Inverted Pendulum control with EPICS

リリース ***1.3***

Noboru Yamamoto

2019 年 **10** 月 **31** 日

目 次

概 要

EPICS での制御の一例として、倒立振り子装置の制御システムを作成した。倒立振り子の制御は状態推定器と最適レギュレータを組み合わせたフィードバックで行う。制御のために作成した EPICS レコードおよびフィードバックループの定数決定のために使われた Octave プログラムについても詳細に解説する。[1]

[1] Octave から Python/control module へ移行したので、python 版のプログラムについても A.4.1 節で説明する。

第1章　EPICSによる倒立振り子の制御

1.1 始めに

EPICS での制御の一例として、倒立振り子を制御するシステムを作成した。状態制御則にもとづくフィードバックを実現するために、新たに EPICS レコードを作成した。

このレコードと、時間割込みを使うことで、VME 計算機とリアルタイム OS を使ったシステムでは kHz 程度のフィードバックを実現することができた。実際には、状態フィードバックを使うことで、60 Ｈｚ程度の処理間隔でも十分な制御性能を持つ。この報告では、倒立振り子装置の概要、状態フィードバックの説明、状態フィードバック法によるフィードバックの設計、EPICS レコードへの実装、実際の装置の性能と設計値との比較について述べる。

1.2 倒立振り子装置

倒立振り子装置は、株式会社リンクスコーポレーション [A-2][A-3] が販売している倒立振り子装置を使用した。装置は図 1.1 に示した外観のように、倒立振り子が固定された台車と、それを一次元的に移動させるためのモータから構成されている。また、台車の位置および、倒立振り子の傾きを検出するための検出装置〔ポテンシオ・メータ〕が装着されている。角度検出用のポテンシオ・メータは磁気抵抗素子を使ったもので、接触子を持たない、360 度回転可能なタイプが使われている。これらの検出器と、モータの制御入力はドライバ・ボックスに接続されている。ドライバ・ボックスにはポテンシオ・メータのモニタの為の二つのアナログ出力とモータの速度指令値入力のためのアナログ入力端子が設けられている。

1.2.1 制御装置

現行 (2019 年 9 月 1 日現在) の倒立振り子の制御システムは、

1) モータドライブ基盤を含む制御ボックス。

2) CPU/ADC/DAC/高速 DIO モジュールを搭載した PLC システム

から構成されている。[1]

PLC の CPU は Linux ベースの CPU を採用し、この上で直接 EPICS[A-4] を動作させている。EPICS データベースから PLC の各種モジュールを利用するために必要となるデバイスサポートライブラリはこの PLC 環境に標準のものを使っている。また、倒立振り子の制御専用のレコードを開発し、制御アルゴリズムはこのレコードに実装した。

[1] 次の節にあるように当初は CAMAC の IO モジュールと VME 計算機を使って構築されていた。EPICS の上に構築されたアプリケーションは基本的に同じものが使われている。

CAMAC と VME 計算機を用いた制御装置 (Obsolete!)[2]

此処で構築したシステムでは、標準的な EPICS の構成である VxWorks の動作する VME シングルボード計算機 (Force PowerCore 6750 など) を用いた 。倒立振り子制御ボックスとのインターフェイスには CAMAC の DAC/ADC ボードを用いた。VME ボード計算機に設置された CAMAC Serial Highway Driver 経由で、VME 計算機上のフィードバックプログラムは台車位置および振り子の角度をよみとり、また速度指令値をドライバ・ボックスに出力する。

VME 計算機上には、 EPIC S のランタイム・データベースソフトウェアが稼働している。データベースには、CAMAC からの入出力をつかさどるレコード、CAMAC から読みだされた値を物理的な量へ変換を行うレコード、フィードバックを構成するレコードなどがロードされている。

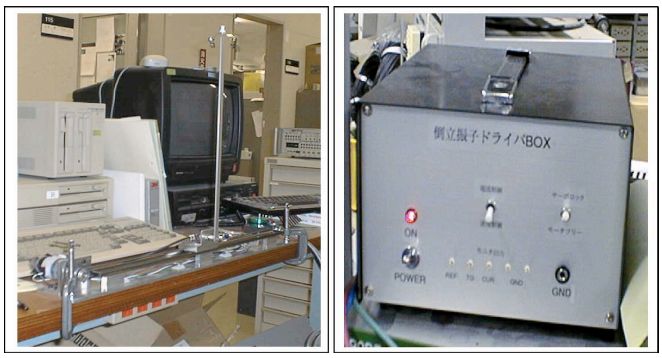

図. 1.1: 倒立振り子装置とドライバ・ボックスの外観: 左)　稼働中の倒立振り子装置 右) 倒立振り子のドライバ・ボックス

PLC を用いた制御装置

現行 (2017 年~) の倒立振子制御システムは、Yokogawa 製の PLC を使って構築されている。このシステムでは、PLC と言いながらも CPU としてラダープログラムを実行するシーケンサではなく、LINUX を OS として使用する CPU を用いている。入出力は PLC のモジュールを使っている。この上で、EPICS-IOC を構築し、EPICS-sequencer プログラムと組み合わせることで倒立振子制御システムとしている。

振子の倒立状態を維持するためのフィードバックは,　旧システムと同じく、EPIC record として実装されている (Feedback record)。

EPICS-sequencer は、台車可動域のチェック、倒立フィードバック停止後に台車を原点に移動させる FB, 振り上げ遷移の三つの state sets を実装してある。

1.3 倒立振り子の物理モデルと制御

ここでは、制御対象である倒立振り子の物理的なモデルについて説明します。この物理的モデルに基づいて線形化した制御モデルを作り、それに最適制御理論を適用する事で、実際の制御システムが構築されています。

[2] 初代の制御システムは CAMAC を使って構築された。今となっては歴史的資料としての意味しかないが、参考のために此処に残しておく。

図. 1.2: PLC システムの外観. DAC/ADC/高速入出力モジュールと Linux が動作する CPU などから構成されている。

1.3.1 モデルの運動方程式から制御モデルへ

制御対象の物理モデルは質量 M の台車と質量 m, 慣性モーメント I の棒として表現される。棒の一端は台車の中央に固定されている。固定端から棒の重心までの距離を L とする。振り子が鉛直方向となす角度を θ とした時、倒立位置 $(\theta = 0)$ からのずれが小さいとするとその運動方程式 Eq.(1.1) および Eq.(1.2) は、

$$(m + M)\,\ddot{z} - ml\ddot{\theta} = -\mu_z \dot{z} + F_{ext}(t) \tag{1.1}$$

$$\left(I + ml^2\right)\ddot{\theta} - ml\ddot{z} = +mgl\theta - \mu_\theta \dot{\theta} \tag{1.2}$$

である [付録 「第 A.1 章」参照]。此処で $F_{ext}(t)$ は台車駆動装置から受ける外力である。台車駆動装置は、速度指令値 $u(t)$ を受け取り、それが台車の速度 $\dot{z}$ となるような制御ループを持っている。理想的な駆動装置では $\dot{z} = u$ となるはずであるが、制御ループの時間遅れなどを考慮して（ [A-3][A-5] による)、

$$\ddot{z} = -\eta \dot{z} + \xi u \tag{1.3}$$

を採用することにする[3]。　η は時間遅れを表しており、平衡速度は $\dot{z} = \frac{\xi}{\eta} u$ となる。運動方程式は結局、

$$\ddot{z} = -\eta \dot{z} + \xi u \tag{1.4}$$

および

$$\left(I + ml^2\right)\ddot{\theta} = mgl\theta - \mu_\theta \dot{\theta} + ml\left(-\eta \dot{z} + \xi u\right) \tag{1.5}$$

となる。これを行列形式で書き直すと、

$$\frac{d}{dt}\begin{pmatrix} z \\ \theta \\ \dot{z} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} 0, & 0, & 1, & 0 \\ 0, & 0, & 0, & 1 \\ 0, & 0, & -\eta, & 0 \\ 0, & \frac{mgl}{(I+ml^2)}, & \frac{-\eta ml}{(I+ml^2)}, & \frac{-\mu_\theta}{(I+ml^2)} \end{pmatrix} \begin{pmatrix} z \\ \theta \\ \dot{z} \\ \dot{\theta} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \xi \\ \frac{ml\xi}{(I+ml^2)} \end{pmatrix} u \tag{1.6}$$

この装置では、台車の位置と、振り子の角度が観測量 $y = \left(z, \theta\right)$ である。行列形式では、

$$y = \left(z, \theta\right) = \begin{pmatrix} 1, 0, 0, 0 \\ 0, 1, 0, 0 \end{pmatrix} \begin{pmatrix} z \\ \theta \\ \dot{z} \\ \dot{\theta} \end{pmatrix} \tag{1.7}$$

[3] 結局まじめに取り扱おうとすると速度指定値 u と駆動装置の FB ループ、台車からの反動を考慮したモータの回転の運動方程式までをふくんだ系を考えることになってしまう。 駆動用のモーターには、回転速度検出用の TG(Trubo Generator) が入っており、ドライボードは Reference 信号 (速度指令値）と TG の信号の差分でフィードバックをかけているようなので、モーターの動作としては、$\ddot{z} = \xi\left(u - \frac{\eta}{\xi}\dot{z}\right)$ で記述されると考えても良いだろう (?)。

となる。

1.4 フィードバックの設計

上記で得た制御対象のモデルに対して、現代制御理論に基づいてフィードバックを設計する。フィードバックは状態推定器つき状態フィードバックにより構成する。この方法では、状態推定器と状態フィードバックの二つのループのフィードバックパラメータを決定する必要があるが、このパラメータは最適レギュレータの理論を用いて**決定**される。

