初心者向けテキスト：制御

MATLAB を利用した制御系設計

--- 発 展 編 ---

発　行：2001 年 8 月（1997 年作成）
著　者：加納　学
京都大学大学院工学研究科化学工学専攻
連絡先：
E-mail: kano@cheme.kyoto-u.ac.jp
http://www-pse.cheme.kyoto-u.ac.jp/~kano/

PID 制御系の設計

本テキストでは、

１．プロセスの同定（ステップ応答データからのモデリング）

２．PID 制御系の設計（IMC 法の利用）

３．リセットワインドアップ対策

をとりあげます。以下の文章を読み、空欄を埋め、指示に従って作業を進めて下さい。

１．プロセスの同定

制御対象となるプロセスのステップ応答データを図１に示す。これと同じデータが、”stepdata.mat”というファイルに保存されている。このステップ応答データを用いて、プロセスの伝達関数を求めたい。

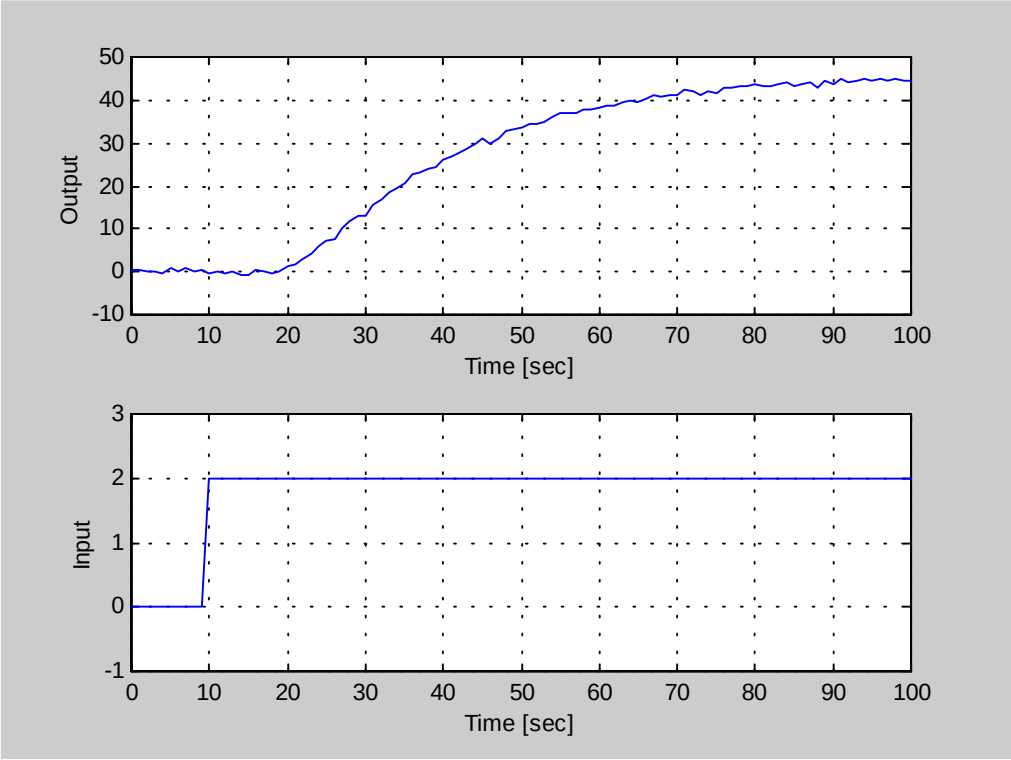

図１　ステップ応答データ

伝達関数モデルを構築するためには、まず、伝達関数の構造を決定する必要がある。プロセス制御で一般的に利用される伝達関数としては、

Ａ．一次遅れ＋むだ時間系

Ｂ．二次遅れ＋むだ時間系

などが挙げられる。実際のプロセスは非常に複雑であると考えられるが、制御系を設計するためには、上記のような簡単な伝達関数モデルで十分な場合が多い。図１のステップ応答データから、対象プロセ

