制御系設計に状態遷移表が必要なわけ

サイバネットシステム株式会社
モデルベース開発推進室　執行役員　CTO
石塚　真一

1. はじめに

「制御」あるいは「制御系設計」という言葉は誤解を生みやすいと思います。たとえば、モータを正確に回すことを考えている人達、いわゆる制御理論屋にとって「状態遷移表」や「FSM：有限状態マシーン」と言っても、いったい何のことだか、それが必要なものか分からないかもしれません。これは制御系設計ツールで世界的にデファクトスタンダードとなっているMATLABの開発の歴史を見ても明らかです。MATLABにはFSMを記述するツールとしてStateflowというものがあります。しかしStateflowがリリースされたのは、MATLABが制御系設計ツールとして認識されたずっと後のことでした。それくらい制御理論屋にとって、FSMは縁遠い存在だったのです。

一方、マイコンの組込みプログラマにとっては、制御プログラムと言えば、状態遷移図に代表されるFSMが当たり前のように頭に浮かび、制御理論屋が好んで使う状態空間実現であるとかロバスト制御といった設計手法を思い浮かべることはあまり無いのではないでしょうか。

「離散」という概念においても同様なことが言えます。組込みプログラマにとって、離散システムと言えばFSMそのものですが、制御理論屋にとっては離散システムというと、それは微分方程式（連続系）を実装に適するように「離散化」した差分方程式（離散系）で表現されたシステムを意味します。

このように同じ「制御」でも、理論系と実装系では全くとらえ方が違っており、これが、いわゆる「モデルベース開発」の普及の障害になっているのも事実であると思います。

本解説は、本誌を読む方は実装エンジニアが大半であろうと考え、制御理論の立場から制御系設計を整理し、FSM/状態遷移表の役割と可能性を、筆者のおもむくままに書いた読み物的なものです。

2. 制御系設計レビュー

制御理論に基づく設計とはいかなるものでしょうか。現代的な制御設計アプローチは、いわゆる「モデルに基づく設計」です。近年、MBD（Model-Based Development/Model-Based Design）という言葉が色々なところで使われるようになり、その使われ方も広がっているようにも思えます。制御系設計の世界では1960年頃から使われている概念で、約50年もの歴史を持ちます。ここでは簡単な例により具体的な設計手順を説明し、普段、皆様が使われているMBDとどこが同じでどこが違うかを、明確に認識したいと思います。

制御設計におけるモデルに基づく設計とは、「制御対象の数理モデルに基づく理論的なコントローラの設計」を意味します。これを、簡単な振動制御問題を例に具体的に説明します。

1.1 制御対象

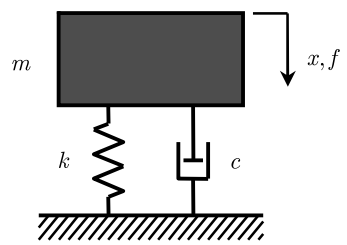

図1　1自由度振動モデル

図１は自動車のサスペンション制御などの基本となる1自由度振動モデルです。現代的な制御ではこのダイナミクスを微分方程式系で表現します。多くの場合、(1) 式のニュートンの運動

方程式で表します。

$$m\ddot{x} + c\dot{x} + kx = f \qquad (1)$$

制御系設計では、(1) 式をそのままつかうのではく、通常、状態方程式という形に変換します。状態方程式というのは、変数変換により、1階の連立微分方程式に変換したもので、次のような手順で得ることができます。今、

$$\left.\begin{aligned} x_1 &= x \\ x_2 &= \dot{x} \end{aligned}\right\} \qquad (2)$$

と置き、(1) 式に代入すると、

$$\begin{aligned} m\dot{x}_2 + cx_2 + kx_1 &= f \\ m\dot{x}_2 &= -kx_1 - cx_2 + f \\ \dot{x}_2 &= -\left(k/m\right)x_1 - \left(c/m\right)x_2 + f/m \end{aligned} \qquad (3)$$