1.4.1 最適レギュレータ

フィードバックとしては、状態推定器付き最適レギュレータによる状態フィードバック法を用いる。状態フィードバック法は、制御対象の状態変数 X と出力値 (観測値)Y が行列形式の線型な状態方程式、

$$\frac{dX}{dt} = AX + Bu \tag{1.8}$$

$$Y = CX + Du \tag{1.9}$$

に従うとき、制御入力 (操作量)u を、

$$u = -KX \tag{1.10}$$

によって決定し、制御する方式である。この制御則の下では、制御対象の状態方程式は、

$$\frac{dX}{dt} = (A - BK)X \tag{1.11}$$

$$Y = (C - DK)X \tag{1.12}$$

となる。 $A - BK$ の固有値の実部がすべて負であれば、明らかに、この系は $t \to \infty$ で $x \to 0$ である。行列 A, B が可制御の条件[4]をみたすなら、このような K が存在することが証明されている。倒立振り子の場合には、Free な数値解析用プログラム Octave[付録「第 A.3 章」参照] を用いて、この系が可制御であることが次のようにして示される。

```
J=3.20e-4;
g=9.8;
m = 0.023000;
l = 0.20000;
mu =27.41e-6;
zeta=240;
xsi=90;
A=[0,0,1,0;0,0,0,1;0,0,-zeta,0;
   0,m*g*l/(J+m*l**2),-zeta*m*l/(J+m*l**2),-mu/(J+m*l**2)];
B=[0;0;xsi;xsi*m*l/(J+m*l**2)];
C=[1 0 0 0;010 0];
D=[ 0; 0 ];
rank(A)
rank(ctrb(A,B))
```

[4] 行列 A および B について $rank\left[B, AB, A^2B, \ldots, A^{n-1}B\right] = rank(A)$ が成り立つ時、このシステムは**可制御**であると云われる。

を実行すると、`rank(ctrb(A,B))` が状態空間の次元の 4 であることが示される。これは, この系が可制御であることを示している。

A, B を安定化させるフィードバックゲイン K を求める方法は極配置法などの方法が知られているが、ここでは最適レギュレータ法を用いてフィードバックゲイン K を求めてみる。最適レギュレータ法は、次の (目標）積分:

$$I = \int_{t_0}^{t_f} dt \left\{ x^T Q x + u^T R u \right\} \tag{1.13}$$

が最小となるようにフィードバックゲイン K を求める方法である。 Q および R は正定値の対称行列で、目標積分の重みとなっている。 Q および R を選び直すことで、最終的なフィードバックループの振る舞いをコントロールする。K は、代数的リカッチ方程式 (Riccati algebraic equation)、

$$PA + A^T P - PBR^{-1}B^T P + Q = 0 \tag{1.14}$$

の正定対称解 P を用いて、

$$K = R^{-1}B^T P \tag{1.15}$$

と求めるられる。 Octave では (Matlab とおなじく `lqr()` 関数を使うことで、フィードバックゲイン K を求めることができる。

```
R=1;
Q2=diag([50,1,1,1]);
[Kr2,p,e]=lqr(A,B,Q2,R);
eig(A-B*Kr2);
```

このプログラムの最後では閉ループの固有値を求めているが、この計算結果は、

```
ans =
-420.9663 + 0.0000i
-4.1659 + 2.3125i
-4.1659 -2.3125i
-2.4208 + 0.0000i
```

となって、すべての固有値の実部が負となっている。このことから、最適レギュレータ法による状態フィードバックは漸近安定であることがわかる。

1.4.2 状態フィードバックの安定性

状態フィードバックの漸近安定性をつぎのように示すこともできる。次の量を考える。

$$I = x^T P x \geq 0 \tag{1.16}$$

P は Ricatti 方程式の正定対称解であるから、この量も正定値である。さらにこの量の時間微分をとると、

$$\frac{dI}{dt} = x^T \left\{ PA + A^T P - PBK - K^T BP \right\} x \tag{1.17}$$

$$\begin{aligned} \frac{dI}{dt} &= -x^T \left\{ Q + PBR^{-1}B^T P \right\} x \\ &= -x^T Q x - u^T R u <= \end{aligned} \tag{1.18}$$

となって、Q,R が正定値の行列であることから、この量が $t \to \infty$ で有界、したがって状態フィードバック系は安定であることが示された。

1.4.3 状態推定器

状態フィードバックを行うためには各時刻での状態ベクトル x のすべての成分を知る必要がある。一般には、系の出力 y は状態ベクトル x をすべて再現出来るとは限らない。しかしながら、系 (A, C) が可観測 [A-5] であれば 時系列データ $(u(0), y(0)), (u(1), y(1)), \ldots (u(n), y(n))$ を用いて、時刻 n における状態ベクトル x を推定することができる。このような状態推定器は、

$$\frac{d\hat{x}}{dt} = A\hat{x} + Bu + H\left(y - C\hat{x} - Du\right) \tag{1.19}$$

として構成される。今、$\hat{e} \equiv \hat{x} - x$ と定義すると、$\hat{e}$ に関する運動方程式は、

$$\frac{d\hat{e}}{dt} = \left(A - HC\right)\hat{e} \tag{1.20}$$

となる。これから直ちに、$(A - HC)$ の固有値がすべて左半面にあれば、$t \to \infty$ で $\hat{e} \to 0$ となることがわかる。最適レギュレータの場合と同じように、系 (A, C) が可観測であれば、このような が存在することが保証される。安定な状態推定器を与える状態推定器のゲイン H は、極配置法や、Kalman Filter による方法で決められる。カルマンフィルタでは、状態推定器のゲイン H は、再びリカッチ方程式、

$$A\bar{X} + \bar{X}A^T - \bar{X}C^T V^{-1} C\bar{X} + GWG^T = 0 \tag{1.21}$$

の正定対称解 $\bar{X}$ を用いて

$$H = \bar{X}C^T V^{-1} \tag{1.22}$$

と求められる。上記のリカッチ方程式で、V および GWG^T は夫々出力値 (観測値) と入力値に含まれる雑音の共分散行列である。

それでは、雑音の無い系ではカルマンフィルタを用いることは出来ないのであろうか?いま、$V = \lambda\bar{V}, W = \lambda\bar{W}, \bar{X} = \lambda\bar{\bar{X}}$ とすると、$\lambda \to 0$ の極限でも、カルマンフィルタのフィードバックゲインは有限な値で存在して、

$$H = \bar{\bar{X}}C^T \bar{V}^{-1} \tag{1.23}$$

となる。ただし、$\bar{\bar{X}}$ は

$$A\bar{\bar{X}} + \bar{\bar{X}}A^T - \bar{\bar{X}}C^T \bar{V}^{-1} C\bar{\bar{X}} + G\bar{W}G^T = 0 \tag{1.24}$$

の正定対称な解である。結局、$\bar{V}, G\bar{W}G^T$ は、最適レギュレータの場合と同じ様に、最適化の重みを与えていると考えることが出来る。 Octave ではカルマンフィルターのゲインを、lqr 関数を用いて求めることができる[5]。

```
G=eye(4);
sigw=eye(4);
sigv=eye(2);
[H,p,e]=lqr(transpose(A),transpose(C),G*sigw*transpose(G),sigv);
H=transpose(H)
```

eye() は単位行列を返す関数である。また、*lqr* の戻り値の p はリカッチ方程式の解、e は $A - HC$ の固有値の配列である[6]。

[5] Octave 2 ではこのために lqe 関数が使えたが、SICOT ベースの Octave3 では lqe 関数は提供されていない。-> GNU Octave 4.4.1 で確認したところ、`lqe()` 関数は利用可能でした。

[6] 可観測性は可制御性の双対であって、(A^T, C^T) が可制御であることと等価である。

1.5 状態推定器を用いたフィードバック

状態推定器をもちいて得られた状態ベクトル x をもちいて、最適レギュレータを構成することが出来る。このレギュレータの状態方程式は、

$$\frac{dx}{dt} = Ax - BK\hat{x} + Bu_i \tag{1.25}$$

$$\frac{d\hat{x}}{dt} = (A - BK - HC + HDK)\,\hat{x} + Hy + Bu_i \tag{1.26}$$

$$y = Cx - DK\hat{x} + Du_i \tag{1.27}$$

$$u = -K\hat{x} + u_i \tag{1.28}$$

と書き表される。 x および x に関する状態方程式の部分を、行列形式で書くと、

$$\frac{d}{dt}\begin{pmatrix} x \\ \hat{x} \end{pmatrix} = \begin{pmatrix} A, & -BK \\ HC, & A - BK - HC \end{pmatrix}\begin{pmatrix} x \\ \hat{x} \end{pmatrix} + \begin{pmatrix} B \\ B \end{pmatrix} u_i \tag{1.29}$$

となる。この拡大系の係数行列の固有値は、 $A - HC$ および $A - BK$ の固有値を合わせたものになる。すなわち、フィードバック系と状態推定器を組み合わせたことによっては、新しい固有値は出てこない。これを分離定理とよぶ。

$$\begin{aligned} &\det\begin{vmatrix} A - \lambda, & -BK \\ HC, & A - BK - HC - \lambda \end{vmatrix} \\ &= \det\begin{vmatrix} A - HC - \lambda, & -A + HC + \lambda \\ HC, & A - BK - HC - \lambda \end{vmatrix} \\ &= \det\begin{vmatrix} A - HC - \lambda, & 0 \\ HC, & A - BK - \lambda \end{vmatrix} \\ &= \det|A - HC - \lambda|\,\det|A - BK - \lambda| \end{aligned} \tag{1.30}$$

先程導入した状態推定器の誤差 e を用いて、上記の状態方程式を書き直すと、

$$\begin{aligned} \frac{dx}{dt} &= (A - BK)\,x - BK\hat{e} + Bu_i \\ \frac{d\hat{e}}{dt} &= (A - HC)\,\hat{e} \end{aligned} \tag{1.31}$$

となることからも、分離定理が成り立つことがわかる。 $\hat{e}$ の運動は $A - HC$ の固有値できまり、 x の運動は $A - BK$ の固有値できまる。さらに状態推定器の誤差 e は x の運動にたいして、外力として作用することがわかる。

1.5.1 Bode 線図

このようにして設計したフィードバックシステムの周波数応答を見るために、 bode 線図を書いてみる。次の Octave program

```
AF=[[A -B*Kr];
[H*C, A-B*Kr-H*C]];
BF=[B;B];
CF=[C [0,0,0,0; 0,0,0,0]];
DF=[0;0];
sys=ss2sys(AF,BF,CF,DF)
bode(sys,[0.1:1000],1,1)
bode(sys,[0.1:1000],2,1)
```

