御入力 $u(t)$ を

$$u(t) = r(t) - Ky(t) \quad (6-2)$$

とする。このフィードバック制御を行うと、外部入力 $R(s)$ から出力 $Y(s)$ までの伝達関数は、

$$\frac{Y(s)}{R(s)} = \frac{1}{s - 0.5 + K} \quad (6-3)$$

となる。フィードバック系の極は $0.5 - K$ であるから、 $K > 0.5$ とすれば制御系は安定となる。

フィードバック則(6-2)は、現時刻 t の情報だけを用いている。このような制御を静的(*static*)な制御という。一方、積分制御は過去の値を積分する、つまり現時刻の情報以外に依存する。このような制御を動的な制御と呼ぶ。

静的な出力フィードバック制御では安定化できないシステムがある。例えば、ダンパ要素がない純粋なばね質量系において、出力が質量の位置である場合、いくら出力フィードバック制御を行っても質量の運動を止めることはできない。このようなシステムを安定化するためには、動的な制御や第 8 章で述べる状態フィードバック制御などの別の方法を用いる必要がある。

6-2 定常応答の改善

制御の目的はいくつかあるが、安定化のつぎに与えられる主な要求の一つに、目標値追従がある。システムの入力 $R(s)$ を目標値 $R(s)$ に一致させるための制御系を目標値追従制御系、もしくはサーボ系(*servo system*)という。このとき、出力と目標値との差を偏差(*error*)と呼ぶ。一般に、制御対象が持つ動特性のため、制御開始時から偏差を完全に 0 にすることは難しい。そこで、まず、十分時間が経ったときの偏差 (= 定常偏差, *steady-state error*) を 0 にすることが第一の目標となる。

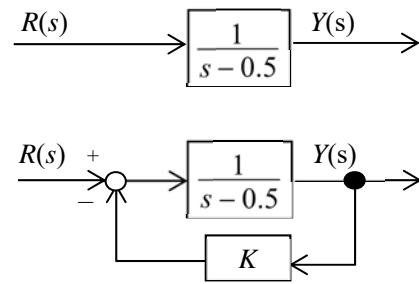


図 6-1 フィードバック制御系 1

例 6-2: 一次系のフィードバック制御の定常偏差

図 6-1 の例において、目標入力 $R(s)$ が大きさ r のステップ関数であるとき、定常偏差は最終値の定理より、

$$\begin{aligned} \lim_{t \rightarrow \infty} e(t) &= \lim_{t \rightarrow \infty} (r(t) - y(t)) \\ &= \lim_{s \rightarrow 0} s(R(s) - Y(s)) \\ &= \lim_{s \rightarrow 0} s \left(\frac{s - 1.5 + K}{s - 0.5 + K} \right) R(s) \\ &= \frac{-1.5 + K}{-0.5 + K} r \end{aligned} \tag{6-4}$$

となる。したがって、 $K=1.5$ とすれば、定常偏差を 0 にすることができる。

6-3 過渡応答の改善

多くの場合、応答はできるだけ早く定常状態に落ち着くことが望ましい。前述の例では、 K を大きくすると、極が $-\infty$ に近づくので、応答を速くすることができる。しかし、応答を速くするためにゲインを大きくしようとする、定常偏差が 0 にできない。

そこで、PID 制御を行うことを考えよう。PID 制御 (*Proportional, Integral and Derivative control*) とは、偏差の比例、積分、微分信号をフィードバックに用いる制御である。すなわち、図 6-2 において、コントローラ $C(s)$ に、

$$\text{比例制御 } C(s) = K_P \tag{6-5}$$

$$\text{積分制御 } C(s) = \frac{K_I}{s} \tag{6-6}$$

$$\text{微分制御 } C(s) = K_D s \tag{6-7}$$

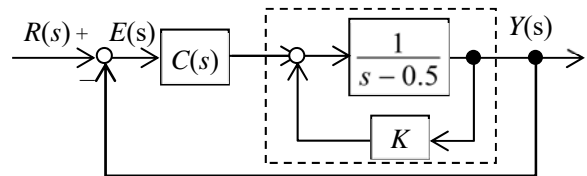


図 6-2 フィードバック制御系 2

もしくはこれらの組み合わせを用いる。

例 6-3: 一次系のフィードバック制御の PID 制御

図 6-2 のシステムで、 $K=1.5$ とする。目標入力 $R(s)$ が大きさ r のステップ関数であるとき、コントローラ $C(s)$ として、比例、積分、微分を用いると、それぞれ、定常偏差 $E(s)$ は、

$$\text{比例制御 } C(s) = K_P$$

$$\begin{aligned} \lim_{t \rightarrow \infty} e(t) &= \lim_{s \rightarrow 0} s \frac{1}{1 + K_p \frac{1}{s+1}} \frac{r}{s} \\ &= \lim_{s \rightarrow 0} \frac{s+1}{s+1+K_p} r \\ &= \frac{1}{1+K_p} r \end{aligned} \tag{6-8}$$

積分制御 $C(s) = K_I / s$

$$\begin{aligned} \lim_{t \rightarrow \infty} e(t) &= \lim_{s \rightarrow 0} s \frac{1}{1 + \frac{K_I}{s} \frac{1}{s+1}} \frac{r}{s} \\ &= \lim_{s \rightarrow 0} \frac{s(s+1)}{s^2 + s + K_I} r \\ &= 0 \end{aligned} \tag{6-9}$$

微分制御 $C(s) = K_D s$

$$\begin{aligned} \lim_{t \rightarrow \infty} e(t) &= \lim_{s \rightarrow 0} s \frac{1}{1 + K_D s \frac{1}{s+1}} \frac{r}{s} \\ &= \lim_{s \rightarrow 0} \frac{s+1}{(1 + K_D)s + 1} r \\ &= r \end{aligned} \tag{6-10}$$

となる．積分制御を行うことでゲインの値にかかわらず，定常偏差が 0 にできる．また，一般に，比例や微分要素を用いることで，速応性(目標値に速く到達する性質)が向上する．それぞれのゲインの調整法は，極配置や経験的な Ziegler-Nichols の方法などがある．

6-4 周波数応答とボード線図

システムに正弦波状入力を加わると，入力と同じ角周波数の正弦波状の出力が生じる．正弦波状入力に対するシステムの定常応答を周波数応答(*frequency response*)と言う．周波数応答は，入出力信号の振幅比(*gain*)と位相ずれ(*phase*)で表される．このゲインや位相は，入力の角周波数 ω の関数になる．これらは，伝達関数 $G(s)$ の s に $j\omega$ を代入した周波数伝達関数 $G(j\omega)$ を用いて，

$$\begin{aligned} \text{ゲイン} &: 20 \log_{10} |G(j\omega)| \text{ [dB]} \\ \text{位相} &: \angle G(j\omega) \text{ [}^\circ \text{]} \end{aligned}$$

で計算できる．横軸に角周波数 ω の対数を取り，縦軸にゲイン，位相をプロットしたグラフは，ボード線図(*Bode diagram*, *Bode plot*)と呼ばれ，周波数領域での解析に用いられる．

6-5 位相余裕，ゲイン余裕

図 6-3 下図のような閉ループ系(*closed loop system*)の安定性は，図 6-3 上図の開ループ系(*open loop system*)のボード線図から判断できることが知られている．なお，上図は，下図の一巡伝達関数とも呼ばれる．

ボード線図において，ゲインが 0[dB]になる周波数を，ゲイン交点角周波数と呼ぶ．この角周波数のときに，位相が -180° より何度上にあるかを位相余裕(*phase margin*)という(図 6-4)．一方，位相が -180° になる周波数を位相交点角周波数と呼ぶ．この角周波数のときに，ゲインが 0dB よりどれだけ下にあるかをゲイン余裕(*gain margin*)という．位相余裕，ゲイン余裕が正であることが，閉ループ系が安定であるための必要十分条件である．位相余裕，ゲイン余裕が大きいほど，モデル化誤差などがあってもシステムの安定性が保たれることになる．誤差や外乱があ

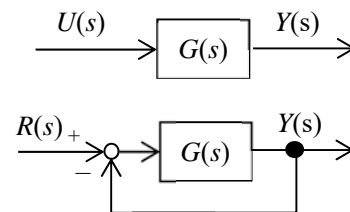


図 6-3 開ループ系と閉ループ系

っても安定性が保つ性質は、ロバスト性(*robustness*)と呼ばれ、制御系に要求される性能の一つである。ただし、余裕を持たせすぎるとは、速応性など、他の仕様を損なうことがある。

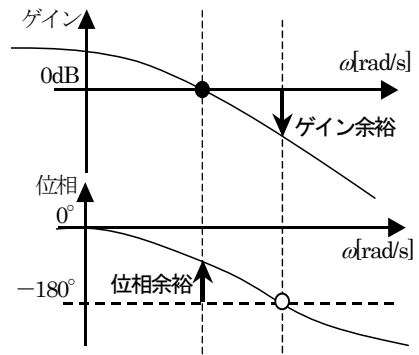


図 6-4 ゲイン余裕，位相余裕

演習6 古典制御（MATLAB/Simulink）

1. 例 6-1 において，入力 $r(t)$ を大きさ 1 のステップ関数として，ゲイン K の値によって，応答 $y(t)$ がどう変わるかを調べてみましょう。
また，偏差 $r(t)-y(t)$ のグラフも描いて，定常偏差を比べてみましょう。

2. 例 6-3 において，ゲイン K_P , K_I , K_D を変化させて，応答の違いを比較しましょう。

3. 図 6-2 のコントローラを

$$C(s) = K_P + K_I \frac{1}{s} + P_D s$$

としたとき，入力 $R(s)$ から $Y(s)$ までの伝達関数を計算しなさい。ただし， $K=1.5$ とします。

また， K_P , K_I , K_D を変化させて，定常偏差を 0 にして，できるだけ速く定常値に収束するように，ゲインを調整してみましょう。そのときのシステムの極も求めて，応答と極の関係を考察しましょう。

4. ボード線図を描いてみましょう。

$$\frac{1}{s}, s, \frac{1}{s+1}, (s+1), \frac{1}{s^2+s+1}, \frac{s+1}{s^2+s+1}, \frac{1}{s^3+s^2+s+2}$$

Mathematica のコマンド

ボード線図の書き方（control system toolbox の利用）：

コマンドラインで行います。伝達関数を作る関数を用います。

`g1 = tf([a1, a0], [b1, b0])` …… 伝達関数 $g1 = (a1s+a0)/(b1s+b0)$ を作る

`bodeplot(g1)` …… ボード線図を描く

`margin(g1)` …… ゲイン余裕，位相余裕，ゲイン角周波数，位相角周波数を求め，ボード線図にプロットする

`pole(g1)` …… 伝達関数 $g1$ の極を求める

m ファイルにまとめる：

コマンドラインで複数の命令を繰り返し行う場合は，m ファイルを作成すると便利です。

「ファイル」→「新規作成」→「M-ファイル」とすると，新規画面が出ます。

コマンドを書き込んで，「デバッグ」→「実行」で実行します。

結果は，コマンドラインに表示されます。

7. 可制御性・可観測性

システムが複雑になると、静的な出力フィードバック制御では安定化できない状態が現れることがある。そのような場合には別の方法を考えることになるだろう。しかし、そもそも、そのシステムは制御を行うことで安定化できる可能性があるのだろうか？そこで、現代制御理論では、対象とするシステムの性質として、全ての状態を制御することができるかどうか、状態を観測することができるかどうか、すなわち、可制御性・可観測性といった概念が登場する。

7-1 可制御性，可観測性

可制御性(*controllability*)は、全ての状態を制御できるかどうかを表す性質である。「任意の初期値 $\mathbf{x}(0)$ に対し、有限時間 T で、 $\mathbf{x}(T)=\mathbf{0}$ を満たす入力 $\mathbf{u}(t)$ が存在する」とき、システムは可制御であるという。一方、可観測性(*observability*)は、入力と出力から、全ての状態が特定できるかを表す性質である。「ある有限時間 T の間の入力 $\mathbf{u}(t)$ と出力 $\mathbf{y}(t)$ から、初期状態 $\mathbf{x}(0)$ が唯一決まる」とき、システムは可観測であるという。 $\mathbf{x}(0)$ が得られれば、 $\mathbf{x}(t)$ は、(5-8)式より得られる。

例：不可制御，不可観測なシステム

$$\begin{aligned}\dot{\mathbf{x}} &= \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix} \mathbf{x} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u \\ y &= (1 \quad -1) \mathbf{x}\end{aligned}\tag{7-1}$$

を考えよう。ただし、 $\mathbf{x} = (x_1, x_2)^T$ とする。それぞれの状態について分けてみると、

$$\begin{aligned}\dot{x}_1 &= x_1 + x_2 + u \\ \dot{x}_2 &= 2x_2\end{aligned}\tag{7-2}$$