となります。これを行列形式で整理し直すと、

$$\begin{Bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{Bmatrix} = \begin{bmatrix} 0 & 1 \\ -k/m & -c/m \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} \{f\} \qquad (4)$$

となり、これを状態方程式と呼びます。

1.2 コントローラ設計

(4) 式は便宜的に、

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \qquad (5)$$

のようにコンパクトに表現します。$\boldsymbol{x}$ は状態ベクトルと呼ばれ、今回の振動問題（図1）では、質点の変位と速度に相当します。これをいかに効率よく抑制するか、がコントローラ設計の課題となります。そこで、設計者によって評価が違わぬよう、「ある基準の下に最適となる」ような設計を考えます。これが評価関数と呼ばれ、例えば、

$$J = \int_0^\infty \left(\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u}\right)dt \qquad (6)$$

となります。これを最小化するようにコントローラを設計します。ここで $\boldsymbol{Q}$ は制御性能に関する重みで、大きくすると性能が良くなる、つまり振動が速やかに抑制されるようになります。その代わり制御エネルギは増大し、結果的にコストが増大します。これを調整するのが $\boldsymbol{R}$ の制御入力の重みで、これを大きくすると、制御性能は悪くなりますが、制御エネルギを節約することができます。設計者はこの $\boldsymbol{Q}, \boldsymbol{R}$ の重みを調整することになるわけですが、得られたコントローラは、「入力エネルギが無駄なく制御に使われる」という意味で「最適」となります。これが数理的・理論的にコントローラを設計していく最大のメリットで、経験と勘に基づくチューニングベースの手法と大きく異なる点です。

具体的なコントローラは、

$$\boldsymbol{P}\boldsymbol{A} + \boldsymbol{A}^T\boldsymbol{P} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} + \boldsymbol{Q} = 0 \qquad (7)$$

という形の方程式から $\boldsymbol{P}$ を求めて、

$$\begin{aligned} \boldsymbol{u} &= -\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P}\boldsymbol{x} \\ &= -\boldsymbol{K}\boldsymbol{x} \end{aligned} \qquad (8)$$

というフィードバックで実現でき、ブロック線図で表現すると、図2のようになります。

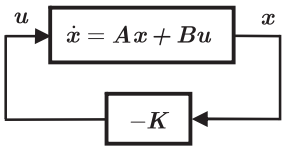

図2　フィードバック制御系

ここで、コントローラを求める基の式となっている（7）式に注目してみると、$\boldsymbol{A}, \boldsymbol{B}$ という変数が入っているのに気付きます。これは、もともとの制御対象物を表現した状態方程式（4）式の $\boldsymbol{A}, \boldsymbol{B}$ のことです。つまり、コントローラ設計時に、制御対象の特性（ダイナミクス）を積極的に取り入れて設計していることになり、この意味で「**モデルに基づく設計**」と言っています。

3. ロバスト制御とは？

2章で述べた設計法は大変、洗練されていますが、

・制御対象を正確にモデル化できない。
・モデルのパラメータにはバラツキがある。
・温度などの環境変化により特性が変わる。

といった、モデル化誤差、変動に対しての考慮がなされておらず、これらに対応すべく「ロバスト制御」なる概念が生まれてきました。今日までにいくつかの設計法が提案されておりますが、難解な数学を用いるものが多く、本誌面