スを表現するための伝達関数の構造としては、「二次遅れ＋むだ時間系」が適切であると判断される。そこで、次の伝達関数を採用する。

$$P(s)=\frac{K}{(Ts+1)^2}e^{-Ls}$$

次に、伝達関数に含まれているパラメータ（K,T,L）を決定しなければならない。まず、ステップ変化前後の定常値より、定常ゲイン K を容易に求めることができる。

K=＿＿＿＿＿＿＿＿＿＿

さらに、入力をステップ状に変化させてから出力が変化しはじめるまでの時間から、むだ時間 L を求めることができる。

L=＿＿＿＿＿＿＿＿＿＿

最後に、応答の速さを表すパラメータ T を決定しなければならない。ここでは、試行錯誤により T を決定することにする。そのために、以下のプログラムを作成し、”ident.m”というファイル名で保存する。

番号		＜説明＞
1.	clear	変数のクリア
2.	load stepdata	データの読み込み
3.	K=＿＿＿＿ ;	定常ゲイン K の入力
4.	L=＿＿＿＿ ;	むだ時間 L の入力
5.	T=＿＿＿＿ ;	T の入力（試行錯誤）
6.	time=0:100;	
7.	[ym,x,t]=step(K,[T^2 2*T 1],time);	ステップ応答の計算
8.	ym=[zeros(L,1); ym];	むだ時間の付加
9.	yp=y(11:101)/2;	
10.	figure(1)	
11.	plot([ym(1:91,1) yp])	ステップ応答の描画

パラメータ T を与えて、このプログラムを実行することにより、試行錯誤的に最適な T を求めることができる。なお、この段階で、定常ゲイン K およびむだ時間 L も再調整する。

T=＿＿＿＿＿＿＿＿＿＿

以上より、対象とするプロセスの伝達関数は

となる。

２．PID 制御系の設計

構築した伝達関数モデルに基づいて、PID 制御系を設計する。PID コントローラの設計方法は数多くあるが、ここでは IMC 法を用いることにする。

IMC はモデルをコントローラの内部に持つため、むだ時間を補償できるという特徴を持つ。しかし、ここでは PID 制御系の設計を目的としているため、むだ時間を無視して、すなわち、プロセスの伝達関数を

$$P_0(s) = \frac{K}{(Ts+1)^2}$$

とみなして、コントローラを設計する。IMC フィルタを

$$F(s) = \frac{1}{\lambda s+1}$$

とすると、フィードバックコントローラは、

$$C(s) = \frac{F(s)P_0^{-1}(s)}{1-F(s)P_0^{-1}(s)P_0(s)}$$

で与えられる。この式を実際に計算すると、

$$C(s) = \frac{T^2s^2+2Ts+1}{K\lambda s}$$

となる。このコントローラは

$$C(s) = K_P\left(1+\frac{1}{T_I s}+T_D s\right)$$

という PID コントローラと同一の構造をしており、以下の関係式が成り立つ。

K_P=_____

T_I=_____

T_D=_____

PID 制御パラメータ中の K, T は既に求められているので、この PID コントローラはチューニングパラメータとして IMC フィルタの時定数λのみを有する。すなわち、時定数λを調節して、希望する制御性能を実現しなければならない。