によって閉ループ系の Bode 線図を描いたものが、図. 1.3 である。この系は 1 入力 2 出力系であるので、ここに示したように、二つの bode 線図が書かれる。

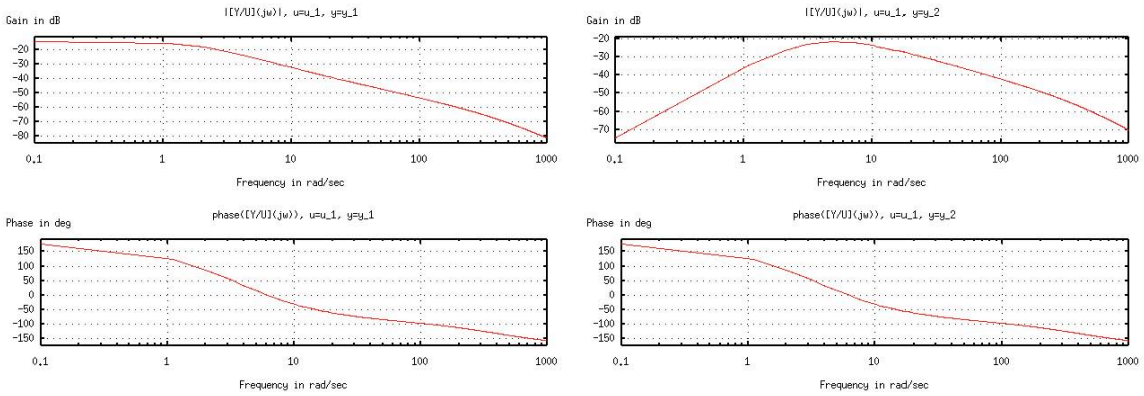

図. 1.3: 倒立振り子系の Bode 線図。どちらも入力：台車速度に対する　a) 出力：台車位置 , b) 出力：振り子角度のゲイン（上図）と位相（下図）の周波数依存性を示している。

1.5.2 制御システムのシミュレーション

次に閉ループ系の応答をみる。 Octave にはインパルス入力とステップ入力に対する応答をプロットする関数、impluse() および step() が用意されている。

```
sys=ss2sys(AF,BF,CF,DF)
step(sys,1,10,400)
impluse(sys,1,10,400)
```

これによって描かれた系の応答が、図 1.4 である。

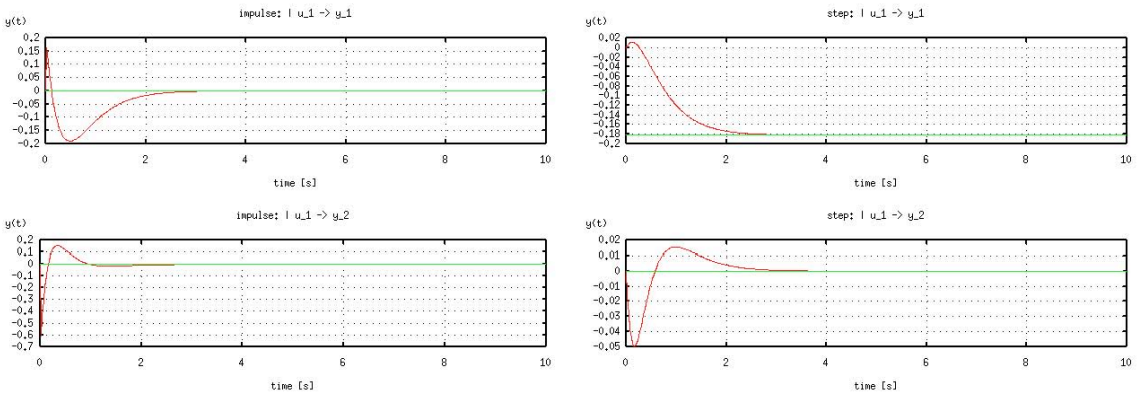

図. 1.4: 閉ループ系のインパルス入力 (a) およびステップ入力)(b) による応答。上図：台車位置 (m)、下図：振り子の振れ角 (rad). 横軸：時間（秒）a) インパルス入力応答。 b) ステップ入力応答。

1.5.3 離散化

ここまでの解析は連続系の状態方程式としての解析であった。このフィードバック則を EPICS などの計算機ベースの制御システムに実際に組み込むためには、時間方向に関しての離散化が必要である。ここでは、Octave の c2d() 関数を用いて、離散化を行った。

```
sysd=c2d(ss2sys(A-B*Kr3-H*C,H,-Kr3),1./60);
Ad=sysd.a;
Bd=sysd.b;
Cd=sysd.c;
Dd=sysd.d;
```

Ad, Bd, Cd, Dd はゼロ次ホールド法によって離散化された系の状態方程式の係数行列である。 c2d の最後の引数は時間の離散化の単位時間 Ts である。ここでは 60Hz で制御を行うこととし 1/60. を指定した。

1.5.4 測定誤差の影響

このフィードバック系で測定値に誤差があった場合の影響を考察する。測定誤差を δy とすると、観測値 y は、

$$y = Cx + Du + \delta y \tag{1.32}$$

となる。これから系の状態方程式は、

$$\frac{dx}{dt} = (A - BK)\, x - BK\hat{e} \tag{1.33}$$

$$\frac{d\hat{e}}{dt} = (A - HC)\, \hat{e} + H\delta y \tag{1.34}$$

と変更される。形式的にこの方程式の解は、

$$\hat{e}(t) = \int_{-\infty}^{t} e^{(A-HC)(t-t')} H\delta y dt' \tag{1.35}$$

$$x(t) = -\int_{-\infty}^{t} dt'' e^{(A-BK)(t-t'')} BK \int_{-\infty}^{t} e^{(A-HC)(t''-t')} H\delta y dt' \tag{1.36}$$

と書くことが出来る。今誤差が定数であるとすると、定常状態では、

$$0 = (A - BK)\, x - BK\hat{e} \tag{1.37}$$

$$0 = (A - HC)\, \hat{e} + H\delta y \tag{1.38}$$

を解くことによって、定常状態の状態ベクトルを求めることが出来る。

$$x_{st} = (A - BK)^{-1} BK\hat{e}_{st} \tag{1.39}$$

$$\hat{e}_{st} = -(A - HC)^{-1} H\delta y \tag{1.40}$$

係数行列 A の性質によって[7], $(A - BK)^{-1} BK$ は第一行を除いて総ての成分が 0 となる。したがって、δy に係わらず、定常状態で残る誤差の影響は台車位置のずれだけになる [A-5]。

[7] プログラムプログラム 1.1 を SageMath[A-6] で実行することで、$(A - BK)^{-1} BK$ は第一行だけがゼロ出ない成分を持つことが確認できる。

プログラム 1.1: Error propagation from y to state space.

```
for i in range(4):
  var('b_%d'%i)
  var('k_%d'%i)
  for j in range(4):
    var('a_%d%d'%(i,j))
A=matrix([
     [0, a_01, a_02, a_03],
     [0, a_11, a_12, a_13],
     [0, a_21, a_22, a_23],
     [0, a_31, a_32, a_33]]);
B=matrix([[b_0],[b_1],[b_2],[b_3]]);
K=matrix(((k_0, k_1, k_2, k_3)))
((A-B*K)**(-1)*B*K).simplify_full()
```

1.6 EPICS を用いた制御システムへの実装

以上に述べた制御則を EPICS にもとづく制御装置に組み込むために、専用のレコード InvPendRecord を作成した。制御プログラムの本体部分は参考文献 [A-5][A-7][A-8][A-9] を参考にした。制御プログラムなどに必要な係数は、 EPICS の SCAN の繰り返し時間毎に InvPendRecord に定数として書き込まれている。SCAN フィールドを変更すると適切な係数が選ばれる。実機では、SCAN の周波数を 5 Hz 以下にすると、系を安定化することが出来なかった。

InvPendRecord の process 関数中では、取り込んだ台車の位置と、振り子の角度から、状態推定器を更新し、それにもとづいて新たな出力指令値を計算している。

連続系では、

$$\begin{aligned}\dot{\hat{x}} &= (A - BK_r - HC)\,\hat{x} + Hy \\ &= (A - BK_r - HC)\,\hat{x} + Hy + (B - HD)u_i\end{aligned} \tag{1.41}$$

$$u_c = -K_r\hat{x} \tag{1.42}$$

であるが、離散系では:

$$\hat{x}(k+1) = \hat{A}_d\hat{x}(k) + H_d y(k) \tag{1.43}$$

$$u_c(k+1) = -KC_d\hat{x}(k+1) \tag{1.44}$$

となる。 VxWorks を用いる VME 計算機などで稼働する EPICS の periodic scan は VxWorks の Tick である 1/60 秒以下にすることは出来ない。これ以上の繰り返しでレコードのプロセスを行うために、 VxWorks では Auxially Clock による時間割込みを利用することが出来る。この仕組みを利用することで、 60Hz〜1KHz までのフィードバックループによる倒立振り子の制御も可能である。

1.6.1 倒立振り子装置の動作試験

実機での指令値のステップ変更に対する応答を次の図 (図-1.5) に示す。離散化されたフィードバックの繰り返し周波数はこの場合、60Hz である。倒立振り子を立てるだけならこの程度の繰り返しでも充分である[8]。実測値と、シミュレーションによる結果は定性的にはよくあっている。

[8] 振り上げなどの制御を行うためには、フィードバック制御の繰り返しは速い方が良いだろう。繰り返しの速度と共に繰り返し周期の精度も問題になる。VxWorks などのリアルタイム OS では繰り返し周期の精度が高い。