では紹介しきれませんので、ここではそのイメージを図3を用いて説明します。

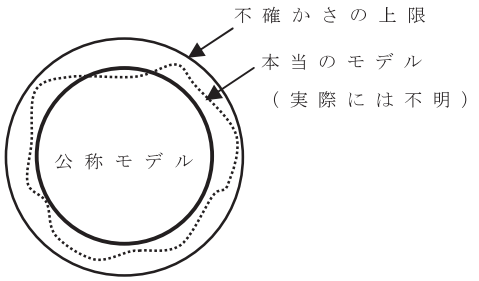

図3　ロバスト制御のイメージ

私たちは制御対象のモデル（(4) 式）を求める時は、特徴をなるべく正確に抽出しつつもシンプルなモデルを目指します。これを「公称モデル」と呼びます。本当のモデルはもっと複雑であり、バラツキや環境変化による特性変動があるわけですが、ロバスト制御では、これらを「不確かさ・摂動の集合」と考えます。これを正確に求めることは不可能ですが、「この範囲には入っているだろう」という、上限の境界を見積もることは可能であると考え、これをモデル化誤差とします。そして公称モデルに関しては要求性能を満たし、モデル化誤差があっても制御系が破綻しないようなコントローラを求めることがロバスト制御の概念です。制御系が破綻しないとは、暴走して不安定になったりせず、最低限の制御性能を保証することを意味します。ブロック線図で表すと、図4のように表されます。

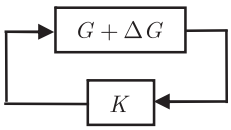

図4　ロバスト制御系

ロバスト制御は、その理論の難解さとエレガントさ、設計の困難さから、「ロバスト制御＝性能が良い」、「不確かさを吸収してくれる」といった間違った解釈、過剰な期待が寄せられた時期がありました。しかし、現在ではコントローラを状況によって切り替えていくことにより、より高性能な制御系を構成する研究も進み、そのスケジューリングにFSMは有効です。図5に環境によりFSMでコントローラを切り替える制御系を示します。

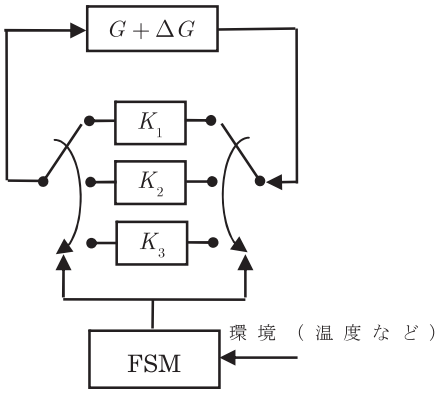

図5　FSMを用いたロバスト制御

4. ロボットアームの非線形制御

3章の制御系は、1つのコントローラが取り扱える範囲に幅を持たせて変動を吸収しようという考えで、いわゆる線形制御と呼ばれているものです。この場合、ロボットアームなどの姿勢によりそのダイナミクスが大きく変化する、非線形性の強いシステムへの適用は困難です。

ロボットアームの場合、非線形性を打ち消すフィードバックを掛け、見かけ上、線形システムとして設計する手法が考えられています。そのイメージを図6に示します。

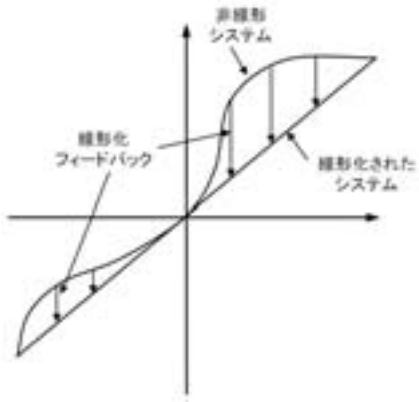

図6　非線形システムの線形化イメージ

ブロック線図で表すと図7となります。

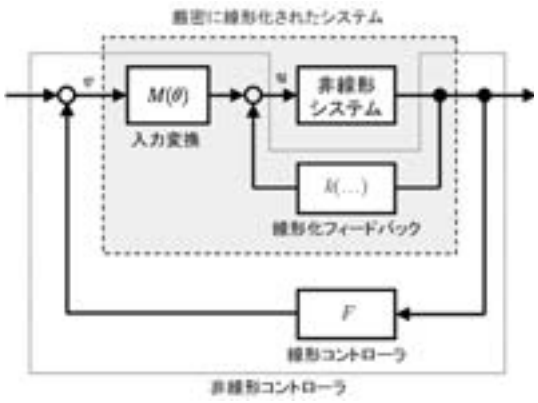

図7　線形化フィードバックによる制御系

コントローラの設計手順を説明します。ロボットアームのようなリンク機構の運動方程式は、

$$M(\theta)\ddot{\theta} + k(\theta,\dot{\theta}) = \tau$$
$M(\theta)$: 慣性モーメント
$k(\theta,\dot{\theta})$: 遠心力やコリオリカ
τ: 各リンクへのトルク　　(9)