となり、どのような入力 $u(t)$ を用いても、 x_2 を操作できないことがわかる。したがって不可制御なシステムである。

一方、出力 $y(t)$ を調べてみよう。ここでは、 $u(t)=0$ の場合を考える。

$$y = x_1 - x_2\tag{7-3}$$

出力 $y(t)$ からは、状態の差しかわからない。そこで、出力信号を微分してみよう。

$$\begin{aligned}\dot{y} &= \dot{x}_1 - \dot{x}_2 \\ &= (x_1 + x_2) - 2x_2 \\ &= x_1 - x_2\end{aligned}\tag{7-4}$$

微分した信号も、やはり、 $x_1 - x_2$ という状態の差しかわからない。もう一度微分をしても $x_1 - x_2$ になる。すなわち、このシステムは、入力(この例では 0)と出力 $y(t)$ から、内部状態の差はわかるが、 x_1, x_2 を特定することができない。したがって、不可観測である。

7-2 可制御性と可制御正準形

システムが可制御であるとは、任意の初期状態 $\mathbf{x}(0)$ をある有限時刻 t_1 で $\mathbf{x}(t_1)=\mathbf{0}$ に移す入力 $\mathbf{u}(t)$ 、 $0 \leq t \leq t_1$ が存在することである。システムの可制御性は状態方程式の係数行列 A と B で決まるので、

システムが可制御であることを， (A,B) は可制御なペアだと表現することもある。

システムの状態数が n である場合，可制御の必要十分条件はつぎのように知られている。

$$\text{システムが可制御} \Leftrightarrow \text{rank}(B, AB, A^2B, \dots, A^{n-1}B) = n \quad (7-5)$$

$$\Leftrightarrow \text{rank}(sI - A \mid B) = n, \text{ for } \forall s \in \mathbb{C} \quad (7-6)$$

である。ただし， $\forall s \in \mathbb{C}$ は，全ての複素数を表す。また，(7-5)式の右辺の行列を可制御性行列 (*controllability matrix*) という。可制御性行列は n 行の横長(もしくは正方)行列である。 n 行の横長行列のランクは n より大きくはなり得ないので，ランクが n であることをフルランク (*full rank*) ともいう。

可制御なシステムは，正準形と呼ばれる形に相似変換できる。一入力一出力系の場合の可制御正準形は，

$$\begin{aligned} \dot{\mathbf{x}} &= \begin{pmatrix} 0 & 1 & \cdots & \mathbf{0} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \\ -a_0 & -a_1 & \cdots & -a_{n-1} \end{pmatrix} \mathbf{x} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} u \\ y &= (b_0 \quad b_1 \quad \cdots \quad b_{n-1}) \mathbf{x} \end{aligned} \quad (7-7)$$

である。ただし， a_i, b_i は任意の定数である。なお，多入力多出力系の正準形についてはここでは省略するので，他の図書を参考にして欲しい。

可制御正準形(7-1)のシステムの特性方程式は，

$$s^n + a_{n-1}s^{n-1} + \cdots + a_1s + a_0 = 0 \quad (7-8)$$

である。また，伝達関数は，

$$y = \frac{b_{n-1}s^{n-1} + \cdots + b_1s + b_0}{s^n + a_{n-1}s^{n-1} + \cdots + a_1s + a_0} u \quad (7-9)$$

と計算できる。各自確認しておいて欲しい。

(7-7)が可制御であることを確認しておこう。可制御性行列(7-5)の各項を順次計算すると，

$$B = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{pmatrix}, AB = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ -a_{n-1} \end{pmatrix}, A^2B = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ -a_{n-1} \\ -a_{n-2} + a_{n-1}^2 \end{pmatrix} \cdots \quad (7-10)$$

となる。これをまとめた行列 $(B, AB, A^2B, \dots, A^{n-1}B)$ は行列式が 1 になる正則行列なので，

$$\begin{aligned} \text{rank}(B, AB, A^2B, \dots, A^{n-1}B) &= \begin{pmatrix} 0 & \cdots & 0 & 1 \\ \vdots & \ddots & \ddots & * \\ 0 & 1 & \ddots & \vdots \\ 1 & * & \cdots & * \end{pmatrix}, \quad * \text{は係数 } a_i \text{ の多項式} \\ &= n \end{aligned} \quad (7-11)$$

であり，確かに正準形(7-7)は可制御である。

7-3 可観測性と可観測正準形

システムが可観測であるとは、ある有限時刻 t_1 に対して、 $0 \leq t \leq t_1$ での出力 $y(t)$ と入力 $u(t)$ から、初期状態 $x(0)$ が唯一に決定できることである。システムの可観測性は状態方程式の意係数行列 C と A で決まるので、システムが可観測であることを (C, A) は可観測なペアだと表現することもある。可制御と同様に、可観測性の必要十分条件はつぎのようになることが知られている。

$$\text{システムが可観測} \Leftrightarrow \text{rank} \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{pmatrix} = n \quad (7-12)$$

$$\Leftrightarrow \text{rank} \begin{pmatrix} sI - A \\ C \end{pmatrix} = n, \text{ for } \forall s \in \mathbb{C} \quad (7-13)$$

である。(7-13)の右辺の行列を可観測性行列(*observability matrix*)という。可観測なシステムも正準形と呼ばれる形に相似変換できる。一入力一出力系の場合の可観測正準形は、

$$\begin{aligned} \dot{\mathbf{x}} &= \begin{pmatrix} 0 & \cdots & -a_0 \\ 1 & \ddots & \vdots & -a_1 \\ \vdots & \ddots & 0 & \vdots \\ \mathbf{0} & \cdots & 1 & -a_{n-1} \end{pmatrix} \mathbf{x} + \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix} u \\ y &= (\mathbf{0} \ \cdots \ 0 \ \ 1) \mathbf{x} \end{aligned} \quad (7-14)$$

である。可観測正準形(7-14)のシステムの特性方程式、および伝達関数は、可制御正準形(7-7)の特性方程式(7-8)、および伝達関数(7-9)と等しい。

7-4 可安定性、可検出性

可制御性、可観測性は、全ての状態を対象にしていた。しかし、もともと制御をしなくても安定な状態であるならば、可制御でなくても構わないだろう。そこで、可制御ではないが、状態フィードバックを用いればシステムを漸近安定にできるような性質を可安定(*stabilizability*)という。

例：可安定なシステム

次の2次系を考えてみよう。

$$\frac{d}{dt} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u \quad (7-15)$$

このシステムの可制御性行列は

$$(\mathbf{b}, A\mathbf{b}) = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \quad (7-16)$$

であるので可制御ではない。しかし、状態フィードバック入力

$$\begin{aligned} u &= -(k_1 \quad 0) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ &= -k_1 x_1 \end{aligned} \quad (7-17)$$

を用いる。ただし、 $k_1 > 0$ とする。このときフィードバック系は

$$\frac{d}{dt} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1-k_1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad (7-18)$$

となり、システムの極は -1 、 $1-k_1 < 0$ であり、漸近安定である。よって、このシステムは可安定である。

同様に、漸近安定な状態は何もしなくても 0 に収束するので観測できなくても大きな問題にはならない。漸近安定でない状態は観測できる性質を可検出(*detectability*)という。

システムが可安定、可検出であるかどうかは、それぞれ可制御の条件(7-6)、可観測の条件(7-13)のランク条件が、全ての複素数 s についてではなく、不安定な複素数、すなわち実部が非負の複素数について成り立つかどうかを調べればよい。

7-5 双対システム

可制御性行列(7-5)の転置行列を考えてみよう。 $(AB)^T = B^T A^T$ であることに注意すると、

$$\begin{bmatrix} B^T \\ B^T A^T \\ B^T (A^T)^2 \\ \vdots \\ B^T (A^T)^{n-1} \end{bmatrix} = [B \quad AB \quad A^2 B \quad \cdots \quad A^{n-1} B]^T \quad (7-19)$$

である。よって、 (A, B) が可制御であるとき、可制御性行列はフルランクである。したがって、転置した行列(7-19)もフルランクであり、 (B^T, A^T) は可観測なペアとなる。同様に、 (C, A) が可観測なペアであるとき、 (A^T, C^T) は可制御なペアとなる。このような関係を双対性(*duality*)という。

これを状態方程式で表現すると、システム、

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} \end{aligned} \quad (7-20)$$

に対して、

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}^T \mathbf{x} + \mathbf{C}^T \mathbf{u} \\ \mathbf{y} &= \mathbf{B}^T \mathbf{x} \end{aligned} \quad (7-21)$$

を双対システム(*dual system*)とよぶ。システム(7-20)が可制御可観測ならば、双対システム(7-21)も可制御可観測である。

演習7「可制御性・可観測性」（MATLAB/control system toolbox）

次の係数行列を考えましょう。

$$A = \begin{pmatrix} 0 & 1 & 3 & -1 \\ 0 & -1 & -2 & 2 \\ -1 & 0 & -2 & -1 \\ -1 & 0 & -3 & 0 \end{pmatrix}, \quad b_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad b_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$c_1 = (0 \quad 1 \quad 0 \quad 0)$$

$$c_2 = (1 \quad 0 \quad 0 \quad 0)$$

1. 可制御性を確認しなさい。 ① (A, b_1) ② (A, b_2)
2. 可観測性を確認しなさい。 ① (c_1, A) ② (c_2, A)
3. 伝達関数を求めなさい。 ① (c_1, A, b_1) ② (c_1, A, b_2) ③ (c_2, A, b_1) ④ (c_2, A, b_2)
4. 極を求めなさい。
 - i) $|sI - A| = 0$ を解いて求める。
 - ii) 3. で求めた伝達関数の分母=0 を解いて求める。

	MATLAB のコマンド
状態空間表現のシステムを定義	$H = \text{ss}(A, b, c, 0)$
可制御性行列	$Cc = \text{ctrb}(A, b)$, または, $Cc = [b, A*b, A*A*b \dots]$
可観測性行列	$Co = \text{obsv}(A, c)$, または, $Co = [c; c*A; c*A*A \dots]$
行列のランク	$\text{rank}(Cc)$
行列の固有値	$\text{eig}(A)$
伝達関数行列	$\text{tf}(H)$, または, $c * \text{inv}(s * \text{eye}(4) - A) * b$

8. レギュレータ

レギュレータ(調整器, *regulator*)とは, 外乱などによって平衡点からずれた状態をすみやかに平衡点に戻すためのフィードバック制御システム(*feedback control system*)である. ここでは線形の状態フィードバック制御のうち, 極配置法と最適レギュレータについて説明する.

8-1 極配置

状態数が n , 入力数が m の可制御なシステムを考えよう.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u \quad (8-1)$$

今, 全ての状態は直接測定可能であるとする. システムの状態 $\mathbf{x}(t)$ を入力 $u(t)$ の調整に用いる制御法は状態フィードバック (*state feedback*) と呼ばれる. 特に, 線形の状態フィードバックは,

$$\mathbf{u} = \mathbf{K}\mathbf{x} \quad (8-2)$$

と表される. ここで, \mathbf{K} は $m \times n$ 行列であり, フィードバックゲイン行列などと呼ばれる定数行列である.

さて, 状態フィードバックを用いた場合, 閉ループ系(*closed loop system*)の状態方程式は,

$$\dot{\mathbf{x}} = (\mathbf{A} + \mathbf{B}\mathbf{K})\mathbf{x} \quad (8-3)$$

となる. この閉ループ系の特性方程式は,

$$|s\mathbf{I} - (\mathbf{A} + \mathbf{B}\mathbf{K})| = 0 \quad (8-4)$$

である. したがって, フィードバックゲイン行列 \mathbf{K} の選び方によって, システムの極が変わることがわかる. ここで, 状態フィードバックの重要な性質として, 次のことが知られている.

「可制御なシステムの全ての極は, 状態フィードバックによって任意に設定できる」

可制御正準形(7-7)に対して, $\mathbf{K} = (k_0, k_1, \dots, k_{n-1})$ としたときの閉ループ系の特性方程式(8-4)を求めると,

$$s^n + (a_{n-1} - k_{n-1})s^{n-1} + \dots + (a_1 - k_1)s + (a_0 - k_0) = 0 \quad (8-5)$$

となり, フィードバックゲイン \mathbf{K} によって特性方程式の n 個の係数を自由に設定できる. すなわち, n 個の極を自由に設定することができる.

システムの極が指定された値 p_1, p_2, \dots, p_n になるようにフィードバックゲイン \mathbf{K} を決定することを極配置という.