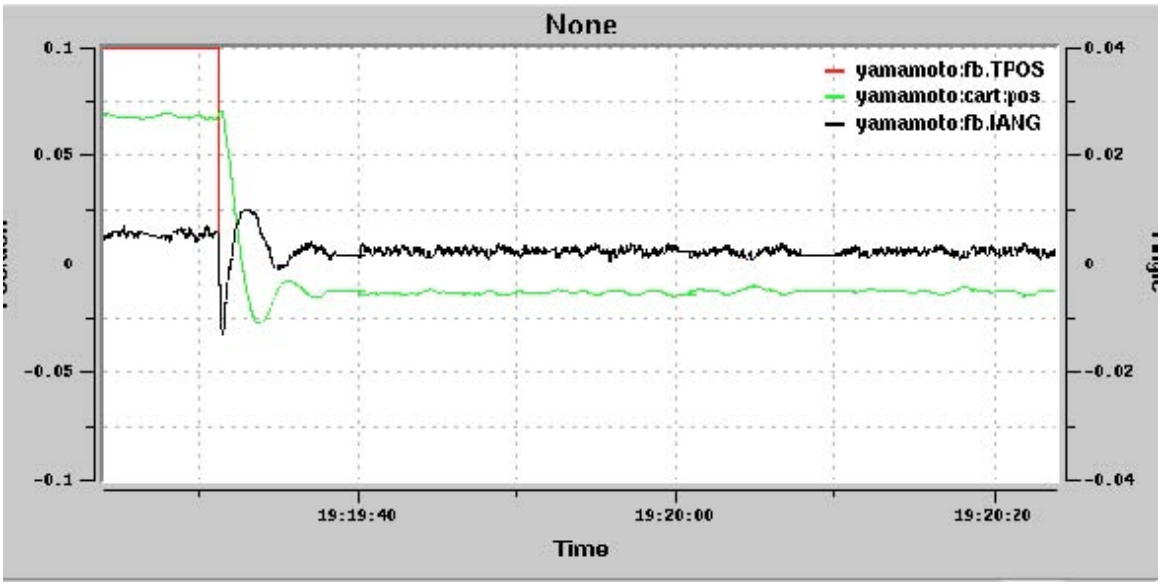

図. 1.5: 実機のステップ入力に対する応答の実測値。赤：台車位置設定値、緑：台車位置の読み出し値、黒：振り子の振れ角 (rad)、横軸：時刻（全幅が一分）

フィードバックの調整

ここで採用した状態フィードバックでも最適化条件のパラメータを調整することで、最終的な動作の調整を行う。よく使われる PID 制御の調整と比べてこの方法が有利なのは、パラメータを調整してもシステム全体の安定性は保証されていることである。

フィードバックの調整は Eq.(1.13) のパラメータ Q および R を変更することで行う。Q および R は任意の正定値の行列で良いが、実際的には対角行列が使われることが多いでしょう。全体のスケールを変更しても、結果を変えないことから、独立なパラメータとしては、4+1-1 = 4 のパラメータがあることになります。

台車の位置がレールの範囲の中に収まっていること、振り子が倒立状態を保つこと、モータ駆動電圧が 10V を超えないことを考えて、最終的に (2019.9.26 現在) 次のパラメータを採用しています。

$$\begin{aligned} Q &= diag([1./0.05**2, 1/0.2**2., 1., 1.]) \\ R &= 1./3.**2 * diag([1]) \end{aligned} \tag{1.45}$$

このパラメータを採用した時のシステムのステップ応答は次の図 (図. 1.6) のようになります。

離散化アルゴリズムの選択

計算機制御でこの制御を実行するためには、離散化の手続きが必要です。離散化の方法を変えた時の、システムのステップ応答の違いをみてみます。ここでは、python control moddule がサポートする `zoh`, `euler`, `bilinear`, `backward-diff` の四つの場合について結果を示します。それぞれの図には、サンプリングの周期を変えた場合の結果を合わせて表示しています。最も単純でかつ matlab でも python control でも default のアルゴリズムである `zoh` が悪くありません。 `backward-diff` の結果もよく似ています。`bilinear` の結果は、制御信号に振動がみられます。 `euler` は話になりません[9]。ということで、2019 年 9 月 26 日現在のパラメータは ‘‘backward-diff‘‘を採用しています。

[9] サンプリングの時間が十分に短ければ、euler(gbt alpha=0) も安定した結果を与える。 Ts=0.01 程度では、この問題に対して euler は安定なシステムを与えない。アルゴリズムの問題なのか、python control の実装上の問題なのかは未調査。

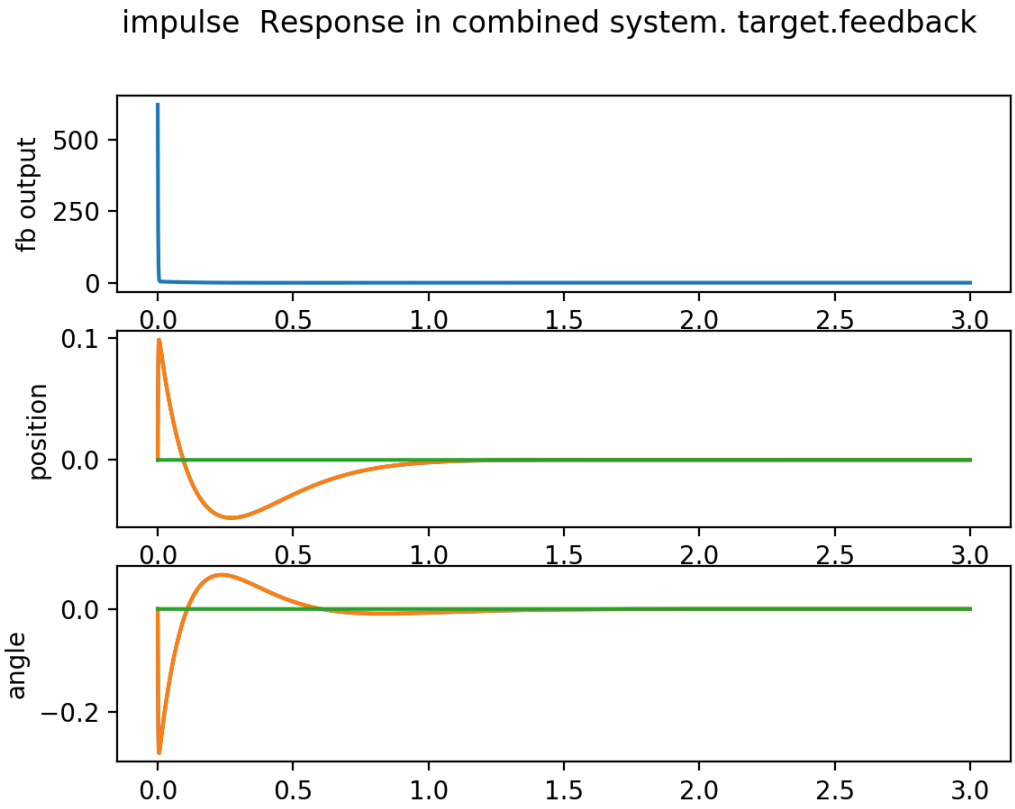

図. 1.6: システムのステップ応答。最上段のグラフはフィードバックの出力、二段目は台車の位置、三段目は振り子の角度を示している。この図の作成は pytho control モジュールの step_response 関数を用いて行っています。