と表されます。これに新たな入力 v を用いて、線形化フィードバック、

$$\tau = k(\theta,\dot{\theta}) + M(\theta)v \qquad (10)$$

を用いると、

$$\ddot{\theta} = v \qquad (11)$$

となります。これを状態方程式で表すと、

$$\frac{d}{dt}\begin{Bmatrix}\theta\\ \dot{\theta}\end{Bmatrix} = \begin{bmatrix}0 & I\\ 0 & 0\end{bmatrix}\begin{Bmatrix}\theta\\ \dot{\theta}\end{Bmatrix} + \begin{Bmatrix}0\\ I\end{Bmatrix}v \qquad (12)$$

となり、線形システムとなります。これに適当な線形制御を適用し、フィードバック、

$$v = F\begin{Bmatrix}\theta\\ \dot{\theta}\end{Bmatrix} \qquad (13)$$

を得ます。実際のコントローラは、これに線形化フィードバックを合わせて、

$$\tau = k(\theta,\dot{\theta}) + M(\theta)F\begin{Bmatrix}\theta\\ \dot{\theta}\end{Bmatrix} \qquad (14)$$

という形の非線形コントローラとなります。
(13) 式を見ると、リンクの角度 θ と角速度 $\dot{\theta}$ のパラメータになっていることが分かります。これを実装する場合、
①直接非線形項 $k(\theta,\dot{\theta})$ と $M(\theta)$ を計算
② $k(\theta,\dot{\theta})$ と $M(\theta)$ をマップデータで近似

といった方法が考えられます。しかし①の場合、複雑な非線形関数を計算しなくてはならず、組込み系マイコンでリアルタイムに計算するのは困難です。

一方、②の場合は変数が少ない場合は良いのですが、多変数になると多次元マップとなって指数的にメモリを消費してしまいます。

図8に示す、間接が1軸の2リンク（2自由度）マニュピュレータでも、その状態変数は、$\theta_1,\dot{\theta}_1,\theta_2,\dot{\theta}_2$ と4つあり、実際的には多軸になるので更に増え、複雑度も増し、①、②のいずれの方法も実現が困難です。

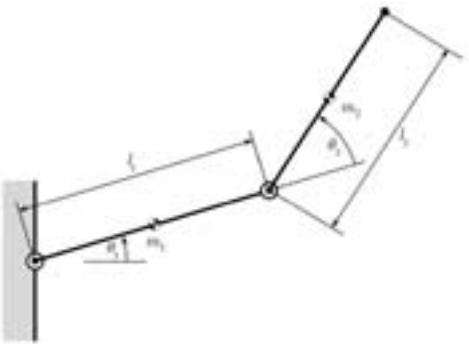

図8　2自由度マニュピュレータ

そこでマニュピュレータの使用状況を考えます。マニュピュレータの可動範囲（変数でいうと、θ_1,θ_2）は、通常は制限がある場合がほとんどです。またその範囲も、最大で $\leq 2\pi$ です。一方、動作速度（変数でいうと、$\dot{\theta}_1,\dot{\theta}_2$）は、なるべく速く動かしたいのが普通であり、通常はその上限を設ける必要はありません（実際上はある）。また、正確にゆっくりと動かす必要もあり、その範囲を極めて広いと考えられます。

以上のことより、角度 θ をパラメータにして、FSMで角速度 $\dot{\theta}$ を変数としたマップデータを切り替える手法が、メモリサイズを抑制でき、リアルタイム実行可能な実用的な非線形コントローラ実現するのに有効と考えられます。これを