例：極配置

つぎの状態方程式で表されるシステムを考えよう.

$$\dot{\mathbf{x}} = \begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix} \mathbf{x} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u \quad (8-6)$$

このシステムの極は $1 \pm \sqrt{2}$ なので, 不安定なシステムである. このシステムの極を -1 に配置

することを考えよう．状態フィードバック，

$$u = (k_1 \ k_2)x \quad (8-7)$$

を用いると，(8-6)式は，

$$\begin{aligned} \dot{x} &= \begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix}x + \begin{pmatrix} 0 \\ 1 \end{pmatrix}(k_1 \ k_2)x \\ &= \begin{pmatrix} 0 & 1 \\ 1+k_1 & 2+k_2 \end{pmatrix}x \end{aligned} \quad (8-8)$$

となる．このシステムの特性方程式は，

$$\begin{aligned} \left| sI - \begin{pmatrix} 0 & 1 \\ 1+k_1 & 2+k_2 \end{pmatrix} \right| &= 0 \\ s^2 - (2+k_2)s - (1+k_1) &= 0 \end{aligned} \quad (8-9)$$

である．極が -1 になるためには，特性多項式が

$$s^2 + 2s + 1 = 0 \quad (8-10)$$

となればよい．したがって，(8-9)(8-10)式を， s についての恒等式として解いて，ゲイン k_1, k_2 を求めれば，極配置のためのフィードバック則がつぎのように得られる．

$$u = (-2 \ -4)x \quad (8-11)$$

極配置のためのフィードバックゲインを決める方法には，システムの状態方程式を可制御正準形 (*controllability canonical form*) に変換して係数を比較する方法(上記の例)や，可制御正準形を用いない疋田，Ackermann らの方法がある．

8-2 最適レギュレータ

可制御なシステムは状態フィードバックで任意の極配置ができる．しかし，極の値はいくつにすればよいのだろうか？

例：入力と出力

つぎのシステムを考えよう．

$$\dot{x} = x + u \quad (8-12)$$

入力を，

$$u = -kx \quad (8-13)$$

とする．このとき，閉ループ系は，

$$\dot{x} = (1-k)x \quad (8-14)$$

となる．(8-14)式を解くと，状態 x の時間変化は，

$$x(t) = e^{(1-k)t} x(0) \quad (8-15)$$

である。これより、 k が大きければ大きいほど、状態は速く 0 に収束することがわかる。一方、 k が大きいということは、(8-13)式から、入力の絶対値が大きくなることがわかる。

この例に示すように、一般に状態の収束を速くするためには、大きな入力を必要とする。実システムでは、制御入力の大きさに制約がある場合が多いので、あまり大きな制御入力は望ましくない。そこで、入力と状態の両方をできるだけ小さくするために、評価関数を導入して、評価関数を最小にする制御を考える。

まとめ：最適レギュレータ

可制御なシステムに対して、評価関数

$$J = \int_0^{\infty} (\mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u}) dt, \quad \mathbf{Q} \geq 0, \mathbf{R} > 0 \quad (8-16)$$

を最小にする入力は、つぎの状態フィードバックで与えられる。

$$\mathbf{u} = -\mathbf{R}^{-1} \mathbf{B}^T \mathbf{P} \mathbf{x} \quad (8-17)$$

ただし、 \mathbf{P} は、Riccati 方程式、

$$\mathbf{P} \mathbf{A} + \mathbf{A}^T \mathbf{P} - \mathbf{P} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^T \mathbf{P} + \mathbf{Q} = 0 \quad (8-18)$$

の唯一の正定対称解である。この制御則を最適レギュレータ (*Linear Quadratic Regulator, LQR*) という。

最適レギュレータ(8-17)を用いたとき、評価関数(8-16)は、

$$J = \mathbf{x}(0)^T \mathbf{P} \mathbf{x}(0) \quad (8-19)$$

となる。また、評価関数の中の \mathbf{Q} , \mathbf{R} は重み関数と呼ばれ、入力と出力のどちらを重視するかを決める値である。例えば、 \mathbf{R} を大きくすると(8-16)式の2項目において $\mathbf{u}(t)$ を小さくしないと評価関数の値が大きくなってしまふ。よって、 \mathbf{R} を大きくすると入力を小さくすることに重きをおき、より \mathbf{u} が小さくする最適レギュレータが得られる。

演習8「極と応答／レギュレータ」（MATLAB/simulink/control system toolbox）

1. Simulink を用いて，応答を見てみましょう．

$$\dot{x} = Ax + bu$$

$$y = cx$$

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -9 & -9 & -1 \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$c = (9 \ 0 \ 1)$$

- i) 入力を 0 として，初期値を適当に変えて，状態と出力を見てみましょう。
(システムの極を求めてみましょう．固有の振動モードを持っていることがわかります)
- ii) 入力をステップ関数 $u(t)=1$ ，正弦波関数 $u(t)=5\sin t$ にした場合はどうなるでしょうか．
(システムの固有のモードに加えて，入力モードが加わっていることがわかつています)

2. 極配置を行う状態フィードバック則を求めなさい．

$$A = \begin{pmatrix} 0 & 1 & 3 & -1 \\ 0 & -1 & -2 & 2 \\ -1 & 0 & -2 & -1 \\ -1 & 0 & -3 & 0 \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$c = (1 \ 0 \ 0 \ 0)$$

- ① 全ての極を -1 にする
- ② 全ての極を -3 にする

3. 2で求めたフィードバック則を用いた場合，それぞれ，出力，状態，制御入力の時間変化を見てみましょう．

	MATLAB のコマンド
極配置(Ackermann の方法)	<code>acker(A, b, p)</code> p は，配置したい極を並べたベクトル $p=[p_1, p_2, \dots, p_n]$ 答えは， $u = -kx$ の k . 符号に注意.
最適レギュレータ	<code>lqr(A, B, Q, R, N)</code> $\dot{x} = Ax + Bu$ 評価関数は， $J = \int_0^{\infty} (x^T Qx + u^T Ru + 2x^T Nu) dt$ N は <i>cross term</i> という．通常は 0 でよい. 答えは， $u = -kx$ の k . 符号に注意.

※ シミュレーションを行う場合，「コンフィギュレーションパラメータ」の設定で，「相対許容誤差」を小さくすると，計算精度が上がる．

4. 2. のシステムに対して，最適レギュレータを求めましょう．ただし，評価関数の重み行列を
つぎのように変えて求めなさい．
- ① $Q=I_4, r=1$ (I_4 は， 4×4 の単位行列)
 - ② $Q=I_4, r=1000$
 - ③ $Q=1000 \cdot I_4, r=1$
5. 4. で設計した最適レギュレータを用いた場合の閉ループ系の極を求めなさい．
6. 4. で設計した最適レギュレータを用いた場合の初期値応答を比べてみましょう．

9. オブザーバ

9-1 同次元オブザーバ

可制御なシステムは、状態フィードバックで安定化できる。しかし、通常は、内部状態を直接計測することはできない。そこで、入力と出力から内部状態を推定するオブザーバ(観測器, *observer*)を用いる。

制御対象のモデルは次式でわかっているとす。

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned} \quad (9-1)$$

このとき、最も直感的な方法としては、実システムへの入力をモデルにいれて、内部状態を推定する方法である。すなわち、 x の推定値 \hat{x} を、

$$\dot{\hat{x}} = A\hat{x} + Bu \quad (9-2)$$

という式を解いて得ることとする。この解は、

$$\hat{x}(t) = e^{At} \hat{x}(0) + \int_0^t e^{A(t-\tau)} Bu(\tau) d\tau \quad (9-3)$$

である。入力 $u(t)$ は制御対象への入力と同じ値を用いばよい。しかし、オブザーバの状態はシステムの状態によって修正されないので、システムに外乱が加わって $\hat{x}(0) \neq x(0)$ となった場合、その差が残ってしまう。そこで、図 9-1 のように、制御対象とオブザーバの出力の差を用いて推定値を修正することを考える。

$$\dot{\hat{x}} = A\hat{x} + Bu - L(y - C\hat{x}) \quad (9-4)$$

このオブザーバは、同次元オブザーバ(*full order observer*)と呼ばれる。このオブザーバに関して、次のことが知られている。

「可観測なシステムの状態は、オブザーバ(9-4)によって推定できる」

これは、簡単に証明することができる。まず、状態の推定誤差を $e = x(t) - \hat{x}(t)$ とする。これを微分すると、

$$\begin{aligned} \dot{e} &= \dot{x}(t) - \dot{\hat{x}}(t) \\ &= (Ax + Bu) - (A\hat{x} + Bu - L(y - \hat{y})) \\ &= Ax - (A\hat{x} - L(Cx - C\hat{x})) \\ &= (A + LC)e \end{aligned} \quad (9-5)$$

となるので、 $(A + LC)$ が安定行列ならば、 $t \rightarrow \infty$ で $e \rightarrow 0$ に収束する。すなわち、 $\hat{x} \rightarrow x$ である。

ところで、 $(A + LC)$ の固有値は、 $(A + LC)^T = A^T + C^T L^T$ の固有値と同じだから、 (A^T, C^T) が可制御ならば、任意に配置できる。可制御性、可観測性の判定条件の双対性から、これは (C, A) が可観測であることと等価である。

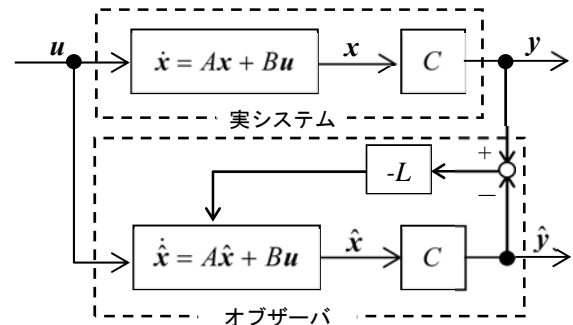


図 9-1 オブザーバ

9-2 最小次元オブザーバ

同一次元オブザーバでは n 個の状態全てを推定していた。しかし、直接観測できる出力が p 個ある場合、状態のうち p 個分の情報は座標変換などで得られる可能性がある。オブザーバの次数が下がれば、その分、動的な計算が不要になり、メモリなどの節約になる。

例えば、出力方程式が

$$\begin{aligned} \mathbf{y} &= \begin{pmatrix} I & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} \\ &= \mathbf{x}_1 \end{aligned}$$

だった場合、状態ベクトル \mathbf{x} のうち、 p 個の要素 \mathbf{x}_1 は推定しなくても直接測定できる。このような場合、残りの $n-p$ 個だけを推定する $n-p$ 次元の低次元オブザーバ(*minimal order observer*)を設計すればよい。これを一般的に考えてみよう。

まず、出力方程式の C 行列に対して、

$$\text{rank} \begin{pmatrix} C \\ V \end{pmatrix} = n \quad (9-6)$$

を満たす行列 V を定義し、ベクトル $V\mathbf{x}$ を \mathbf{z} とおく。

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{z} \end{pmatrix} = \begin{pmatrix} C \\ V \end{pmatrix} \mathbf{x} \quad (9-7)$$

(C, A) が可観測な場合、任意に与えられた $(n-p) \times (n-p)$ の安定行列 T に対して、

$$V(A-LC) = TV \quad (9-8)$$

を満たす行列 L が存在する。これらを用いて、状態の推定値 $\hat{\mathbf{x}}$ を

$$\hat{\mathbf{x}} = \begin{pmatrix} C \\ V \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{y} \\ \hat{\mathbf{z}} \end{pmatrix} \quad (9-9)$$

$$\dot{\hat{\mathbf{z}}} = T\hat{\mathbf{z}} + VL\mathbf{y} + VB\mathbf{u} \quad (9-10)$$

で求める。これを最小次元オブザーバという。

最小次元オブザーバ(9-9)において、 \mathbf{y} は直接測定できているので、残りの \mathbf{z} が推定できることを確かめてみよう。状態 \mathbf{z} の偏差を計算すると、(9-7)(9-10)を用いて、

$$\begin{aligned} \dot{\mathbf{e}} &= \dot{\mathbf{z}}(t) - \dot{\hat{\mathbf{z}}}(t) \\ &= V(\mathbf{Ax} + \mathbf{Bu}) - (T\hat{\mathbf{z}} + VL\mathbf{y} + VB\mathbf{u}) \end{aligned} \quad (9-11)$$

となる。ここに、(9-9)を代入して変形すると、

$$\begin{aligned} \dot{\mathbf{e}} &= V(\mathbf{Ax} + \mathbf{Bu}) - (TV\hat{\mathbf{x}} + VLC\mathbf{x} + VB\mathbf{u}) \\ &= V(\mathbf{Ax} + \mathbf{Bu}) - ((TV + VLC)\mathbf{x} + VB\mathbf{u} - TV(\mathbf{x} - \hat{\mathbf{x}})) \end{aligned} \quad (9-12)$$

であり、(9-8)を代入すると、

$$\begin{aligned} \dot{e} &= V(Ax + Bu) - V(Ax + Bu) + TV(x - \hat{x}) \\ &= TV(x - \hat{x}) \\ &= T(z - \hat{z}) \end{aligned} \tag{9-13}$$

となる。\$T\$ が安定行列であることから、\$z\$ の偏差 \$e\$ は漸的に \$0\$ に収束する。したがって、最小次元オブザーバの推定値(9-9)と状態 \$x\$ の偏差も \$0\$ に収束する。

9-3 オブザーバを用いたレギュレータ

制御対象(9-1)に対して、オブザーバ(9-4)で推定した状態 \$\hat{x}\$ を用いて、フィードバック制御を行う。

$$u = K\hat{x} \tag{9-14}$$

このときのシステム全体の安定性を考えてみよう。(9-1)(9-4)および(9-6)をまとめると、\$x, \hat{x}\$ を状態とした拡大系が得られる。

$$\begin{pmatrix} \dot{x} \\ \dot{\hat{x}} \end{pmatrix} = \begin{pmatrix} A & BK \\ -LC & A+BK+LC \end{pmatrix} \begin{pmatrix} x \\ \hat{x} \end{pmatrix} \tag{9-15}$$