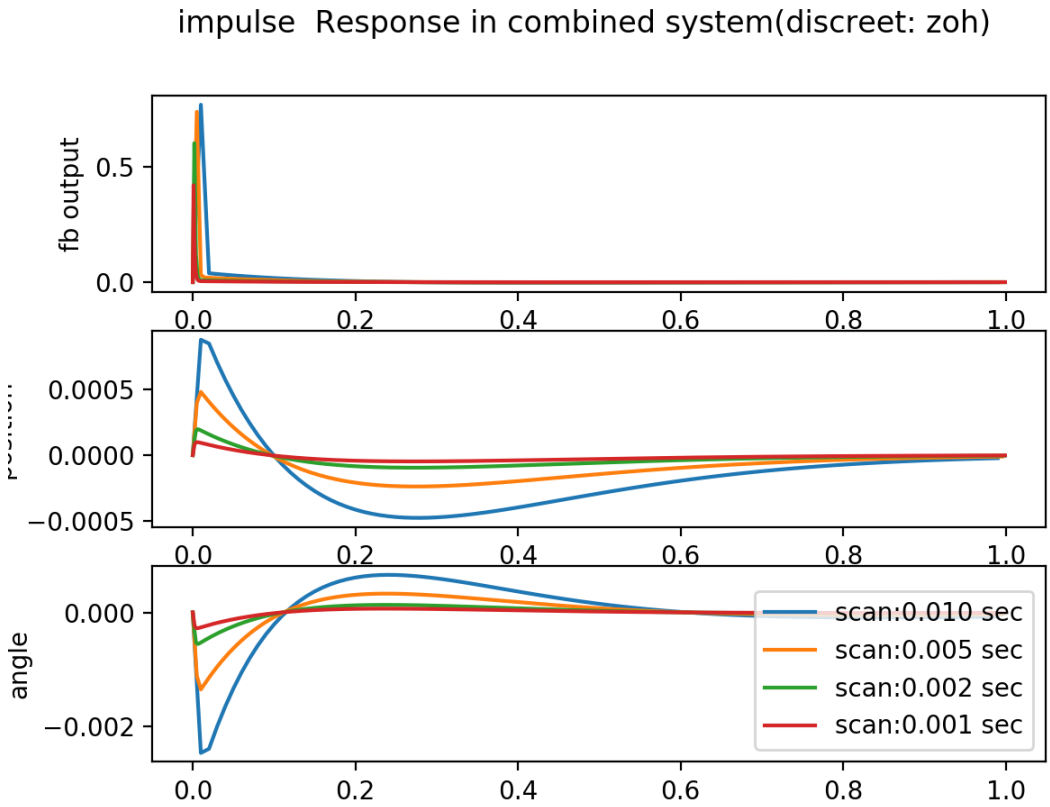

図. 1.7: 離散化アルゴリズムとして `zoh(zero-order hold)` を使った場合のシステムのステップ応答

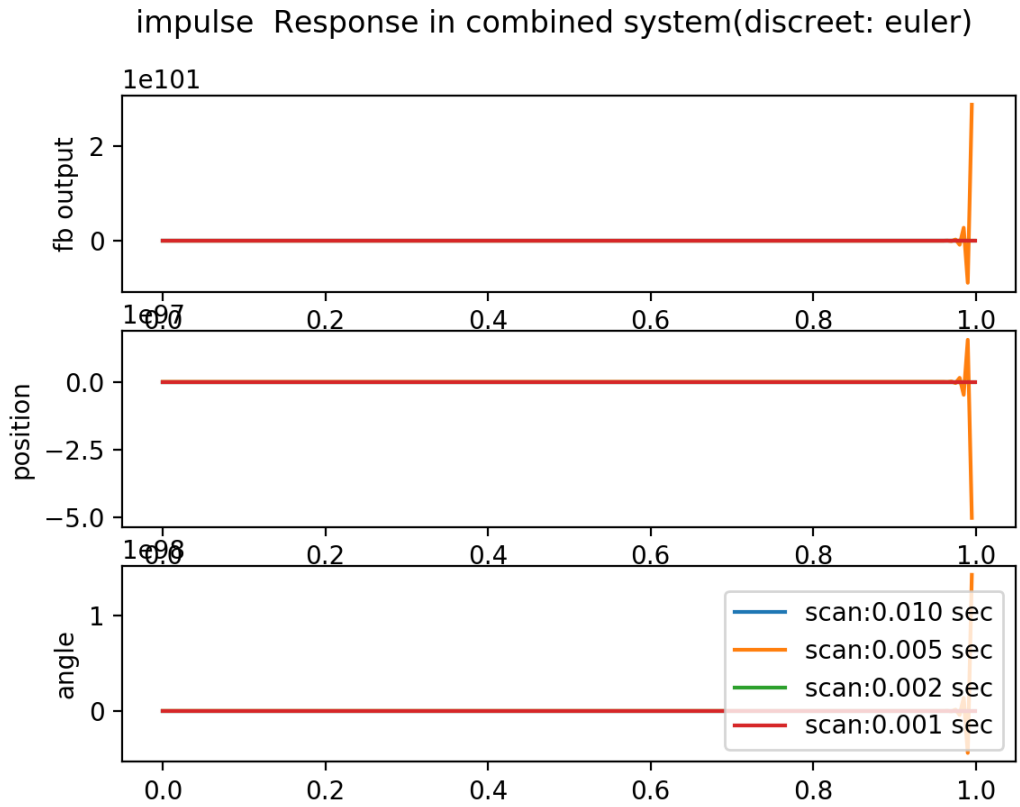

図. 1.8: 離散化アルゴリズムとして `euler(gbt alpha=0)` を使った場合のシステムのステップ応答

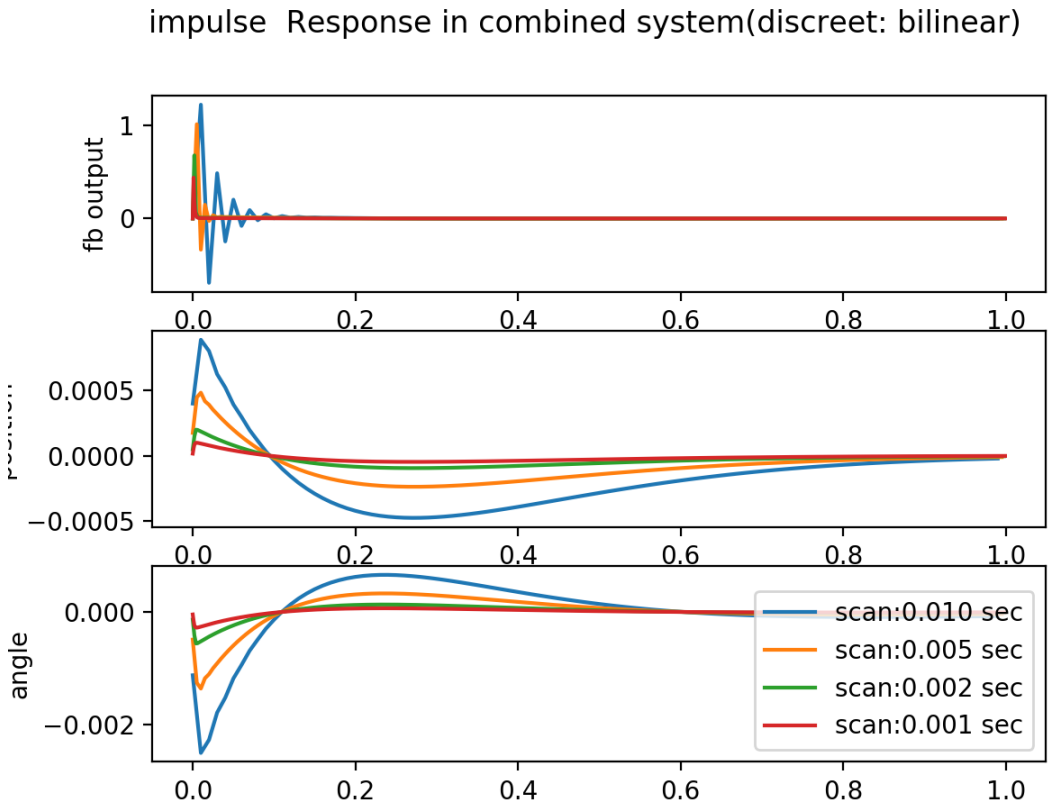

図. 1.9: 離散化アルゴリズムとして `bilinear(gbt alpha=0.5)` を使った場合のシステムのステップ応答

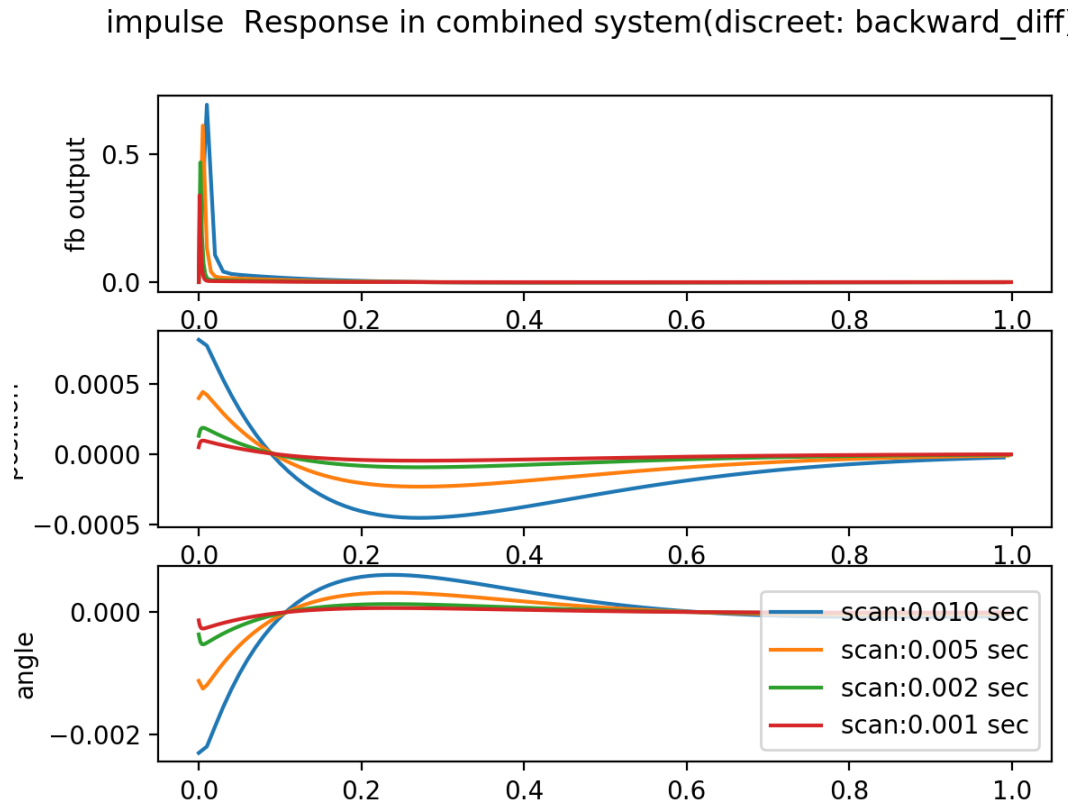

図. 1.10: 離散化アルゴリズムとして `backward_diff(gbt alpha=1.0)` を使った場合のシステムのステップ応答

1.7 まとめ

倒立振り子を例として、状態推定器付き状態フィードバックによるレギュレータの構成と設計法の概略を示した。Octave（や Matlab, python) を使うことで、これらの設計法がより身近なものとなりました。

EPICS を用いて、倒立振り子装置を実際に制御してみました。結果は線型のシミュレーション結果と定性的にはよく一致しています。今回は状態方程式で使われる、振り子の重さや、慣性モメントなどの定数は文献 [A-3] のものを用いています。この装置はほとんど手作りであることを考えると、これらの定数も実測値のもとづいたものを使えば、よりシミュレーションと測定値の比較に意味が出るでしょう。これらの定数を、システム同定の手法を用いて同定する事で、安定性／制御性の向上が図れるでしょう[10]。

制御理論としては、此処で採用した線型理論の状態推定器による制御則より進んだ制御方式として、H∞理論などのロバスト制御や、非線型性や時間ドリフトの影響を取り除くための、適応制御などの方式が知られています。これらの制御方式は、加速器制御にたいしても有効な方法となり得ます。これらの最新の制御方式の検討を続け、テストベンチとしてこの装置を利用することも期待されます。動的に繰り返し周波数を変更する、あるいは適応制御的にフィードバックパラメータそのものを変更することを考えると、`c2d()` や `lqr()` 相当のルーチンを VxWorks 上で用意する必要がある。すでに LAPACK のサブセットである、 LAPACK-lite は VxWorks 上で稼働の実績があるので、これらのルーチンを入手あるいは開発することが必要である。これらは今後の課題である[11]。

[10] システム稼働中に AI などの技術を応用して、これらのシステム定数の同定を同時に行うことも考えられます。時間的にゆっくりと変化するドリフトにも対応できるようになると考えられます。

[11] KEKB,J-PARC などの加速器制御で使われる IOC の OS の主流は Linux ベースのものになっています。これらのルーチンを使った動的最適化を実現することの障壁はなくなったとも言えるでしょう。