ブロック線図で表すと、図9のようになります。

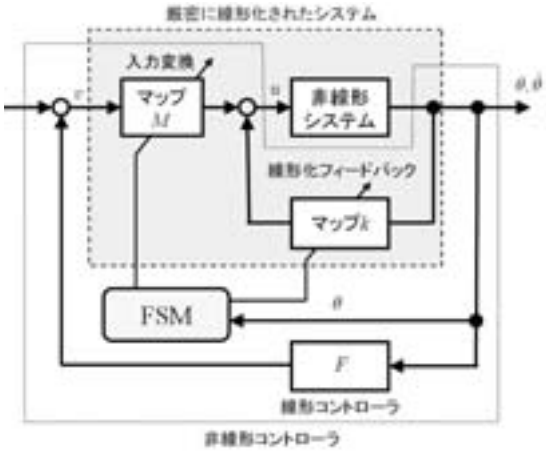

図9　FSMを用いた非線形制御

参考までに図8のシステムに対して実際に M, k を計算してみると、(15) 式のようになり、非常に複雑な非線形性を持つことが分かります。

5.状態遷移図を使えば

4章で述べた制御系の動作を、簡単化して状態遷移図で表してみましょう。まず、図10のような2リンクマニュピュレータを考えます。

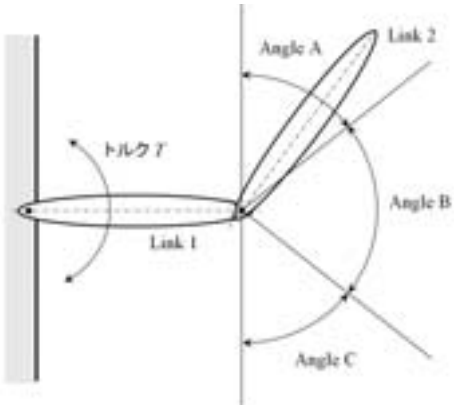

図10　2リンクマニュピュレータ

ここでは、Link 2の角度によって制御入力 T に対するコントローラの非線形特性を変えることとします。Link 2の角度がAngle AおよびAngle Cでは、Link 1にとっての動特性は同じと考えて同一のコントローラ（Map 1）とし、Angle Bでは別のコントローラ（Map 2）を用いることとします。これを状態遷移図で書くと図11のようになります。同様に状態遷移表で書くと図12のようになります。状態遷移表により、いくつかの未定義部分があることが明確に分か

$$
\begin{bmatrix}
> M \\
\begin{bmatrix}
2 + l1^2\, m2 + \frac{1}{4}\, l1^2\, m1 + l1\, m2\, l2 \cos\left(\theta_2(t)\right) + \frac{1}{4}\, l2^2\, m2 & 1 + \frac{1}{2}\, l1\, m2\, l2 \cos\left(\theta_2(t)\right) + \frac{1}{4}\, l2^2\, m2 \\
1 + \frac{1}{2}\, l1\, m2\, l2 \cos\left(\theta_2(t)\right) + \frac{1}{4}\, l2^2\, m2 & 1 + \frac{1}{4}\, l2^2\, m2
\end{bmatrix}
\end{bmatrix}
$$

$$
\begin{bmatrix}
> k[1] \\
-\frac{1}{2} \cos\left(\theta_1(t)\right) l1\, m1\, G + eM\_R1(t) + \frac{1}{2}\, l1\, m2\, l2 \left(\frac{\mathrm{d}}{\mathrm{d}t}\, \theta_2(t)\right)^2 \sin\left(\theta_2(t)\right) - \cos\left(\theta_1(t)\right) l1\, m2\, G \\
\quad + \frac{1}{2}\, l2 \sin\left(\theta_1(t)\right) \sin\left(\theta_2(t)\right) m2\, G + l1\, m2\, l2 \left(\frac{\mathrm{d}}{\mathrm{d}t}\, \theta_1(t)\right) \left(\frac{\mathrm{d}}{\mathrm{d}t}\, \theta_2(t)\right) \sin\left(\theta_2(t)\right) \\
\quad - \frac{1}{2}\, l2 \cos\left(\theta_1(t)\right) \cos\left(\theta_2(t)\right) m2\, G
\end{bmatrix}
\qquad (15)
$$

$$
\begin{bmatrix}
> k[2] \\
-\frac{1}{2}\, l2 \cos\left(\theta_1(t)\right) \cos\left(\theta_2(t)\right) m2\, G + \frac{1}{2}\, l2 \sin\left(\theta_1(t)\right) \sin\left(\theta_2(t)\right) m2\, G \\
\quad - \frac{1}{2}\, l2 \sin\left(\theta_2(t)\right) m2 \left(\frac{\mathrm{d}}{\mathrm{d}t}\, \theta_1(t)\right)^2 l1 + eM\_R2(t)
\end{bmatrix}
$$