このシステムの特性多項式は、つぎのように計算できる。

$$\begin{aligned} \left| sI - \begin{pmatrix} A & BK \\ -LC & A+BK+LC \end{pmatrix} \right| &= \begin{vmatrix} sI - A & -BK \\ LC & sI - (A+BK+LC) \end{vmatrix} \\ &= \begin{vmatrix} sI - A & I \\ LC & I \end{vmatrix} \begin{vmatrix} I & -I \\ 0 & sI - (A+BK) \end{vmatrix} \\ &= |sI - (A+LC)| |sI - (A+BK)| \end{aligned} \tag{9-16}$$

したがって、拡大系(9-15)の極は、状態フィードバックだけを用いた場合の極と、オブザーバの極である。すなわち、状態フィードバックゲイン \$K\$ とオブザーバゲイン \$L\$ は、それぞれ独立に設計することができる。

一般に、レギュレータの極 (\$A+BK\$ の固有値)よりもオブザーバの極(\$A+LC\$ の固有値)をより \$-\infty\$ 側に配置することで、内部状態の推定を速く行う。

9-4 カルマンフィルタ

白色雑音(white noise)が加わっているときに、推定誤差を最小にするようにオブザーバゲイン \$L\$ を決定することを考えよう。システムに、システム雑音 \$v(t)\$ と測定雑音 \$w(t)\$ が加わっているとする。\$v(t)\$ と \$w(t)\$ は互いに無相関な白色雑音であり、それぞれの共分散行列は \$Q, R\$ であるとする。

$$\begin{aligned} \dot{x} &= Ax + Bu + v \\ y &= Cx + w \end{aligned} \tag{9-17}$$

このとき、評価関数をつぎのようにおく。

$$J = E[e(t)^T e(t)] \tag{9-18}$$

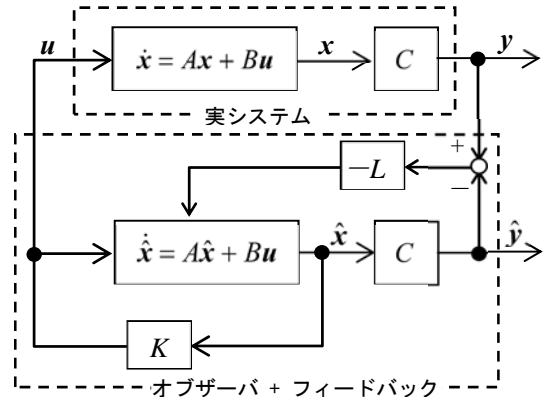


図 9-2 オブザーバを用いたレギュレータ

ここで, $E[x]$ は, x の期待値をあらわす. この評価関数を最小にするオブザーバは, 定常カルマンフィルタ(Kalman filter)と呼ばれる. このオブザーバゲインは,

$$L = PC^T R^{-1} \quad (9-19)$$

である. ただし, P は, Riccati 方程式,

$$PA^T + AP - PC^T R^{-1} C P + Q = 0 \quad (9-20)$$

の正定対称の解である.

演習9「オブザーバ」（MATLAB/simulink/control system toolbox）

1. 演習8の2のシステムに対して、オブザーバを構成しなさい。オブザーバの極を

- ① 全ての極を -1 にする
- ② 全ての極を -3 にする

とします。このときの推定誤差の収束の様子をグラフで確認しましょう。

2. 上記で設計したオブザーバを用い、演習8の2で求めたフィードバックゲインを組み合わせ、フィードバック制御を行ったときの応答をみてみましょう。

10. サーボ系

通常、制御系設計では、まずシステムの安定化を考える。そのうえで各制御系に対する要求に応じて目的を達成させるコントローラを設計する。この制御目的の一つに目標値追従がある。目標値追従を目的としたシステムをサーボ系(*servo system*)と呼ぶ。目標値追従が可能かどうかは、制御対象の特性と目標値の種類に依存する。たとえば、自動車は緩やかなS字状の目標コースには追従できるが、直角に曲がる目標値には追従させられない。

制御対象 $P(s)$ の出力 $Y(s)$ を目標値 $R(s)$ に追従させるための制御系構成は、大きく分けて図 10-1 のようなものがある。図 10-1 は上から、フィードフォワード制御系、フィードバック制御系、2 自由度制御系と呼ばれる。

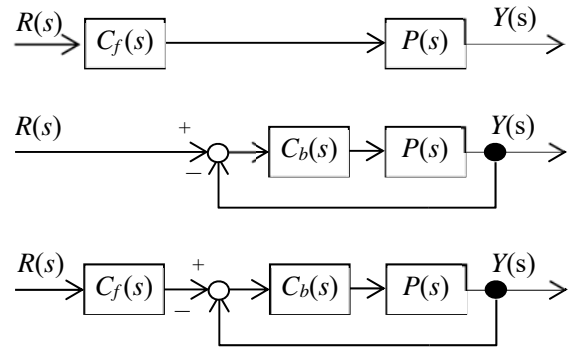


図 10-1 体系的な制御系の構成

フィードフォワード制御によって、目標値追従を達成するための直感的な方法は、逆システム(*inverse system*)を構成することである。すなわち、 $C_f(s) = 1/P(s)$ となるコントローラ $C_f(s)$ を用いれば、 $Y(s) = R(s)$ となり、出力は任意の入力に追従する。しかし、一般的には厳密な逆システムを構成することは困難である。通常、制御対象はプロパー(分母の次数 \geq 分子の次数)である。したがって、要求されるコントローラ $1/P(s)$ はプロパーではなく、微分器が必要となる。加えて、 $P(s)$ が不安定零点を持つ場合には、 $1/P(s)$ は不安定となるため、システム内部で信号が発散する。さらに、フィードフォワード制御系は一般に外乱に弱いなどの問題がある。

一方、フィードバック制御系はシステムに加わった外乱に応じて制御入力に補正をかけるので、ある程度のロバスト性が期待できる。そのため、安定化だけでなく、サーボ系の構成でもよく用いられる。本章では、フィードバック制御系について説明する。

しかし、フィードバック制御は、出力を用いて制御入力を変更するため、どうしてもシステムの動特性に応じた遅れが生じてしまう。これを改善する目的で、フィードフォワード制御を組み合わせた 2 自由度制御系が用いられることがある。

10-1 システムの形と定常偏差

ロボットアームを決められた位置に停止させる問題などを考えてみよう。このとき、目標位置は制御の開始から終了まで一定値である。

そこで、一定の目標値 r_d に対して、図 10-1 のフィードバック制御系の定常偏差、

$$\lim_{t \rightarrow \infty} e_{ss}(t) = \lim_{t \rightarrow \infty} (r_d - y(t)) \tag{10-1}$$

を調べてみよう。目標値をラプラス変換した信号 $R(s)$ は、

$$R(s) = r_d \frac{1}{s} \tag{10-2}$$

であるから、最終値の定理を用いれば、定常偏差 e_{ss} は、

$$\begin{aligned} \lim_{t \rightarrow \infty} e_{ss}(t) &= \lim_{s \rightarrow 0} s \frac{1}{1 + P(s)C_b(s)} \frac{r_d}{s} \\ &= \lim_{s \rightarrow 0} \frac{1}{1 + P(s)C_b(s)} r_d \end{aligned} \tag{10-3}$$

となる。これが 0 になるためには、制御対象 $P(s)$ かコントローラ $C_b(s)$ に積分器を一つ含めばよい。例えば、コントローラが $C_b(s)=C_b'(s)/s$ であれば、

$$\begin{aligned} \lim_{t \rightarrow \infty} e_{ss}(t) &= \lim_{s \rightarrow 0} \frac{1}{1 + P(s) \frac{C_b'(s)}{s}} r_d \\ &= \lim_{s \rightarrow 0} \frac{s}{s + P(s)C_b'(s)} r_d \\ &= 0 \end{aligned} \tag{10-4}$$

となり、定常偏差は 0 になることがわかる。

このことから、一定の位置に追従させる場合には積分器を用いることが多い。積分器によって定常偏差は 0 にできるが、過渡状態では偏差が残る。この過渡状態での収束を速くするためには比例要素、微分要素を併用すればよい(PID 制御)。

また、積分器を用いて定常偏差をなくす考え方は、「システムの形」として古典制御のテキストなどで述べられている(表 10-1)。本節での定式化においては、システムの形とは、 $P(s)C_b(s)$ に含まれる積分器の数である。

表 10-1 システムの形

システムの形	定常位置偏差 (目標値がステップ)	定常速度偏差 (目標値がランプ)	定常加速度偏差 (目標値が二次関数)
0	定数 (0 以外)	∞	∞
1	0	定数 (0 以外)	∞
2	0	0	定数 (0 以外)
3	0	0	0

10-2 内部モデル原理 (internal model principle)

つぎに、さらに一般的な目標値に追従させる方法について述べる。目標値をラプラス変換した関数が一般的に、

$$R(s) = \frac{p(s)}{r(s)} \tag{10-5}$$

であるとする。図 10-1 のフィードバック制御系の制御対象 $P(s)$ とコントローラ $C_b(s)$ をそれぞれ分母と分子に分けて、

$$P(s) = \frac{b(s)}{a(s)}, \quad C_b(s) = \frac{d(s)}{c(s)} \tag{10-6}$$

と表わす。ここで、多項式 $r(s)$ と $p(s)$, $a(s)$ と $b(s)$, $c(s)$ と $d(s)$ は、それぞれ互いに素である(*coprime*)とする。互いに素(=既約)とはそれ以上約分できないことを意味する。

このシステムの偏差は、

$$\begin{aligned}
 e(s) &= \frac{1}{1 + \frac{b(s)d(s)}{a(s)c(s)}} \frac{p(s)}{r(s)} \\
 &= \frac{a(s)c(s)}{a(s)c(s) + b(s)d(s)} \frac{p(s)}{r(s)}
 \end{aligned}
 \tag{10-7}$$

と計算できる．この $e(s)$ の分母が安定多項式(多項式を 0 にする s の実部がすべて負)ならば，偏差 $e(t)$ は 0 に収束する．

そのためには，まず閉ループ系の特性多項式 $a(s)c(s) + b(s)d(s)$ を安定にする必要がある．つぎに， $r(s)$ に関しては，一般にサーボ系で対象にする目標値は 0 に収束するような関数ではない．例えば，前節で扱った一定目標値は $r(\infty) = r_d$ である．したがって，偏差が 0 に収束するためには，分子の $a(s)c(s)$ が $r(s)$ の不安定モード(0 に収束しないモード)を含むことが必要となる．制御対象の分母が $r(s)$ を含まない場合にはコントローラの方母 $c(s)$ に $r(s)$ を入れる．しかし，このとき制御対象の分子 $b(s)$ に $r(s)$ を持っているとき， $a(s)c(s) + b(s)d(s)$ が $r(s)$ を共通項として持つことになり，定常偏差は 0 にできないことに注意する．

以上をまとめると，つぎのように言える．

制御対象の分子と目標値関数の分母は互いに素であるとする．このとき，サーボ系を構成することができ，その条件は，

- ① 閉ループ系を安定にする
→ $a(s)c(s) + b(s)d(s)$ を安定多項式にする
- ② 制御対象かコントローラに目標値の不安定モードを持つ
→ $a(s)c(s)$ が $r(s)$ を含む (内部モデル原理)

である．

10-3 その他

ロボットアームなど，アクチュエータの数が多いシステムにおいては，内部モデル原理を考えることなく，もっと直接的な制御を行うことができる．

一般に，ロボットアームの運動方程式は，一般化座標 q として各関節の角度を用いて，

$$M(q)\ddot{q} + h(q, \dot{q}) + V\dot{q} + g(q) = \tau \tag{10-8}$$

という形で表わされる．ここで， M は慣性行列， h は回転による遠心力やコリオリ力を表わす項， g は重力項， V は粘性抵抗， τ は関節モータへのトルク入力である．ここで，注意して欲しいのは，右辺の制御入力 τ の係数がない(=単位行列)である点である．このことから，モデルがわかっている場合，目標値 q_d に追従させるためには，

$$\tau = M(q, \dot{q})\ddot{q} + h(q, \dot{q}) + V\dot{q} + g(q) + (q - q_d) \tag{10-9}$$

という入力を用いれば，(10-8)に代入することで，

$$q = q_d \tag{10-10}$$

が理想的には達成できる．しかし， \ddot{q} が測定できない場合には測定値を微分することになり，制御に用いることは一般には好ましくない．また，モデル化誤差や外乱も避けられない．そこで，例えば，

$$\begin{aligned}\tau &= M(q, \dot{q})u + h(q, \dot{q}) + V\dot{q} + g(q) \\ u &= \ddot{q}_d + K_v(\dot{q} - \dot{q}_d) + K_p(q - q_d)\end{aligned}\tag{10-11}$$

として,

$$(\ddot{q}_d - \ddot{q}) + K_v(\dot{q}_d - \dot{q}) + K_p(q_d - q) = 0\tag{10-12}$$

を達成させる方法がある. 適当な K_v , K_p として正定な行列を選ぶことで, 漸近的に $q \rightarrow q_d$ が達成できる.