付 録A　Appendix

A.1 倒立振り子の運動方程式

倒立振り子装置は水平方向に一次元的に動く台車および一端がその台車に固定された倒立振り子から構成されている。台車と倒立振り子の質量をそれぞれ、 M および m で表す。また、倒立振り子の重心 (x, y) の周りの慣性モーメントを I とする。台車の水平方向の位置および倒立振り子の鉛直方向からはかった角度をそれぞれ z および θ とする。このとき、系のラグランジアンは次のように書き下せる。

$$L = \frac{1}{2}M\dot{z}^2 + \frac{m}{2}\left(\dot{x}^2 + \dot{y}^2\right) + \frac{I}{2}\dot{\theta}^2 - mgl\cos\theta \tag{1.1}$$

此処で g は重力加速度である。倒立振り子の一端が台車に固定されていることは拘束条件、

$$x = z - l\sin\theta \tag{1.2}$$

$$y = l\cos\theta \tag{1.3}$$

で表せる。この拘束条件を解いて、ラグランジアンに代入することで、拘束条件のないラグランジアン、

$$L = \frac{1}{2}M\dot{z}^2 + \frac{m}{2}\left[\left(\dot{z} - l\cos\theta\dot{\theta}\right)^2 + l^2\sin^2\theta\dot{\theta}^2\right] + \frac{I}{2}\dot{\theta}^2 - mgl\cos\theta \tag{1.4}$$

を得る。

A.1.1 線型化

θ の変化が微小な範囲では、運動方程式を線型化近似で取り扱う。式 (51) のラグランジアンを微小な θ について展開し、θ の二次の範囲までをとると、

$$L_2 = \frac{1}{2}Mz^2 + \frac{m}{2}\left[\dot{z} - l\dot{\theta}\right]^2 + \frac{I}{2}\dot{\theta}^2 + mgl\frac{\theta^2}{2} \tag{1.5}$$

となる。これより運動方程式は、

$$(M+m)\,\ddot{z} = ml\ddot{\theta} \tag{1.6}$$

$$\left(I + ml^2\right)\ddot{\theta} = ml\ddot{z} + mgl\theta \tag{1.7}$$

となる。

A.2 振り上げについての若干の考察。(未完)

フィードバックの失敗、大きな擾乱などにより、ある程度以上の振り子の傾きや台車位置がそのリミットに到達するような時には、フィードバックの異常動作を避けるため、フィードバックの働きを一時的に停止し、台車位置をレール中心に移動させる。プログラムはさらに台車を振り子の動きに合わせて動かすことで、倒れてしまった振り子を倒立の位置に持ってくる。効率的にこの操作を行うための考察をここに述べる。(未完)

A.3 Octave

Octave は GPL(GNU General Public License) にしたがって配付されているフリーソフトウェアである。ほぼ MATLAB 互換の処理系であって、制御に関するパッケージも MATLAB の制御用パッケージのサブセット (＋独自拡張) が実装されている。この報告では、 Octave2.1.34 を使用した。 Octave はもともと MATLAB 互換なフリーソフトウェアを目指して開発が進められてきたが、一部では独自の発展を遂げている。

Octave のソースコードなどは、 http://www.octave.org/download.html から入手可能である。

2.1.34 は（2001 年 5 月 30 日現在）開発版の状態であり、安定版は 2.0.16 であるが、2.1.34 でも特に問題は起きていない。この報告で紹介した octave プログラムは、すべて 2.1.34 でのみ動作を確認している。 Octave の control パッケージは 2.0.16 にくらべ、 2.1.34 は大きく進歩している。　2.0.16 で実行するためには、かなりの変更が必要と思われる。

改訂版の執筆時 (2014 年 8 月) の最新版は 3.8.2 となっている。再改訂版執筆時 (2019 年 10 月 31 日) の最新版は、5.1.0(2019.2.25) となっている。

Linux/Windows/MacOSX に binary install が提供されている。ソースコードも提供されているので、自ら build することも可能である。ビルドには C コンパイラと fortran コンパイラ（たとえば gfortran) が必要である。此処で利用している control パッケージは octave 本体のインストール後に、別途インストールする手続きが必要である。octave に control パッケージをインストールするには、octave プロンプトで:

```
octave:1> pkg install -forge control
```

を実行すればよい。この他のオプションパッケージについては、http://octave.sourceforge.net/control/function/initial.html　<Octave Forge> に詳しい。

octave の Control システムライブラリは 2.x までとなっている。新しいライブラリは SLICOT とよばれるライブラリを利用する形で利用可能のようだが、調査中。SLICOT は Python ラッパーも用意されているらしい (slycot)。ただ、SLICOT の build には Fortran compiler が必要。

http://hpc.sourceforge.net/

から gfortran や g77 はダウンロード出来るみたいである。残念ながら、Python/slycont/SLICOT の c2d 関数は MIMO 系 (ここで考えているシステムは　入力 2、出力 1 の MIMO 系）はまだサポートされていない。(MIMO がサポートされないのは、現代制御理論としては片手落ちと云う気がするけれど）ということで、現在 Freesoft でこれらの係数を計算し直す方法がない。(Octave2 を動くようにする、Octave3 で control システムが使えるようになるのをまつ等の手段を考えるのか？)

A.3.1 極配置法

Octave の標準制御パッケージには、極配置法によるフィードバックゲインを支援するための関数、 place が用意されている。 MATLAB の place とは異なり、一入力系にしか適用出来ない (残念ながら)。

```
.. _CodeOctPoles:
```

```
sys=ss2sys(A,B,C,D);
poles=[-240, -5+2 j, -5 -2 j , -5];
K=place(sys,poles);
```

A.4 Python-control モジュール

プログラム言語 Python でも MATLAB/Octave と同等の制御システム向けライブラリが開発されている。python-control モジュール [A-1] がそれである。

```
python -m pip install control scipy slycot
```

A.4.1 Python control モジュールによる設計例

本文中で示された matlab/octave のプログラムを python.control を用いて書き直してみた。一部を取り出して紹介する。

プログラム 1.1: 制御対象をモデル化した State Space を設定する。

```
A=np.matrix(
       [[0.00000000e+00,   0.00000000e+00,   1.00000000e+00, 0.00000000e+00],
       [0.00000000e+00,   0.00000000e+00,   0.00000000e+00, 1.00000000e+00],
       [0.00000000e+00,   0.00000000e+00,  -160.22529505e+02, 0.0],
       [0.00000000e+00,   43.66133683,   573.55755483, 5.00908711]])
B=np.transpose(np.matrix([[0.,   0.         , 94.57663738, -337.84229409]]))
# 角度と位置がシステムの出力（観測値）
C=np.matrix([[1, 0, 0, 0],[ 0, 1, 0, 0 ]]);
D=np.matrix([[0], [0]]);
matrix_rank(A)
matrix_rank(ctrb(A,B))
```

プログラム 1.2: Observer のゲインを決める。

```
sigw=diag([40,1,1,1])
sigv=eye(2)
# solve Ricatti equation.
#[H,p,e]=lqe(A,G,C,sigw,sigv);
[H,p,e]=lqr(np.transpose(A), np.transpose(C), sigw, sigv)
H=np.transpose(H)
```

プログラム 1.3: Optimal Requlator のゲインを求める。

```
Q=diag([10.0, 4.0, 1.0, 1.0])
R=eye(1)
[Kr,p,e]=lqr(A, B, Q, R)
eig(A-B*Kr)
```

python-control の `bode()` 関数は SISO 系の場合だけに有効である MIMO の場合には、`freqresp()` メソッドあるいは `freqresp()` 関数を使って、周波数応答を求めたあと、Bode 図を描く。グラフの作成には、python.matplotlib.pyplot を利用している。

プログラム 1.4: Bode Diagram を表示する。

```
freqs=logspace(-2,1,1024)
mag,phase,omega=sys.freqresp(freqs)

pyplot.subplot(2,1,1)
pyplot.plot(omega, mag[0,0])
```

(次のページに続く)

(前のページからの続き)

```
pyplot.subplot(2,1,2)
pyplot.plot(omega, phase[0,0])
```

振幅 mag および 位相 phase は、主力信号、入力信号、および周波数に対応した指標を持つ三次元の配列として求められる。

上記の状態フィードバックゲイン Kr と状態推定きのゲイン H を使って、コントローラーを定義し、そのコントローラによるフィードバックシステムを構成する。

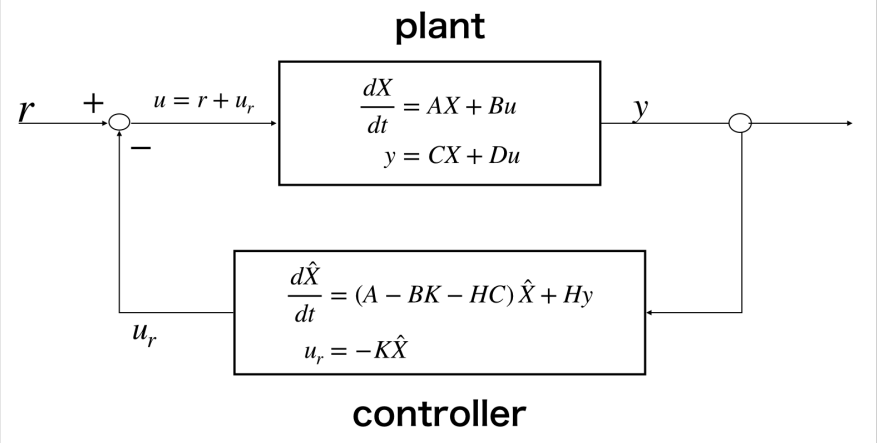

図. 1.1: Feedback system のダイアグラム:コントローラは対象システム (plant) の出力から、状態量を推定、推定状態量に基づき、制御量を決定する。

プログラム 1.5: Feedback response

```
from control import StateSpace,initial_responcse,feedback

plant=StateSpace(A,B,C,D)

controller=StateSpace(A-B*Kr-H*C, H, Kr, matrix(zeros((1,2))))

fbsys=series(plant, controller).feedback(1)
# or
fbsys=target.feedback(controller)
```