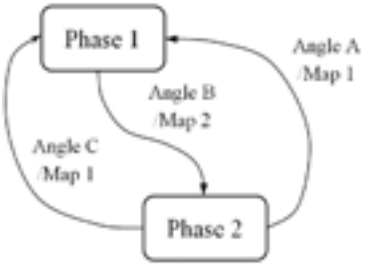

図11　制御ロジックの状態遷移図

	S	Phase 1	Phase 2
E		1	2
Angle A	1	Undefined	1 Map 1
Angle B	2	2 Map 2	Undefined
Angle C	3	Undefined	1 Map 1

図12　制御ロジックの状態遷移表

ります。

例えばPhase　1の状態にあるときに事象Angle Aが起きた場合の遷移先やアクションが分かりません。そんなことが起きるのか？考える必要があるのか？図10をもう一度見ると、Phase 1ではすでにAngle Aの領域に入っているので、事象Angle Aが起きることは論理的にありえない、だから考える必要が無いんだ。果たしてそれは正しいでしょうか？確かに制御ロジック的には合っています。しかし、仮にそのようなことがあるとしたらどんな場合でしょうか？センサーの故障、配線の断線、電気的ノイズの発生、プログラムの暴走、バグ等、色々なことが考えられます。

いずれにせよ、フォールトトレランスの立場からすれば、システムに某かの異常が発生している可能性が高い訳ですから、モータを急停止するなどの非常モードに遷移するのが望ましいと言えるでしょう。特にロボットアームのような機械的エネルギを取り扱う制御システムでは非常に重要です。いくらロバスト制御を使っても、いくら難解な非線形制御を使っても、センサーの異常を克服することはできません。

状態遷移表を用いれば、「非常モードに遷移する」のか、あるいは「無視」して良いのかが明確に定義できます。

最近の制御は高度にシステム化され、使用されるセンサーの数も膨大です。昔からセンサーの数（制御理論の世界では観測量という）を減らすというのは制御屋の大きな研究対象でした。それは観測できない、あるいは困難な内部状態を知りたいというのが研究のモチベーションです。しかるにコストの問題は置いておき、センサーを使うことができるのなら、その方がずっと性能を上げることができますし、制御系設計も楽になります。センサーに故障はつきものですが、故障しても成り立つ制御系を考えるのは非常に重要であると言えます。

高度な制御理論を生かすのも、システムとしての信頼性を向上させるにも、にも、状態遷移表による「ヌケ」、「モレ」チェックは非常に重要と考えます。

6.おわりに

制御理論は簡単ではありません。比較的易しいと言われている線形制御理論でも、理解にそれなりの時間を要します。ロバスト制御となるとさらに難しくなり、非線形制御となると、高度な抽象的数学を駆使しなくてはならず、扱える人が限られるのも事実でしょう。

誤解を恐れずに言うのであれば、私は、いわゆる制御理論家は、その数学的難易度やエレガントさが研究のモチベーションとなって、もちろんそれ自身はいいのですが、ややもすると現実の世界を見失って、無駄とまでは言いませんが、過剰であることに気付かないでいるのでは無いか？という気がいたします。なぜ、そこまでの操作範囲を考えるのか？なぜ、そこまで色んなケースを考えるのか？その為に難しくなりすぎて、解けるものも解けなくなってないか？答えは一般化しないと理論として成り立たないからですが、設計の側面からすれば、機構設計で、例えば不安定領域に入らないようにストッパなどをつけるとか、ソフトウェア的に回避するなど、色々な方法が考えられると思います。制御理論とFSMの融合が、より良い制御システムを実現することを願っております。