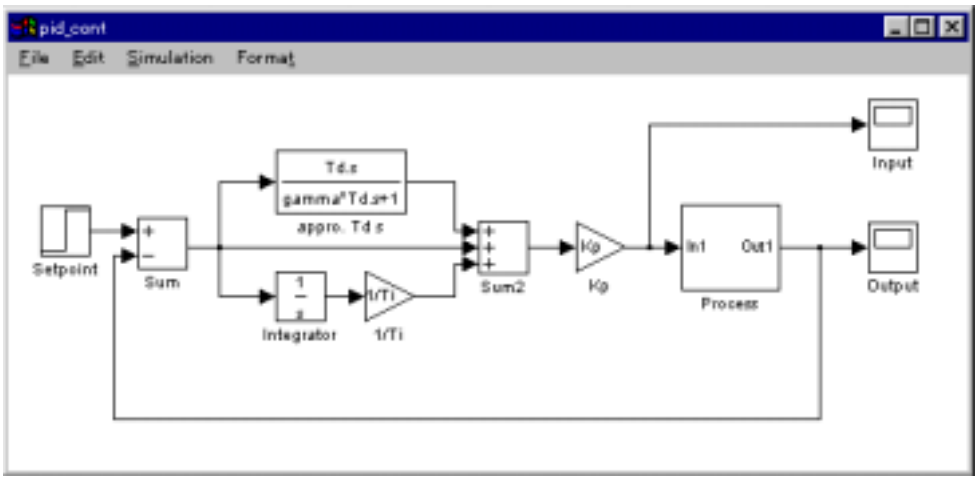

図２　PID 制御系

SIMULINK を用いて、試行錯誤によりチューニングを行うことにする。SIMULINK 上に構築したブロック線図を図２に示す。このプログラムは、”pid_cont.mdl”というファイル名で既に用意されている。このブロック線図において、”Process”は対象プロセスを表しており、既に定義されている。また、微分制御には近似微分を利用している。このプログラムを実行するためには、PID 制御パラメータ（Kp, Ti, Td）および近似微分のパラメータ（gamma）を与えておく必要がある。そのために、以下のプログラムを作成し、”set_pid.m”というファイル名で保存する。

番号		＜説明＞
1.	clear	
2.	K=_____ ;	
3.	T=_____ ;	
4.	L=_____ ;	
5.	lambda = _____	調整パラメータ
6.	Kp = 2*T/K/lambda	
7.	Ti = 2*T	
8.	Td = T/2	
9.	gamma=0.1	近似微分のパラメータ

先に求めた伝達関数モデルに従って K, T, L を入力し、さらに lambda を与えることにより、PID 制御パラメータが決定される。このプログラムを実行したのち、SIMULINK のプログラムを実行することにより、制御シミュレーションを行うことができる。そこで、制御シミュレーションの結果を見ながら、試行錯誤により、最適なパラメータ lambda を決定する。

３．リセットワインドアップ対策

＜リセットワインドアップとは＞

現実の操作量には上下限制約が課せられることが多い。例えば、弁の全開と全閉などである。偏差の積分量がこの上下限制約を越えたときに生じるのがリセットワインドアップと呼ばれる問題である。

簡単のために、開ループで積分動作だけを取り出し、入力である偏差が図３(a)のように変化する場合を考える。上下限制約がなければ、操作量は OABCD と推移する。当然ながら、偏差の積分量が増加するときには、それに比例して操作量も増加し、偏差の積分量が減少するときには、それに比例して操作量も減少するという、積分動作の本来の機能が果たされている。操作量が図の L という値で飽和する場合、積分器の出力は依然として OABCD と変化するが、実際の操作量は OAECD と推移することになる。このとき、AE 間では、偏差の積分量の増加に比例して操作量も増加するという機能が失われているが、それは制約のためやむを得ない。一方、EC 間では、偏差の積分量が減少するにもかかわらず、操作量は減少しない。それは、積分器出力が AB と巻き上げ（ワインドアップ）られているため、BC と巻き戻されるまでは、偏差の積分量の減少に比例して操作量も減少するという機能が回復しないためである。

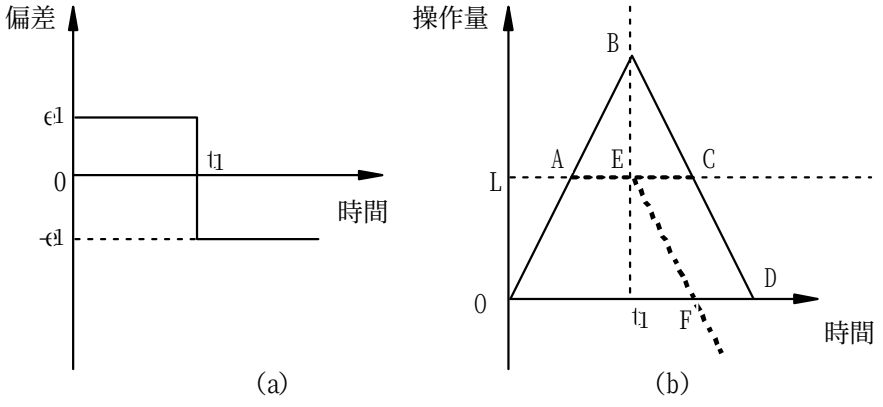

図３　リセットワインドアップ

開ループの場合について説明したが、フィードバック制御系に組み込まれた場合も同様で、巻き戻しの期間だけ本来の機能の回復が遅れ、そのために偏差の整定が遅れ、オーバーシュートも大きくなる傾向がある。