図. 1.3 はこのフィードバックシステムによるインパルス応答を計算した例です。

プログラムに実装する際には、連続系のパラメータを離散系に変換する。matlab/octave では *c2d* が使われたが、python.control モジュールでは, control.sample_system() 関数あるいは　システムオブジェクトの sample() メソッドを使う。離散化の方法 (method) として、"zoh"(default),"bilinear(tustin)", "forward_diff(euler)","backword_diff"が用意されている。応答関数で定義される SISO システムでは"matched"もサポートされる。

```
discrete_sys=sample_system(sys, scanperiod, "bilinear"
```

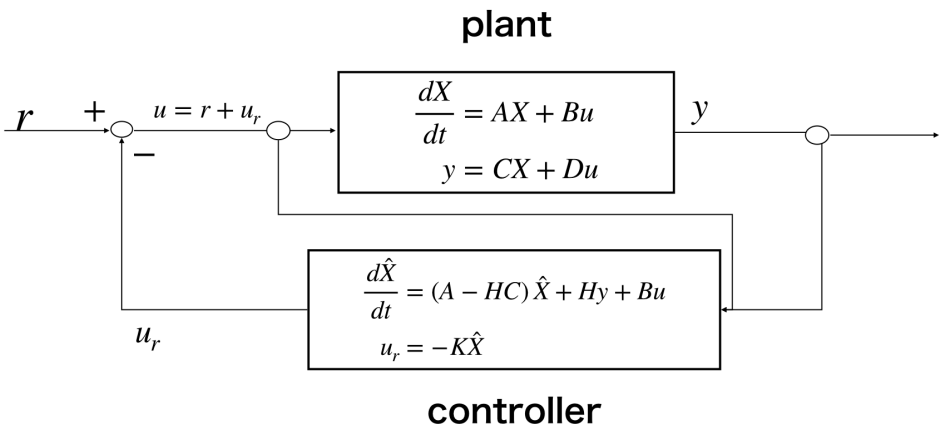

図. 1.2: Feedback system のダイアグラム：コントローラにシステム入力を接続した場合。状態推定には、plant の出力 (y) および plant の入力 (u) を用いる。plant への参照値 (r) が変化した場合にも、plant の状態と推定値の差が大きくならないと期待される。

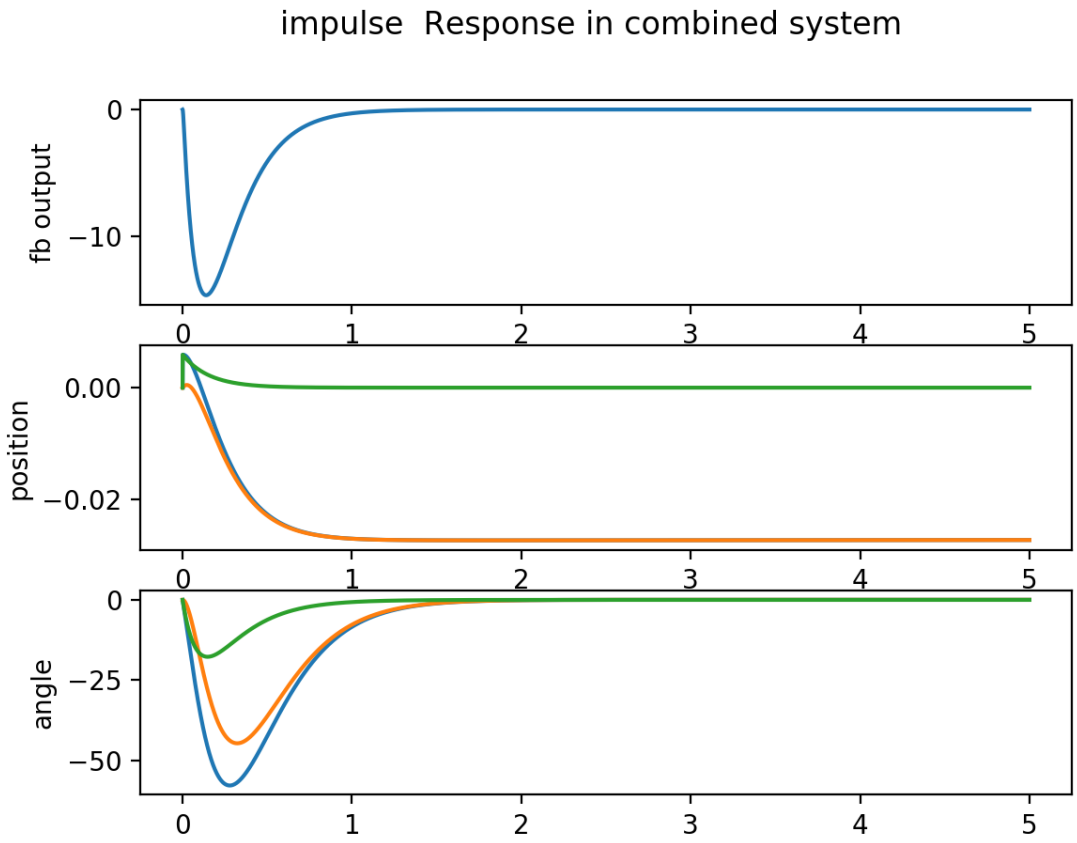

図. 1.3: 状態フィードバックによるインパルス応答。`control.impulse_response()` を用いた。一段目はコントローラーからの出力 (モーターの速度指令値). 二段目と三段目はそれぞれ、台車の位置と振り子の角度を示す。青が対象系の状態値、橙色は状態推定器の推定値、緑は状態値と推定値の差を示している。

A.5 MATLAB から　Python-control モジュールへの移行

python.control モジュールには control.matlab モジュールが含まれており、matlab の制御関係の関数などが含まれている。しかしながら、完全な互換性があるわけでは無い。　東京電機大学出版会から出版されている「MATLAB による制御理論の基礎」および「MATLAB による制御系設計」に掲載されている MATLAB プログラムは、「MATLAB による制御系設計」に付属の CD-ROM に収録されている。これらのプログラムのいくつかを Python.control モジュールを使った Python モジュールに書き直してみた。この際に、注意すべき点をいくつか上げておく。

A.5.1 import するモジュール

制御関係の関数などは、control.matlab モジュールを import することで、概ね使えるが、グラフの描画などのために matplotlib から必要な関数を import する必要がある、

```
from control.matlab import *

from numpy.linalg import matrix_rank as rank, eig, inv, pinv,svd
from numpy import matrix,zeros,diag,eye,sqrt,log10

from matplotlib.pyplot import clf,plot,show,draw,figure,subplot
from matplotlib.pyplot import xlabel,ylabel,axis,loglog,semilogx,semilogy,xlim,
↪ylim,legend,title,grid,suptitle
```

これらの関数をインポートしておけば、大体の用は足りるでしょう。numpy, matplotlib に代えて `pylab()` を `import *`してしまうのも一つの考え方です[1]

A.5.2 配列の定義

matlab では行列の表記が文法に組み込まれており、簡単に行列を入力することが出来ます。例えば、

```
A=[ 1 2 3; 4 5 6]
```

は $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ を変数 A に割り当てます。

python では numpy の matrix を使って

```
A=numpy.matrix(
(
    (1,2,3),
    (4,5,6)
))
```

と定義することで、変数 A に行列を割り当てます。実は numpy の行列は、

```
A=numpy.matrix(" 1 2 3; 4 5 6")
```

といった定義も可能です。しかし、

[1] pylab のディクショナリには、900 以上の名前が登録されており、これらを全て global スコープに import してしまうと、思わぬところで名前の衝突が起きるかもしれません。

```
a=1;b=2
A=numpy.matrix(" a b 3; 4 5 6")
```

のように変数名をその値に含むような表式は受け付けてもらえません。結局のところ、

```
a=1;b=2
A=numpy.matrix([[a, b, 3],[4, 5, 6]])
```

と書き換えることが必要になります。ということで、実用的には `numpy.matrix()` を使うことになるでしょう。

A.5.3 複素数の取り扱い

matlab での複素数は '1+sqrt(2)　i'　のように記号 'i' を使って表現されます。python では複素数は `complex(1,sqrt(2))()` のように complex() 関数を使って複素数オブジェクトを作成することになります。

```
% in matlab/octave
c= 1+ sqrt(2) i

# in python
c=complex(1, sqrt(2))
```

A.5.4 対象システムの指定

matlab では応答関数を表現するのに、s 多項式の分子 (num) と分母 (den) のペア ([num, den]) を使うことが出来ます。このペアを、bode,step などの関数に直接引数として渡すことが出来ます。これらの関数では、StateSpace を表現する四つの行列のくみ、'A, B, C, D' を引数に指定すること一方、python.control ではこれらの関数 (bode, step ...) では、引数として渡せるシステムは StateStat か Transfer Function のオブジェクトになり、このペア [num, den] (あるいは行列の組）を渡すことは出来ません。[num, den] のペアから応答関数のオブジェクトを `tf()` で作成して（あるいは ss 関数を使って、行列の組から State Space オブジェクトを作って)、これを引数として渡します。

```
% in matlab/octave
bode([num, den])
step(A,B,C,D)

# in python
bode(tf(num,den))
step(ss(A,B,C,D))
```

プロパーな応答関数と状態空間表現との間は、`tf2ss()` あるいは `ss2tf()` でお互いに変換可能です。

`tf2ss()` と `ss2tf()` は概ね交換可能で、`tf2ss(ss2tf())()`　あるいは `ss2tf(tf2ss())()` 元のオブジェクトに戻ることと思いますが、例外的にこれが成り立たない場合があります。

具体的な例を示します。

```
>>> ss2=ss(-1,2,1,0)
>>> ss2tf(ss2)
ss2tf(ss2)

  2
-----
s + 1
```

ですが、tf2ss(ss2tf(ss1)) は ValueError になってしまいます。これは python control では応答関数 tf([2],[1,1]) を定義することは出来ますが、これを tf2ss() で変換することが出来ず、ValueError になることが原因です。 原理的には、StateSpace で定義されたシステムを応答関数に変換しても、再び状態空間表示に戻すことは、いつも可能なのですから、これは tf2ss の実装上の問題と思われます。(python control ではシステムは、状態空間表示を基本とする方が良さそうです。)