演習10「サーボ系」（MATLAB/Simulink, control system toolbox）

1. 次の伝達関数で表わされる制御対象に対して，サーボ系を構成しましょう．

$$y = \frac{2}{s^2 + s + 1} u$$

目標値

- ① 一定位置 $y_d = 3$
- ② 一定速度 $y_d = 2t$
- ③ 周期運動 $y_d = \sin t$

2. 次の伝達関数で表わされる不安定な制御対象に対して，サーボ系を構成しましょう．

$$y = \frac{2}{s^2 - s + 1} u$$

目標値

- ① 一定位置 $y_d = 3$
- ② 一定速度 $y_d = 2t$
- ③ 周期運動 $y_d = \sin t$

11. 簡単な非線形制御

11-1 システムのモデル

良い制御を行う場合には、制御対象の性質を理解し、制御対象に適した制御法を用いる必要がある。実在システムの特徴を、工学的に利用できる客観的な表現でシステムを表したものをモデル(model)という。

実在システムは、非線形性を含むものがほとんどである。たとえば、コイルばねの発生力は、

$$F = kx \quad (11-1)$$

のように、ばねの変位に比例して表現される。しかし、実際には、ばねを強い力で伸ばしていくと、ばねは塑性変形し、もとの形に戻らなくなる。つまり、(11-1)式は、ばねの伸びがある範囲内(弾性領域)にある場合だけ成り立つ関係式である。

このように、非線形な性質を持つシステムでも、対象とする稼動範囲を限定することで、線形モデルで表すことができる。

11-2 非線形システムの線形化

システムの状態が非線形関数に従って変化するシステムを非線形システムという。非線形システムに対しては、線形制御理論のような一般的なコントローラ設計法はない。そのため、システムを線形モデルで表して、コントローラ設計を行うことが多い。

非線形システムのモデルに対して、変数変換や座標変換を用いることで、見かけ上、線形なモデルとして表すことができる。これを厳密な線形化という。

例：厳密な線形化

次のシステムは、非線形なシステムである。

$$\dot{x} \cos x = \sin x + u \quad (11-2)$$

このシステムにおいて、

$$z = \sin x \quad (11-3)$$

とおくと、(11-2)式は、

$$\dot{z} = z + u \quad (11-4)$$

と書き直せる。このシステムは、 $u = -kz$ 、 $k > 1$ で安定化できることが容易にわかる。よって、

$$u = -k \sin x \quad (11-5)$$

とすればシステムは安定になる。ただし、 $z \rightarrow 0$ のとき、 $x \rightarrow n\pi$ であり、 x は複数ある平衡点のうちの一つに収束することに注意する。

しかし、3次以上のシステムに対しては、厳密な線形化ができるとは限らない。よって、多くの場合、前節で述べたように、システムの入出力、状態の範囲を限り、線形システムに近似して制御系設計を行うことが多い。

例：線形近似

システム(11-2)の状態 x が 0 に近い範囲で動いている場合を考えよう．まず，システム(11-2)を $x=0$ まわりでテイラー展開する．

$$\begin{aligned} \cos x &= \cos(0) + \frac{1}{1!}(-\sin(0))x + \frac{1}{2!}(-\cos(0))x^2 + \frac{1}{3!}(\sin(0))x^3 + \frac{1}{4!}(\cos(0))x^4 + \dots \\ &= 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 + \dots \\ \sin x &= x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 + \dots \end{aligned}$$

ここで， $x \ll 1$ として， x の 2 次以上の項を無視すると，(11-2)は，

$$\dot{x} = x + u \tag{11-6}$$

と線形近似できる．このシステムは，

$$u = -kx \tag{11-7}$$

によって安定化できる．十分時間が経ったとき， $x \rightarrow 0$ に収束する．これは， x が大きな値になると成立しない．

11-3 安定性

非線形なシステムの安定性を論ずる理論として，リアプノフの安定性(Lyapunov stability)がある．

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \tag{11-8}$$

に対して，ある正定関数 $V(\mathbf{x}) > 0$ (任意の $\mathbf{x} \neq 0$ で $V(\mathbf{x})$ は正， $\mathbf{x} = 0$ で $V(0) = 0$) の時間微分が，

$$\dot{V}(\mathbf{x}) \leq 0 \tag{11-9}$$

を満たすとき，この $V(\mathbf{x})$ をリアプノフ関数と呼ぶ．リアプノフ関数が存在するならば，システムはリアプノフの意味で安定であるという．

例：リアプノフの安定論 1

簡単な線形システム，

$$\dot{x} = -x \tag{11-10}$$

を考えよう．このとき，

$$V(x) = \frac{1}{2}x^2 \tag{11-11}$$

を考える．明らかに $V(x) > 0$ である．この時間微分は，

$$\begin{aligned} \dot{V}(x) &= x\dot{x} \\ &= -x^2 < 0 \quad (x \neq 0 \text{ で負}) \end{aligned} \tag{11-12}$$

であるので安定である（この場合は，微分が負定なので，漸近安定である）．

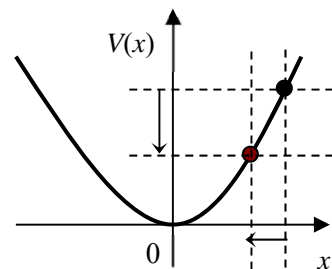


図 11-1 リアプノフ関数と状態の時間変化

例：リアプノフの安定論 2

非線形システム，

$$\dot{x} \cos x = -\sin x \quad (11-13)$$

を考えよう．このとき，

$$V(x) = \frac{1}{2} \sin^2 x \quad (11-14)$$

とすると，これは， $x \neq n\pi$ で正である．この微分は，

$$\begin{aligned} \dot{V}(x) &= \dot{x} \cos x \sin x \\ &= -\sin^2 x \leq 0 \quad (\sin x = 0 \text{で等号成立}) \end{aligned} \quad (11-15)$$

であるので，安定である．このとき，ただし， $V(x)$ が任意の x で正になるわけではないので，大域的に安定であることは言えない．実際に，リアプノフ関数を描いて考えてみると，図 11-2 のように周期関数になっている．このため，たとえば，状態の初期値が $\pi/2 < x(0) < 3\pi/2$ の場合， x は 0 ではなく， π に収束する．したがって， $x=0$ は局所的に漸近安定という．

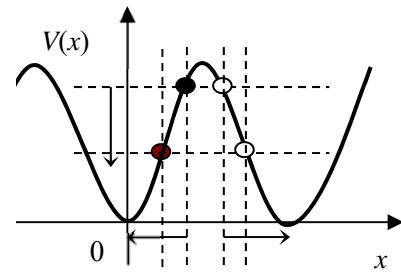


図 11-2 局所安定

12. ロボットの軌道制御の例

ここでは、ロボットを与えられた目標の軌道に沿って走行させることを例に、極と応答の関係や非線形性について考えていこう。

12-1 ロボットの運動モデル

左右のタイヤが独立に制御できるような移動ロボットを対象とする。通常、バランスを取るために補助輪（キcaster）をつける。補助輪がない場合は倒立二輪ロボットのように姿勢安定化制御を同時に行う必要があるため制御は複雑になるが、軌道を制御することはできる³⁾。ここでは補助輪がついているものとして、姿勢制御は考えない。

まず、ロボットのタイヤは地面に対して滑らないと仮定する。この仮定は自動車が凍った路面でスリップする場合や、急発進や急ブレーキをかけたときにタイヤが空転している場合などでは満たされないが、建物の廊下などを比較的ゆっくり走るロボットでは満たされると考えることができる。つまり、ゆっくり走行するロボットの制御を考える問題においては、この仮定は「妥当」だと言えます。

この仮定の下、ロボットの速度と位置の関係を考えてみよう。ロボットの位置を表すために、地上に固定した座標系を定義する。この座標系におけるロボットの中心位置（＝左右の車輪の中心位置）を (x, y) とする。また、この座標系の x 軸に対するロボットの向き（姿勢角）を θ とおく。中心位置 (x, y) が変化する速度を並進速度と呼び、これを v_1 とし、ロボットの向きが中心位置に対して回転する速度を旋回角速度と呼び、これを v_2 とおく（図 12-1）。

ロボットの並進速度 v_1 を x 軸、 y 軸方向に分解した成分 v_x, v_y は、ロボットの向き θ を用いて、それぞれ、

$$v_x = v_1 \cos \theta \quad (12-1)$$

$$v_y = v_1 \sin \theta \quad (12-2)$$

である。位置を時間で微分したものが速度なので、 x 軸方向の成分についていえば、 $dx/dt = v_x$ が成り立つ。したがって、ロボットの位置 (x, y) の時間変化は、

$$\frac{dx}{dt} = v_1 \cos \theta \quad (12-3)$$

$$\frac{dy}{dt} = v_1 \sin \theta \quad (12-4)$$

という式で表わすことができる。一方、姿勢角は、

$$\frac{d\theta}{dt} = v_2 \quad (12-5)$$

という式にしたがって変化する。(12-3)～(12-5)式をあわせたものが二輪車両の運動モデルとしてよく用いられる式である。

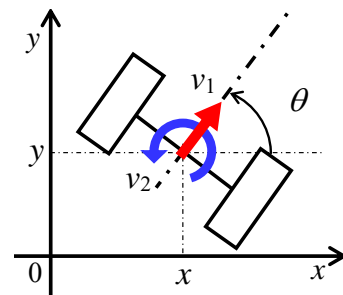


図 12-1 ロボットの位置と姿勢

12-2 目標軌道への追従制御

水平面内の関数として与えられた目標軌道 $y_r(x)$ に追従させる。 $y_r(x)$ を定数の場合、図 12-2 のよ

うに自動車のレーンチェンジのイメージである。ロボットの状態（位置と速度）は計測できるものとし、このロボットの状態をフィードバック制御を行っていかう。なお、位置と速度を計測する方法の例として、エンコーダを用いる方法を付録 D に示す。

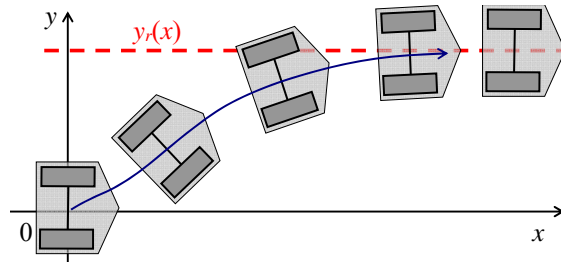


図 12-2 目標値への追従制御

12-2-1 並進速度の PI 制御

まず、並進速度の制御を考える。ここでは一定の目標速度での走行を目指すこととし、この目標速度を v_d とする。今の並進速度が目標の v_d よりも遅ければ加速させ、速ければ減速させることで速度を目標に近づけるように比例制御を用いてみよう。つまり、ロボットの今の並進速度 $v_1(t)$ を用いて並進方向の入力 $u_1(t)$ を

$$u_1(t) = K_P (v_d - v_1(t)) \quad (12-6)$$

とする。 K_P は正の定数である。

比例制御(12-6)を用いると、目標から大きく離れているときは大きな入力を加え、近づいたら入力が小さくなる。これによって目標軌道に収束することが期待できる。しかし、目標に近づいて入力を小さくしすぎたせいで定常偏差が残ってしまう場合がある。その場合は、

$$u_1(t) = K_P (v_d - v_1(t)) + K_I \int_0^t (v_d - v_1(\tau)) d\tau \quad (12-7)$$

のように積分制御を用いることで改善されることがある。積分ゲイン K_I は正の定数である。(12-7)式のように比例制御と積分制御を組み合わせた制御法を、頭文字を用いて PI 制御 (*proportional-integral control*) という。

12-2-2 旋回角速度の PID 制御

つぎに、目標軌道と y 座標の差に応じてロボットの向きを変えるように旋回方向の入力 u_2 を決めよう。目標 y_r と今の位置 $y(t)$ との差、つまり偏差 (*error*) を

$$e(t) = y_r - y(t) \quad (12-8)$$

とおく。前節の並進速度の制御と同じように PI 制御を用いると、

$$u_2(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau \quad (12-9)$$

である。 k_p , k_i はそれぞれ、比例ゲイン、積分ゲインである。

ところで、もし順調に目標に近づいている状況ならば、それ以上入力を増やさなくても自然に目標に到達するかもしれない。逆に目標から遠ざかっているときには、もっと大きな入力を入れたほ

うが早く目標に近づくかもしれない。そこで、偏差の変化にもとづいて制御入力の値を決める微分制御も加えてみよう。

$$u_2(t) = k_p e(t) + k_I \int_0^t e(\tau) d\tau + k_D \frac{de(t)}{dt} \quad (12-10)$$

k_D は微分ゲインである。比例制御、積分制御、微分制御を組み合わせたものがPID制御である。

PID制御で用いられる偏差、偏差の積分値、偏差の微分値を図で表すと図12-3のようになる。偏差 $e(t)$ は現在の測定値だけから決まるが、積算値は過去の情報を使う。そのため、積分制御を使う際には過去の情報を保存するメモリが必要になる。一方、微分値は厳密には一瞬未来の値が必要である。しかし、現実的には未来の値はわからないので、プログラムで実現するときは、近似的に直前の値と現在値との差を使って求めることが多い。