リセットワインドアップ対策として、操作量が上下限制約値に達した時点でそれを越える方向の積分機能を停止する方法が広く用いられている。もちろん、制約条件内に戻る方向の積分は継続する。図３

でいえば、A で積分が停止され、積分器出力は操作量と同じく AE と推移する。偏差の減少が始まる E で本来の機能を回復して、その後、操作量は EF という軌跡をたどることになる。

リセットワインドアップ対策は、目標値や負荷の大きい変動に対して必要となるほか、常時偏差が持続され、大きい負荷の変動時にのみ制御動作が期待される機器、例えば圧力が高くなりすぎるのを防止するバイパス弁などの制御にも有効である。

＜以上＞

それでは、リセットワインドアップと呼ばれる問題を実感するために、制約条件付きのシミュレーションを行う。そのために、先に利用した SIMULINK 上のブロック線図を図４のように書き換える。元のブロック線図との違いは、入力に上下限制約を付け加えた点である。なお、上下限値は 1, -1 としておく。先に決定した制御パラメータを用いて、制約条件下での制御シミュレーションを行う。そうすると、オーバーシュートが大きくなることが確認できる。

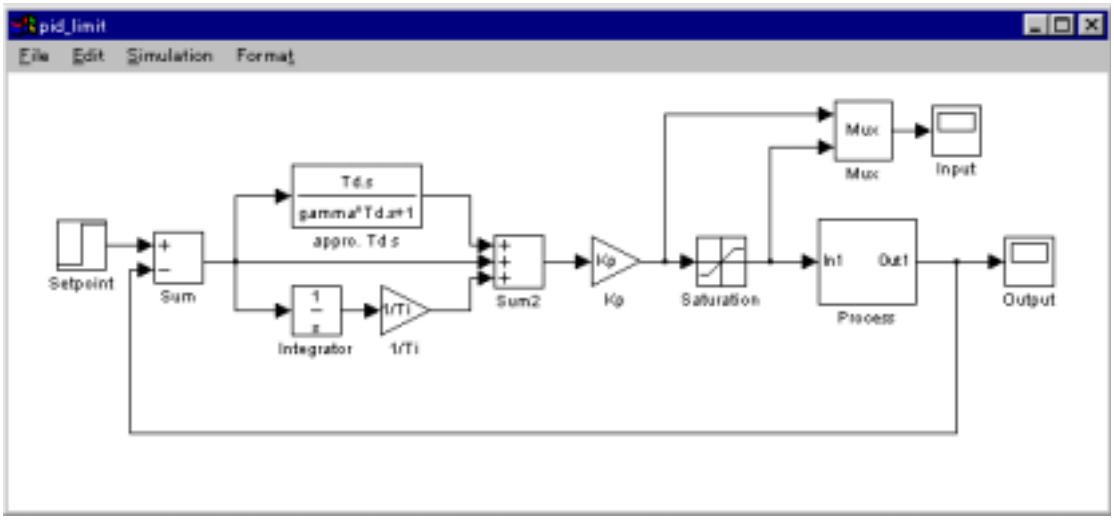

図４　制約条件付き PID 制御系

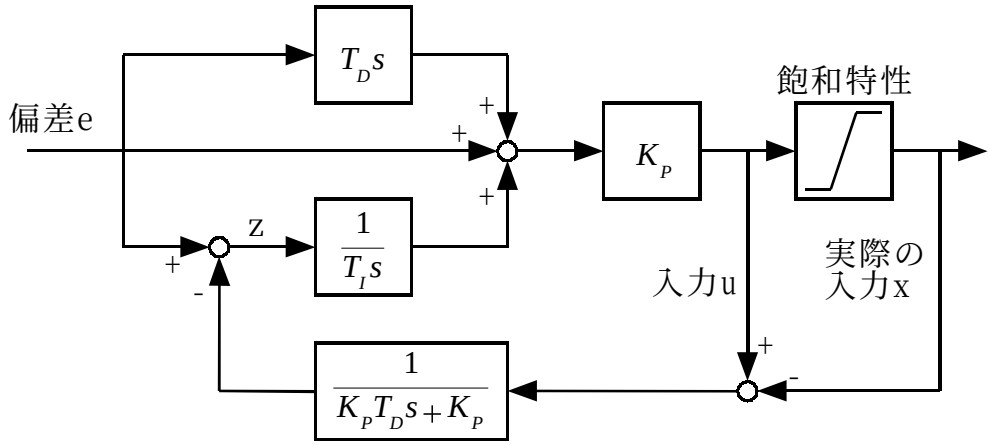

図５　自動整合 PID 制御

リセットワインドアップ対策として、自動整合制御系が提案されている。自動整合 PID コントローラのブロック線図を図５に示す。このコントローラは、入力が上下限制約の範囲内にあるとき、すなわち u=x であるときには、通常の PID 制御となる。一方、入力が飽和している場合には、以下の関係式が成り立つ。