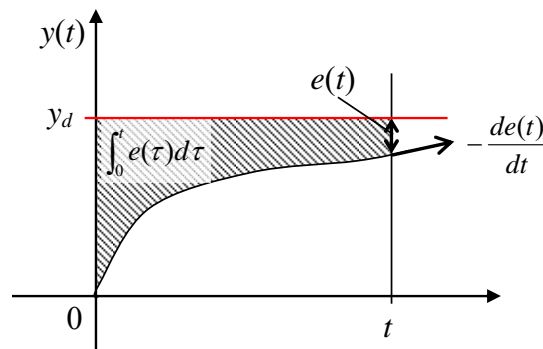


図 12-3 比例，積分，微分要素

12-3 実験結果と考察

(12-10)式に含まれる制御ゲイン k_p 、 k_I 、 k_D の選び方によって、目標軌道への追従の仕方は変化する。Robodesignerという小型のロボットを用いた場合の実験結果の例を示す。制御ゲインは、ロボットの特性に合わせて、並進入力の制御ゲインを $K_p=2.0$ 、 $K_I=8.0$ とした。

旋回入力の制御ゲインを① $k_p=30$ 、 $k_I=0$ 、 $k_D=45$ 、② $k_p=30$ 、 $k_I=10$ 、 $k_D=45$ として実験を行ったときのロボットの軌道を図12-4に示す。このロボットでは、①のPD制御では定常偏差が残った（黒線）。そこで、積分制御を加えてPID制御にすると、図12-4の青線のように定常偏差がほぼ0になった。ただし、積分制御を入れた場合は一度目標値を行き過ぎていた。これは、自転車や自動車の運転でコース変更を行なうときに、急にハンドルを操作して軌道が揺れてしてしまうことに似ている。希望の軌道が得られるように制御ゲインを調整してほしい。

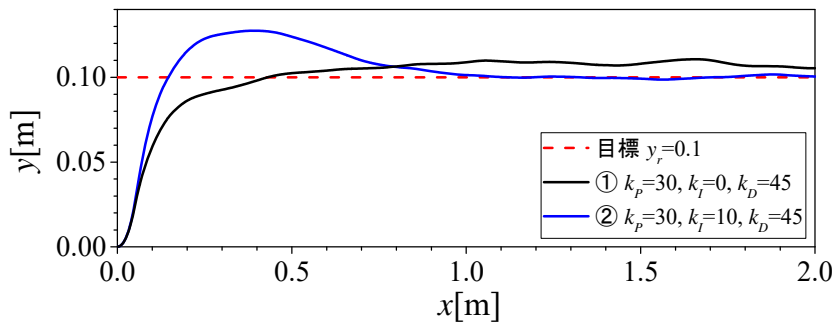


図 12-4 実験でのロボットの軌道例（PID制御）

12-4 軌道制御の理論的な考察

PID 制御では、多くの場合、経験にもとづいてゲインを決める。例えば、一般に積分制御を入れることで十分時間が経ったときの偏差（定常偏差）の改善が期待できる。比例制御や微分制御で目標値に近づく速さを調整する。ゲインの組み合わせによっては、ロボットは全く軌道に追従しなくなってしまうこともある。PID ゲインを決める方法には、経験的な知見にもとづいた限界感度法¹⁾や、運動モデルをたてて理論的な知見にもとづいて決める極配置法²⁾などがある。制御理論にもとづいて考えることで、例えば、微分制御を入れるべきかどうか、おおよそどのようにゲインを選んだら目標を行き過ぎるオーバーシュートを抑えられるかなどが、実験を行う前に予想できることがある。

ここでは、古典制御を学んだ人のために、古典制御の考え方と運動モデル(12-3)~(12-5)式にもとづいてロボットの軌道制御について考察してみよう。まず、古典制御の考え方をを用いるために、

仮定 1 : 並進速度は一定値であり、正の値とする。

仮定 2 : 姿勢角度 θ は十分小さい。

とする。仮定 1 は第 12-2-1 節の制御が十分機能している場合、ある程度の時間以降では妥当だと考えてよいだろう。仮定 2 は急な旋回をすることなく、直線に沿って移動している場合には満たされると考えることができるだろう。

仮定 1 および 2 が成り立つとすると、(12-3)式は、

$$\frac{dx(t)}{dt} = v_1 \text{ (const.)}$$

となるので、 x 方向の位置は $x(t) = v_1 t$ で変化する。一方、 y 方向の運動は(12-4)式の両辺を微分した式、

$$\begin{aligned} \frac{d^2 y(t)}{dt^2} &= \dot{v}_1(t) \sin \theta(t) + v_1(t) \cos \theta(t) \cdot \frac{d\theta(t)}{dt} \\ &\approx v_1 \frac{d\theta(t)}{dt} \\ &= v_1 v_2(t) \end{aligned} \quad (12-11)$$

で表わされる。したがって、 v_2 から y までの伝達関数は、

$$\frac{Y(s)}{V_2(s)} = v_1 \frac{1}{s^2} \quad (12-12)$$

であり、図 12-5 のように PD 制御、すなわち、

$$v_2 = k_p(y_r - y(t)) + k_d(\dot{y}_r - \dot{y}(t)) \quad (12-13)$$

を用いると、 y_r から y までの伝達関数は、

$$\begin{aligned} \frac{Y(s)}{Y_r(s)} &= \frac{(k_p + k_d s) v_1 \frac{1}{s^2}}{1 + (k_p + k_d s) v_1 \frac{1}{s^2}} \\ &= \frac{v_1 k_d s + v_1 k_p}{s^2 + v_1 k_d s + v_1 k_p} \end{aligned} \quad (12-14)$$

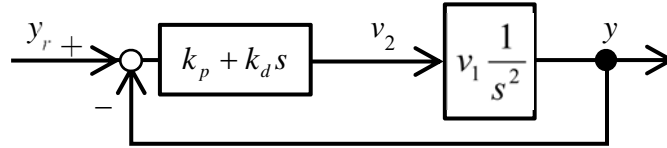


図 12-5 フィードバック制御系

となる．特性多項式が $s^2 + v_1 k_d s + v_1 k_p$ なので， $v_1 > 0$ ならば， $k_p, k_d > 0$ がシステムを安定にするための必要十分条件である．もし，制御則(12-13)に微分項を入れない ($k_d = 0$ とする) とシステムが安定限界となり，軌道が振動して一定値に収束しないことが予想できる．

(12-12)式で表されるシステムに PD 制御(12-13)を用いた場合は，十分時間が経ったときに一定の目標軌道 y_r に収束することが期待できる．これは，(12-14)式に最終値の定理を用いれば確認することができる．しかし，実際に実験を行うと前節の図 6 の①の場合のように，偏差が残ってしまうことがある．この原因の一つとして，前述のモデルではモータの動特性や摩擦などを考えていないことが考えられる（そのため，前節の実験では積分項も加えた PID 制御を用いて定常偏差を 0 に近づけた）．図 12-6 に数値シミュレーション結果を示す．このシミュレーションでは制御対象は(12-3)～(12-5)の非線形モデルを用いており，制御ゲインは図 12-4 の①と同じものを用いている．シミュレーションでは積分制御を入れなくても目標値に収束している．

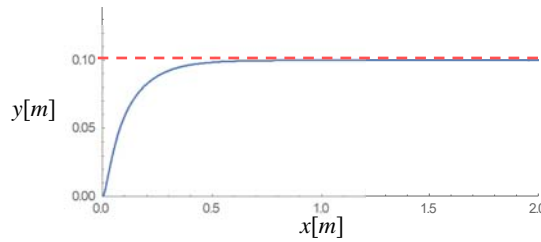


図 12-6 シミュレーション結果(非線形モデル)

ところで，実際の実験では並進速度 v_1 は一定値ではなかった．目標値 v_d に収束するまでの間や外乱（例えば床の摩擦の変化）などによって v_1 が変動した場合は仮定 1 が満たされず，(12-10)式の制御則では軌道が変動してしまいます．また，図 12-4 の実験例では途中で角度 θ が 60 度近くになっており，仮定 2 が満たされていない．図 12-4 の実験②と同じ制御ゲインを用いて，線形近似したモデル(12-12)式を用いた数値シミュレーション結果（図 12-7）と，非線形モデル(12-3)～(12-5)を用いた数値シミュレーション結果(図 12-8)は比較してみよう．非線形モデルを用いた場合のシミュレーション結果は，実験結果(図 12-4)と比較的よく合っている．一方で，線形近似したモデルを用いたシミュレーション結果の軌道は実験と大きく異なっている．このように，線形近似したモデルでの知見にもとづいて制御ゲインを選定することには限界がある．

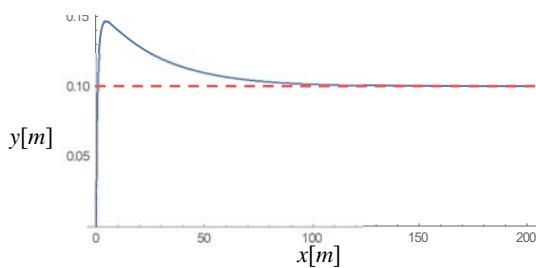


図 12-7 線形近似モデルでのシミュレーション結果

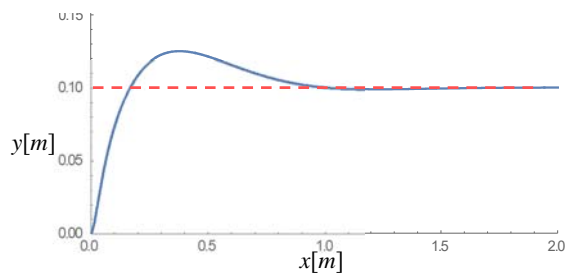


図 12-8 非線形モデルでのシミュレーション結果

12-5 非線形制御

しかし、仮定 1, 仮定 2 が満たされない場合でも、(12-3)~(12-5)式に含まれる非線形性を考慮して設計した制御則を用いるとよりうまく軌道制御を行うことができる。詳しくは参考文献 3 を参照してほしい。ポイントとしては、座標変換と入力変換、それに時間軸の置き換えを用いて第 11 章で説明した厳密な線形化手法を用いる。得られる制御則は三角関数を含む非線形制御則である。この制御法を適用した場合の走行軌道例を図 12-9 に紫の太線で示す。線形の PID 制御の青線よりも定常状態での軌道の変動が減少している。また、この実験例ではオーバーシュートをしないように制御ゲインを調整したが、その調整も比較的容易だった。

同じロボットでも制御法や制御パラメータをうまく選ぶことで走行軌道を改善できる可能性がある。いろいろな制御法を学んで試してみしてほしい。

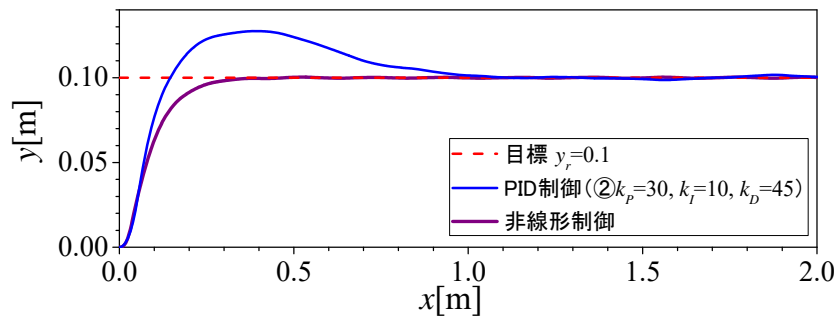


図 12-9 別の制御法を用いた場合の軌道の例（非線形制御）

参考文献

- 1) 古典制御の本（たとえば，吉川著，古典制御理論，コロナ社(2014)）
- 2) 現代制御の本（たとえば，小郷，美多著，システム制御理論入門，実教理工学全書(1980)）
- 3) 山川，時間軸状態制御形にもとづいた車輪型倒立振り子ロボットの軌道追従制御，計測自動制御学会論文集 49-10, pp.936-943(2013)

13. ロボットの軌道制御の実験（後日追加）

付録 A : Mathematica と MATLAB/Simulink

◆Mathematica と MATLAB のコマンド対応

	Mathematica	MATLAB
変数定義	不要	syms t
行列の定義	$A = \{\{1,0\},\{0,1\}\}$	$A = [-1 _0; 0 _ -1]$ ←空白で区切る
指数関数(スカラー) e^x	Exp(x)	exp(x)
行列の指数関数 e^A	MatrixExp[A]	expm(A)
行列の掛け算	$A.B$	$A*B$
行列の転置	Transpose[A]	transpose(A)
単位行列 ($n \times n$)	IdentityMatrix[n]	eye(n)
階乗 $n!$	$n!$	factorial(n)
行列式 $ A $	Det[A]	det(A)
逆行列 A^{-1}	InverseMatrix[A]	inv(A)
微分 $\frac{d}{dt}x(t)$	D[x(t), t]	diff(x(t), t)
積分 $\int x(t)dt$	Integrate[x(t), t]	int(x(t), t)
ラプラス変換	LaplaceTransform[x(t), t, s]	laplace(x(t), t, s)
逆ラプラス変換	InverseLaplaceTransform[x(t), s, t]	ilaplace(x(t), s, t)

◆古典制御のための MATLAB のコマンド

	MATLAB のコマンド
伝達関数 $g = (a_1s+a_0) / (b_1s+b_0)$ を作る	$g = tf([a_1, a_0], [b_1, b_0])$
ボード線図を描く	bodeplot(g)
ゲイン余裕, 位相余裕求め, ボード線図を描く	margin(g)
伝達関数 g の極を求める	pole(g)

◆現代制御のための MATLAB のコマンド

	MATLAB のコマンド
状態空間表現のシステムを定義	$H = ss(A, b, c, 0)$
可制御性行列	$Cc = ctrb(A, b)$, または, $Cc = [b, A*b, A*A*b \dots]$
可観測性行列	$Co = obsv(A, c)$, または, $Co = ([c; c*A; c*A*A \dots])$
行列のランク	rank(Cc)
行列の固有値	eig(A)
伝達関数行列	tf(H), または, $c * inv(s * eye(4) - A) * b$
極配置(Ackermann の方法)	acker(A, b, p) p は, 配置したい極を並べたベクトル $p = [p_1, p_2, \dots, p_n]$ 答えは, $u = -kx$ の k . 符号に注意.
最適レギュレータ	lqr(A, B, Q, R, N) $\dot{x} = Ax + Bu$ 評価関数は, $J = \int_0^{\infty} (x^T Qx + u^T Ru + 2x^T Nu) dt$ N は cross term という. 通常は 0 でよい. 答えは, $u = -kx$ の k . 符号に注意.

◆MATLAB の「m ファイル」を作る

コマンドラインで複数の命令を繰り返し行う場合は、m ファイルを作成すると便利です。

「ファイル」→「新規作成」→「M-ファイル」とすると、新規画面が出ます。

コマンドを書き込んで、「デバッグ」→「実行」で実行します。ファイルに書かれたコマンドをまとめて実行します。結果は、コマンドウィンドウに表示されます。

◆Mathematica のコマンドなど

簡単な関数（関数は大文字から始まる。引数は[]でくくる）

- ・平方根：Sqrt[x], 三角関数：Sin[x], Cos[x], Tan[x], ArcSin[x]など（角度は radian）,
- 対数：Log[x]（自然対数），指数関数：Exp[x],
- ラプラス変換：LaplaceTransform[e^{-t}, t, s], 逆ラプラス変換：InverseLaplaceTransform[s+1, s, t]

関数の定義

- ・関数：x[t]. x は t の関数である
- ・関数の微分：x'[t]. . . . x の t による一階微分
- ・微分：D[f(t), t] . . . f(t)を t で微分する
- ・不定積分：Integrate[f(t), t] . . . f(t)の不定積分

有理関数の分母

Denominator[a(s)/b(s)]. . . 有利関数の分母 b(s)を取り出す

方程式を解く

Solve[b(s)=0, s]. . . s の方程式 b(s)=0 を s について解く

NSolve[b(s)=0, s]. . . s の方程式 b(s)=0 を s について数値的に解く

FindRoot[b(s)=0, {s, s0}]. . . s の方程式 b(s)=0 について s=s0 の近傍から解を探索する
(一つの解を見つけるだけ。一度に全ての解は得られない)

実部と虚部

Re[x]. . . 変数 x の実数部分を取り出す

Im[x]. . . 変数 x の虚数部分を取り出す

◆Mathematica でグラフを描く


Plot[f(t), {t, 0, 10}, PlotRange->{{t_{min}, t_{max}},{f_{min}, f_{max}}}

関数 f(t)を t=0 から t=10 までプロット。


ただし、グラフは横軸が t_{min} から t_{max}, 縦軸は f_{min}, f_{max} の範囲。

付録 B：MATLAB/Simulink の使い方

◆Simulink の起動方法

MATLAB を起動した後，コマンドウィンドウで“simulink”と入力するか，ツールバーのをクリックする．Simulink Library Browser が起動したら，メニューの「新規作成」→「モデル」で新規画面を起動する．

◆伝達関数のステップ応答をみる

1. Simulink Library Browser の「Continuous」→「Transfer Fcn」の伝達関数のブロックをドラッグアンドドロップで，モデルの新規画面に持っていく(図 B-1 上)．
2. 画面に貼り付けたモデルのブロックをダブルクリックすると，図 B-2 のような画面が表示されるので，伝達関数のパラメータを入力する．たとえば， $G(s) = \frac{b_1s + b_0}{a_2s^2 + a_1s + a_0}$ という伝達関数ならば，分子係数を[b_1 b_0]，分母係数を[a_2 a_1 a_0]と入力する．
3. 一定値入力(ステップ入力)を入れるために，「Commonly Used Block」→「Constant」をドラッグアンドドロップ．その後，マウスをクリックしながら動かして，矢印を繋げる(図 B-1 中)．
4. 「Sinks」の Scope をモデル画面へ貼り付けて，伝達関数ブロックの出力側に繋ぐ(図 B-1 下)．
5. Scope をダブルクリックして，グラフウィンドウを表示させる．
6. メニューの「シミュレーション」→「開始」をクリックするか，ツールバーのをクリックするとシミュレーションが開始する．

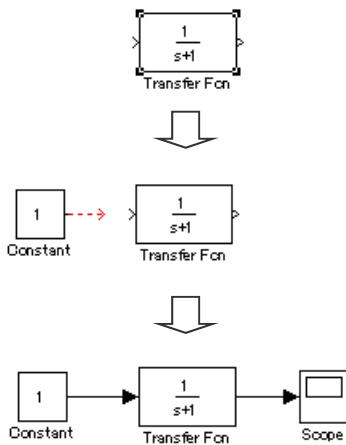


図 B-1 Simulink 上でのブロック線図



図 B-2 伝達関数のパラメータ設定画面

※ シミュレーションを行う場合，「コンフィギュレーションパラメータ」の設定で，「相対許容誤差」を小さくすると，計算精度が上がる．

付録 C：ラプラス変換

◆ ラプラス変換・逆ラプラス変換でよく使う関数

時間関数	ラプラス変換	複素関数
$\delta(t)$	\longleftrightarrow	1
1	\longleftrightarrow	$\frac{1}{s}$
t	\longleftrightarrow	$\frac{1}{s^2}$
e^{-at}	\longleftrightarrow	$\frac{1}{s+a}$
$\cos \omega t$	\longleftrightarrow	$\frac{s}{s^2 + \omega^2}$
$\sin \omega t$	\longleftrightarrow	$\frac{\omega}{s^2 + \omega^2}$

◆ ラプラス変換・逆ラプラス変換の諸定理

$\mathcal{L}[f(t)] = F(s)$

1. 線形性 $\mathcal{L}[a f(t) + b g(t)] = a F(s) + b G(s)$
2. 微分 $\mathcal{L}[\dot{f}(t)] = sF(s) - f(0)$
3. 積分 $\mathcal{L}\left[\int_0^t f(\tau) d\tau\right] = \frac{1}{s} F(s)$
4. むだ時間 $\mathcal{L}[f(t-T)] = e^{-sT} F(s)$
5. e^{-at} との積 $\mathcal{L}[e^{-at} f(t)] = F(s+a)$
6. たたみ込み積分 $\mathcal{L}\left[\int_0^t f(t-\tau) g(\tau) d\tau\right] = F(s)G(s)$
7. 初期値の定理 $\lim_{t \rightarrow 0} f(t) = \lim_{s \rightarrow \infty} sF(s)$
8. 最終値の定理 $\lim_{t \rightarrow \infty} f(t) = \lim_{s \rightarrow 0} sF(s)$

付録 D：移動ロボットのオドメトリ

自動制御によって目標軌道に沿ってロボットを動かそうとするとき、今ロボットがどこにいるのかが分からなければ、目標の場所に向かうことはできない。実際にロボットの軌道制御を行う場合には、ロボットの位置を知る必要がある。

たとえば、単純なラインレースロボットは地面に描かれた目標軌道を赤外線センサなどで読み取り、ロボットと目標軌道との相対的な位置を判断している。一方、自動車などではいつでも軌道が目で見えるような線で描かれているわけではない。そのため、例えば GPS を用いて自分の位置を計測することがある。しかし、トンネルに入ったときなどは GPS 衛星からの信号が受信できないので、判っている位置からどれくらい走行したかによって今の位置を推定する。これをオドメトリ (*odometry*) という。自動車はタイヤが一回転すれば、タイヤの直径×円周率だけ進むし、片輪だけが回転すれば旋回する。つまり、左右のタイヤがどれだけ回転したかを測定して積算すれば、今どこにいるかを推定できる。タイヤの回転は、エンコーダを取り付けることで測定することができる。ここでは、この測定値を利用して水平面内での位置を推定する方法を説明しよう。

まず、ロボットのタイヤは地面に対して滑らないと仮定する。もし、タイヤが滑ってしまうとタイヤが回転していなくてもロボットの位置が変化する。この場合にはそもそもタイヤの回転角度からロボットの位置を推定することはできず、別のセンサ（加速度センサなど）が必要となる。タイヤが地面に対して滑らないという仮定は、自動車が凍った路面でスリップする場合や、急発進や急ブレーキをかけたときにタイヤが空転している場合などでは満たされない。一方、建物の廊下などを比較的ゆっくり走るロボットでは満たされると考えることができる。つまり、ゆっくり走行するロボットの制御を考える問題においては、この仮定は「妥当」だと言えるだろう。

ロボットの運動は(12-1)～(12-3)式の微分方程式にしたがっているとすると、時間によって値が変化する関数 $x(t)$ の時間微分の定義は、

$$\frac{dx(t)}{dt} = \lim_{\Delta t \rightarrow 0} \frac{x(t) - x(t - \Delta t)}{\Delta t} \quad (\text{D-1})$$

である。ここで Δt が十分小さいと仮定して次のように近似する。

$$\frac{dx(t)}{dt} \approx \frac{x(t) - x(t - \Delta t)}{\Delta t} \quad (\text{D-2})$$

つまり、 Δt は十分小さいので極限を取っても取らなくても同じだと考える。(D-2)式を移項して変形すると、

$$x(t) = x(t - \Delta t) + \Delta t \frac{dx(t)}{dt} \quad (\text{D-3})$$

となり、この右辺 2 項目に(12-3)式を代入すると、

$$x(t) = x(t - \Delta t) + \Delta t (v_1(t) \cos \theta(t)) \quad (\text{D-4})$$

となる。 $y(t)$ 、 $\theta(t)$ についても同様に、

$$y(t) = y(t - \Delta t) + \Delta t (v_1(t) \sin \theta(t)) \quad (\text{D-5})$$

$$\theta(t) = \theta(t - \Delta t) + \Delta t v_2(t) \quad (\text{D-6})$$

が得られる。(D-4)～(D-6)式の左辺は時刻 t での位置と角度である。一方、右辺は時刻 $t - \Delta t$ での位置と角度、それに時刻 t での速度の項である。つまり、これらの式は、過去の情報と測定した速度

の情報から，時刻 t での位置と角度を計算する式である．この式を繰り返して用いると，初期時刻 $t=0$ での位置と角度 $x(0)$, $y(0)$, $\theta(0)$ と各時刻での速度 $v_1(t)$, $v_2(t)$ から，各時刻での位置と角度を得ることができる．

つぎに，ロボットの速度 v_1 と v_2 を左右の車輪モータについているエンコーダの測定値から算出しよう．まず，エンコーダは一定の角度回転ごとにパルス信号を発生する．このパルス数をカウントすることでモータの回転角度を測定することができる．多くの場合，モータと車輪の間には減速させるためのギア（歯車）が組み込まれているので車輪の回転数はモータの回転数の定数倍である．ここでは，車輪が一回転するときエンコーダが C 回のパルス信号を発生するとする．この C の値はエンコーダの仕様とモータのギア比で決まる定数なので，実際にロボットを動かす場合には対象のロボットの仕様に合わせて必要がある．