$$z = e - \frac{1}{K_P T_D s + K_P}(u - x)$$

$$u = K_P(1 + T_D s)e + \frac{K_P}{T_I s} z$$

この２式から u を消去すると、

$$z = \frac{T_I s}{(T_D T_I s^2 + T_I s + 1)K_P} x$$

となることから、最終値定理を用いて、

$$\begin{aligned} \lim_{t\to\infty} z(t) &= \lim_{s\to 0} sz(s) \\ &= \lim_{t\to\infty} s \frac{T_I s}{(T_D T_I s^2 + T_I s + 1)K_P} \frac{x_{\text{limit}}}{s} \\ &= 0 \end{aligned}$$

となることがわかる。ここで、x_{limit} は入力の上限値あるいは下限値である。この式は、積分器への入力信号がゼロに漸近することを意味している。すなわち、入力が飽和している場合に、積分動作を切ることに対応している。

自動整合 PID 制御のシミュレーションを行うために、先に利用した SIMULINK 上のブロック線図を図６のように書き換える。先に決定した制御パラメータを用いて、制約条件下での制御シミュレーションを行う。そうすると、オーバーシュートを抑制できることが確認できる。

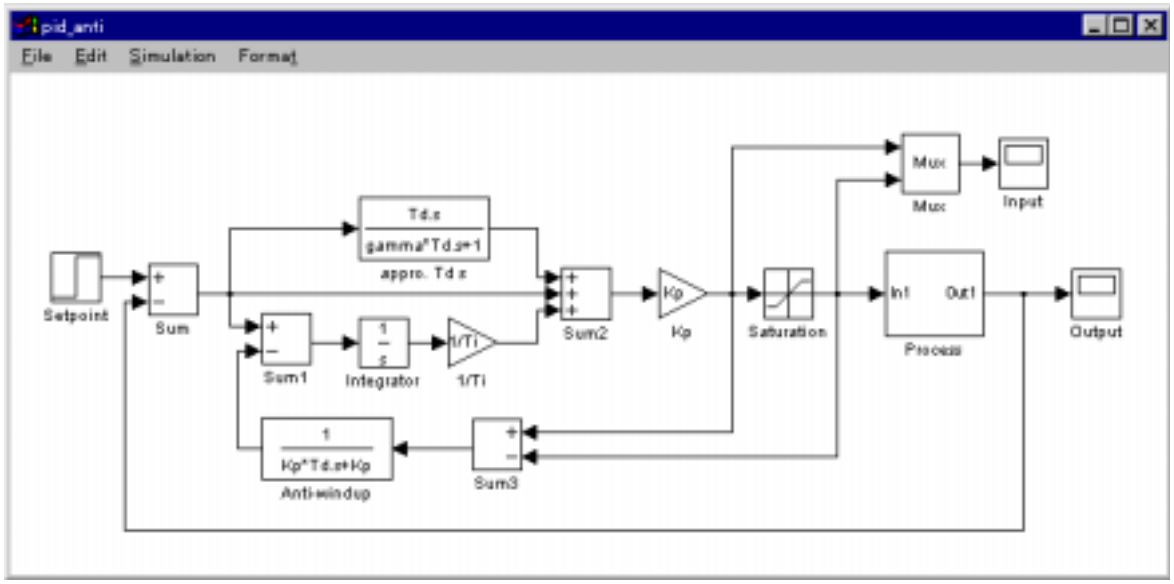

図６　自動整合 PID 制御系

おわり