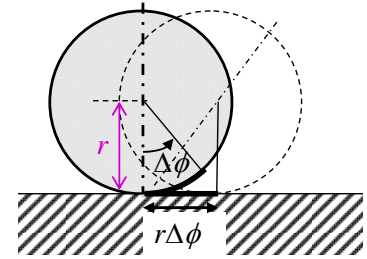


図 D-1 車輪が進む距離

エンコーダが発生したパルス数をカウントしたものをエンコーダ値と呼ぶことにする．エンコーダ値が 1 増えた場合，その間に車輪は $2\pi/C$ [rad] だけ回転する．時刻 $(t-\Delta t)$ から t までの間に右車輪のエンコーダ値が $encR(t-\Delta t)$ から $encR(t)$ に変化したとすると，右車輪は角度，

$$\Delta\phi_r = \frac{2\pi}{C}(encR(t) - encR(t-\Delta t)) \quad (D-7)$$

だけ回転する．車輪の半径を r とすると右車輪の接地点が地面に対して進む距離 ΔL_r は，角度 \times 半径 r なので，

$$\Delta L_r = \frac{2\pi r}{C}(encR(t) - encR(t-\Delta t)) \quad (D-8)$$

である（図 D-1）．同じく時刻 t での左側のエンコーダ値を $encL(t)$ とおくと，左車輪が進む距離 ΔL_l は，

$$\Delta L_l = \frac{2\pi r}{C}(encL(t) - encL(t-\Delta t)) \quad (D-9)$$

である． Δt の間に ΔL_r , ΔL_l 進むから，左右の車輪の接地点での地面に対する速度は，それぞれ

$$\begin{aligned} v_r(t) &= \frac{\Delta L_r}{\Delta t} \\ v_l(t) &= \frac{\Delta L_l}{\Delta t} \end{aligned} \quad (D-10)$$

である．ロボットの並進速度 v_1 とは左右車輪の中心位置の速度，つまり左右の車輪の平均速度であるので，(D-8)～(D-10)式を用いれば，

$$\begin{aligned} v_1(t) &= \frac{v_r(t) + v_l(t)}{2} \\ &= \frac{\pi r}{C} \frac{(encR(t) + encL(t)) - (encR(t-\Delta t) + encL(t-\Delta t))}{\Delta t} \end{aligned} \quad (D-11)$$

と得られる．一方， Δt での姿勢角度の変化 $\Delta\theta$ は左右の車輪間の距離が d であるとき，図 D-2 から，

$$\Delta\theta(t) = \frac{\Delta L_r(t) - \Delta L_l(t)}{d} \quad (D-12)$$

である。したがって、旋回角速度 v_2 は、

$$\begin{aligned} v_2(t) &= \frac{\Delta\theta(t)}{\Delta t} \\ &= \frac{2\pi r (encR(t) - encL(t)) - (encR(t - \Delta t) - encL(t - \Delta t))}{Cd \Delta t} \end{aligned} \quad (D-13)$$

と得られる。以上のように、左右のエンコーダ値からロボットの速度 v_1 , v_2 を算出することができる。

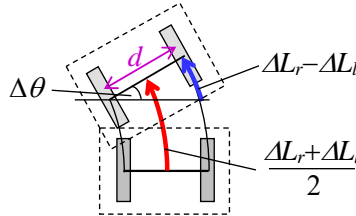


図 D-2 左右車輪の移動距離の平均と差

以上の関係を用いて、エンコーダ値から x , y , θ を求めるプログラムを書いていこう。まず、角度を計算する(D-6)式に(D-13)式を代入すると、

$$\theta(t) = \theta(t - \Delta t) + \frac{2\pi r}{Cd} ((encR(t) - encL(t)) - (encR(t - \Delta t) - encL(t - \Delta t))) \quad (D-14)$$

である。この代入を繰り返して遡っていくと、途中の時刻でのエンコーダ値は打ち消しあって、

$$\theta(t) = \theta(0) + \frac{2\pi r}{Cd} (encR(t) - encL(t)) \quad (D-15)$$

が残る。ただし、初期時刻の $encR(0)$, $encL(0)$ は 0 である。結果的に、角度は繰り返し計算をしなくても初期時刻の角度 $\theta(0)$ と現時刻のエンコーダ値から求めることができる。

一方、位置については(D-4)(D-5)式で各時刻の角度 $\theta(t)$ に依存した $\cos\theta(t)$ や $\sin\theta(t)$ が掛け算されているので、(D-15)式のように途中の値は打ち消されない。そのため、繰り返し計算を行って積算する。プログラムに何度も同じ計算を書くのは見づらくなるので、現時刻 t での左右エンコーダ値の和を sum とおく。

$$sum = encR + encL \quad (D-16)$$

繰り返し計算の一回分の処理にかかる時間を Δt とし、 $(t - \Delta t)$ 時刻での値を sum_0 とおくことにする。同様に、時刻 t での状態を x , y とし、1 回前の $(t - \Delta t)$ 時刻の状態をそれぞれ x_0 , y_0 と添え字 0 をつけて表わす。プログラム開始時には、 $t=0$ での初期値を x_0 , y_0 として定義しておく。つぎに、(D-4)(D-5)式、および(D-16)式を用いて、現時刻 t の状態をエンコーダの値からつぎのように計算する。

$$x = x_0 + \left(\frac{\pi r}{C} (sum - sum_0) \right) \cos \theta \quad (D-17)$$

$$y = y_0 + \left(\frac{\pi r}{C} (sum - sum_0) \right) \sin \theta \quad (D-18)$$

この式では Δt は打ち消し合って消えるので、処理にかかる時間 Δt を計測する必要はない。しかし、実際には Δt が十分小さくない場合には(D-7)式の近似が成り立たなくなり、実際の位置と計算値に誤差が生じるので気にとめておく必要がある。

さて、繰り返し計算を行なうとき、つぎの計算を行う時点では現時刻の状態を一回前の状態として扱う。そのため、繰り返し計算の最後で変数の更新を行う。すなわち、 x_0 、 y_0 および sum_0 に現時刻の状態を代入して更新し、つぎの計算に備える。

$$sum_0 = sum \quad (D-19)$$

$$x_0 = x \quad (D-20)$$

$$y_0 = y \quad (D-21)$$

以上の(D-16)～(D-21)式の計算を繰り返せば、自己位置を計算することができる。

付録 E：PID 制御のプログラム

第 12 章の制御を実現するためのプログラムを示す。対象としたロボットは RoboDesignerRDS-X25 である。このロボットでは速度を直接入力できるわけではないので、以下の制御プログラムでは計算した制御入力 u_1, u_2 に km という定数をかけて入力としている。 km は実際の入力値にあわせて選んだ。

繰り返し計算について付録 D で説明したが、C 言語のプログラムでは $x=x+\alpha$ という記述が右辺の値を左辺に代入するという処理であることを利用して、(D-17)式と(D-25)式をまとめて処理している。そのため、 x_0, y_0 の定義は不要である。

このプログラムでは、シリアル通信でデータを PC へ送信している。送信するデータ数やデータの型によって通信に要する時間が変わるので、繰り返し計算において 1 回の処理にかかるサンプリング時間 Δt も変わる。シリアル通信を行わない場合のように Δt が短すぎると、今回用いるロボットではエンコーダの読み取りが間に合わず、エンコーダの読み飛ばしが生じることがある。この場合、オドメトリで推定した値に対して実際の走行距離が長くなる。一方、通信量を増やしすぎてサンプリング時間 Δt を長くすることは、付録 D で説明した位置計算の精度を下げるだけでなく、制御入力の更新を間引くことにもなり、軌道追従性などの制御性能を悪くする。

```

/*****
Tracking controller for Robo Designer    by S.Yamakawa 20170721
RDS-X25 用, 開発環境は Arduino 1.0.5-r2
PID 制御を用いて目標軌道  $y_r(x)$  に追従させる
*****/
//=====ロボットのパラメータ (※ロボットに合わせて変更) =====
float d = 0.118;           //タイヤ間距離[m]
float r = 0.064;           //タイヤ直径[m]
float a = r*3.14159265/768.0; //1pulse あたり進む距離[m], ギア比 48x1 回転 16pulse
//=====ロボット制御の目標値と制御ゲイン (※自分で設定・調整する) =====
float vd = 0.1;           //並進速度目標値[m/s]
float yr = 0.1;           //目標軌道[m]
float xd = 2.0;           //x 軸方向目標位置[m]
float kp = 30;            //比例ゲイン
float ki = 10;            //積分ゲイン
float kd = 45;            //微分ゲイン
//=====基板のピン設定=====
int M1_1 = 4;
int M1_2 = 5;
int M1_PWM = 6;
int M2_1 = 7;
int M2_2 = 8;
int M2_PWM = 9;
int P_PUSH = 12;
int encoder1 = 2;
int encoder2 = 3;
//=====オドメトリ・制御用パラメータ=====
volatile int enCounter_r, enCounter_l; //エンコーダの直近のカウンタ数
int sum = 0; // (右エンコーダ) + (左エンコーダ) のカウンタ値
int sum0 = 0; // 1 時刻前の値
float vr = 0.0; //右タイヤの目標速度[m/s]
float vl = 0.0; //左タイヤの目標速度[m/s]
float x = 0.0; //車軸中心位置[m]
float y = 0.0; //車軸中心位置[m]

```

```

float theta = 0.0;           //車両向き[rad]
float theta_old = 0.0;
float v1 = 0.0;             //並進速度[m/s]
float v2 = 0.0;             //旋回角速度[rad/s]
float u1, u2;               //並進, 旋回角速度入力[m/s]
float vli = 0.0;           //v1 の偏差の積算値
float e;                     //偏差
float e0 = yr;              //1時刻前の偏差
float ei = 0.0;             //偏差の積算値
float ed = 0.0;             //偏差の微分値
long time;                   //時刻[ms]
long time_old = 0;          //一回前の時刻[ms]
long dt = 0;                 //サンプリング時間[ms]
float temp;

/*****
  初期化処理
*****/
void setup(){
  Serial.begin(9600);
  pinMode(M1_1, OUTPUT);
  pinMode(M1_2, OUTPUT);
  pinMode(M1_PWM, OUTPUT);
  pinMode(M2_1, OUTPUT);
  pinMode(M2_2, OUTPUT);
  pinMode(M2_PWM, OUTPUT);
  pinMode(P_PUSH, INPUT_PULLUP);
  attachInterrupt(encoder1, count_l, CHANGE); //エンコーダ値取得開始
  attachInterrupt(encoder2, count_r, CHANGE);
  enCounter_r = 0; //エンコーダ初期値設定
  enCounter_l = 0;
  //=====ボタンを押してから 0.5 秒後にスタート=====
  while( digitalRead(12) == 1 ){};
  delay(500);
  time_old = millis();
}

/*****
  繰り返しメインループ
*****/
void loop(){
  //=====自己位置計算=====
  time = millis();
  dt = time - time_old; //経過時間[msec]
  time_old = time;
  theta =float(enCounter_r - enCounter_l)*a/d; //旋回角度[rad]
  if(dt != 0){
    v2 = (theta-theta_old)/dt*1000.0; //旋回角速度[rad/s]
    sum = enCounter_r + enCounter_l;
    temp = float(sum-sum0);
    v1 = temp/2.0/float(dt)*1000.0*a; //並進速度[m/s]
    x = x + (temp*cos(theta)*a/2.0); //車軸中心 x 座標[m]
    y = y + (temp*sin(theta)*a/2.0); //車軸中心 y 座標[m]
    sum0 = sum;
  };
  theta_old = theta;

```

```
//=====制御則=====
vli = vli + (vd - v1)*float(dt)/1000.0;
u1 =2.0*(vd - v1)+ 8.0*vli; //並進方向速度入力(PI control)
e = (yr - y);
ei = ei + e*float(dt)/1000.0;
ed = (e - e0)/float(dt)*1000.0;
u2 = kp*e + ki*ei + kd*ed; //旋回方向速度入力(PID control)
e0 = e;
vr = u1 + u2*D-15;
vl = u1 - u2*D-15;

if(x > xd){
    vl = 0;
    vr = 0;
}

//=====モータへの出力=====
motorDrive(M1_1, M1_2, M1_PWM, vl);
motorDrive(M2_1, M2_2, M2_PWM, vr);
//=====シリアル通信=====
Serial.print(x,DEC); //DEC は送信に1つにつき 3ms くらいかかる
Serial.print(",");
Serial.print(y,DEC);
Serial.print(",");
Serial.print(dt);
Serial.print(",");
Serial.print(v1*1000);
Serial.print(",");
Serial.println(v2*1000);
}

/*****
モータ入力の関数
*****/
void motorDrive(int m1Pin, int m2Pin, int PWMpin, float v){
    int maxDuty = 150; //大きすぎるとエンコーダの読み飛ばしができる
    int km = 600.0; //モータデューティ比と速度の比
    int duty;
    if (v > 0){ //カウントが到達目標より小さかったら正転
        digitalWrite(m1Pin, HIGH);
        digitalWrite(m2Pin, LOW);
        duty=(int)(v*km); //速度から duty 比への変換
    }else if(v < 0){
        digitalWrite(m1Pin, LOW);
        digitalWrite(m2Pin, HIGH);
        duty=(int)(-v*km); //速度から duty 比への変換
    }else{
        digitalWrite(m1Pin, LOW);
        digitalWrite(m2Pin, LOW);
        duty = 0;
    }
    if(duty > maxDuty) duty=maxDuty; //リミッタ
    analogWrite(PWMpin, duty); // モータの PWM 設定
}

/*****
エンコーダのカウント割り込み関数
*****/
```



```
void count_l(){
  if (vl >= 0) enCounter_l++; //電圧入力为正ならば正転していると仮定
  if (vl < 0) enCounter_l--;
}
void count_r(){
  if (vr >= 0) enCounter_r++;
  if (vr < 0) enCounter_r--;
}
```