応答関数を零点と極を指定することで指定することが出来ます。MATLAB ではこれには zp2tf 関数が使われます。python.contorl.matlab では、scipy.signal モジュールからインポートされた zpk2tf あるいは zpk2ss 関数を使用します。これらの関数は scipy.signal の関数であるため, [num,den] ペアあるいは 四つの行列の組みを値として返します。これらを python control でのシステムオブジェクト (LTI オブジェクト）とするには tf() あるいは ss() を使う必要があることに注意しましょう。

zpk 表示との変換では、以下の例のように、tf2ss でおきた問題を生じないことがわかります。

```
ss(*zpk2ss(*ss2zpk(ss2.A,ss2.B,ss2.C,ss2.D)))
```

A.5.5 システムの離散化

連続系のシステムを離散化するプログラム c2d は control.matlab にも用意されていますが、MATLAB のそれ以外の離散化関数は用意されていません。また、c2d() は MATLAB のそれとは異なり、[num,den] ペアを受け付けることもありません。というわけで、c2d を使うより、StateSpase や Transfer Function のオブジェクトと sample メソッドを使う方が簡単でしょう。また、matlab の c2d など関数では、"zoh"(zeroth order hold) の他に、"foh"(first order hold), "bilinear"など様々な方法が用意されています。sample では zoh がデフォルトですが、その他には、"bilinear","euler","backward_diff"そして"gbt"が用意されています。これらの変換は zoh を除き全てが、gbt(generalized bilinear transformation) のグループに属しており、gbt が要求するパラメータ α が bilinear(0.5), euler(0), backward_dif(1.0) の場合になっています。前記の教科書を参考にした離散化アルゴリズムによるシステム応答の違いを python.control で計算した結果を以下に示します。

A.5.6 システム応答を求める関数

control.matlab にはシステムの応答を求める関数として、MATLAB と同じく, bode,impulse,step,initial の関数が用意されています。

また、これらの機能のうち、よく使われる bode 図, 根軌跡図、閉ループのステップ応答をまとめて作成する sisotool が用意されています。 control.sisotools は MATLAB の同名のパッケージにちなんで導入されたとのことです。

```
# python
sys = control.tf([1000], [1,25,100,0])
control.sisotool(sys)
```

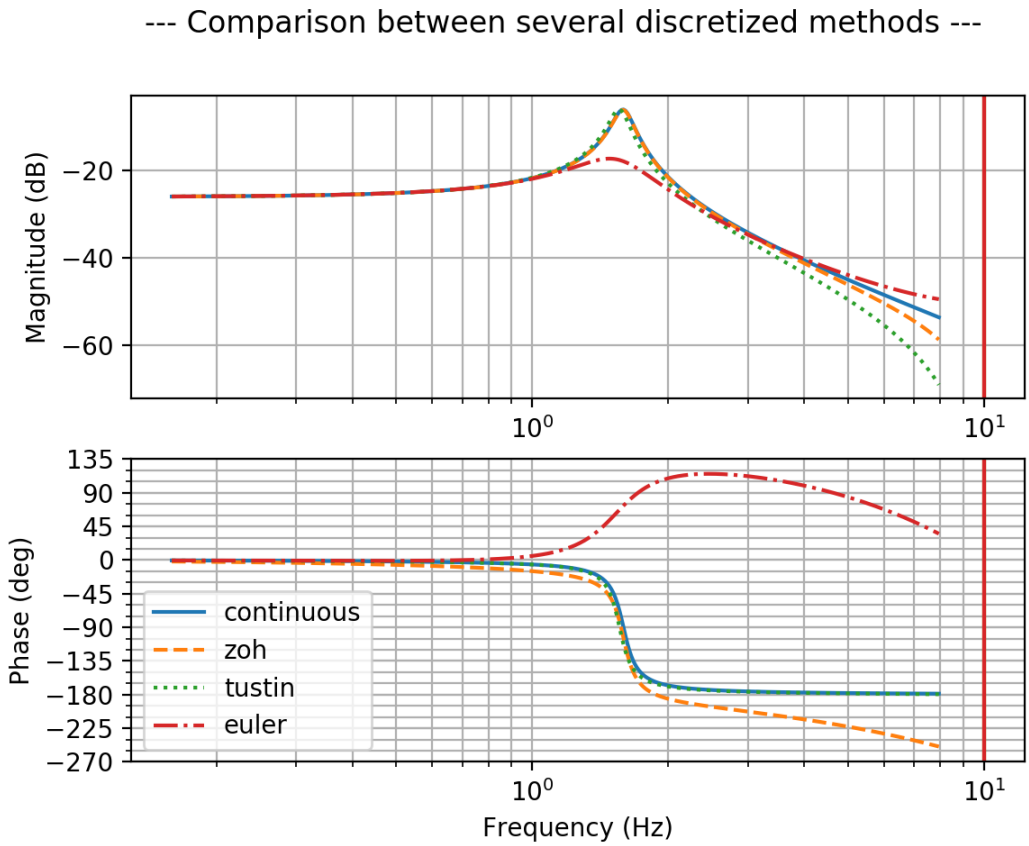

図. 1.4: システムの離散化法によるシステム応答 (Bode 図）の違い

を実行することで、 次のような図が出力されます。

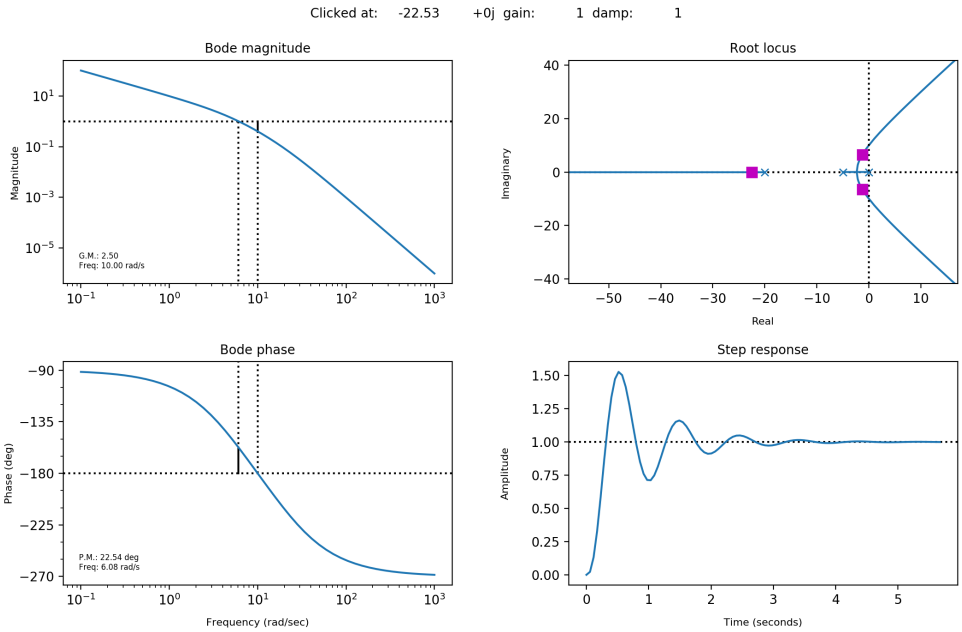

図. 1.5: control.sistool による出力, Bode 図, root locus 図、閉ループのステップ応答がまとめて出力されます。root locus 図をマウスでクリックすることで、閉ループのゲインが変更されます。

bode, rlocus, sisotool は内部で matplotlib の plot コマンドを発行してグラフを描画します (つまりプログラム全体のどこかで、show コマンドが実行されることが必要だということです)。 その他のシステム応答関数 step/impulse/initial/freqresp は描画のためのデータを返しますが、描画自体は別途 明示的にプログラムする必要があります。 これらのシステム応答関数は MATLAB のそれと同様のデータを返しますが、戻り値の順番、数が異なっている場合があります。その場合、python control/control.matlab 側の関数に呼び出し時のオプションを追加する必要があります。例えば step 関数では、状態変数 x の変化を出力する場合には、'return_x' オプションの指定が必要です。

```
% MATLAB
y,t=step(sys)
y,t,x = step(sys)
```

```
# python
t,y = control.step_response(sys)
y,t = control.step(sys)
# python では状態変数のシミュレーション結果は明示的に要求。
t,y,x=control.step_response(sys,return_x=True)
y,t,x=control.step(sys,return_x=True)
```

付 録B　Ricatti 方程式

B.1 Ricatti 方程式の導出[1]

最適レギュレータは状態方程式 [Eq.(2.1)]

$$\frac{d\mathbb{X}}{dt} = \mathbb{A}\mathbb{X} + \mathbb{B}\mathbf{u} \tag{2.1}$$

の下で、次の式で定義される評価汎関数 $I[\mathbb{X}, \mathbf{u}, t]$ を最小化するという問題と捉えることができます。ここで、Q および R は正定値の対称行列とします。

$$\mathcal{O}[\mathbb{X}, \mathbf{u}] = \int_{-\infty}^{\infty} dt \frac{1}{2} \left\{ \mathbb{X}^T Q \mathbb{X} + \mathbf{u}^T R \mathbf{u} \right\} \tag{2.2}$$

条件付きの最小値問題ということで、Lagrange の未定係数 Λ を用いて拡大された評価汎関数 $\tilde{\mathcal{O}}$ を次のように定義してみましょう。

$$\begin{aligned} \tilde{\mathcal{O}}\left[\mathbb{X}, \mathbf{u}, \mathbf{\Lambda}\right] = & \int_{-\infty}^{\infty} dt \\ & \left\{ \frac{1}{2} \mathbb{X}^T Q \mathbb{X} + \frac{1}{2} \mathbf{u}^T R \mathbf{u} + \mathbf{\Lambda}^T \left(\frac{d\mathbb{X}}{dt} - \mathbb{A}\mathbb{X} - \mathbb{B}\mathbf{u} \right) \right\} \end{aligned} \tag{2.3}$$

この汎関数の独立変数の変分をとると、$\mathbf{\Lambda}$ についての変分からは状態方程式 Eq.(2.1) が導かれることは直ちにわかります。 $\mathbf{u}$ の変分からは、

$$\begin{gathered} \frac{\delta \tilde{\mathcal{O}}}{\delta \mathbf{u}^T} = R\mathbf{u} - \mathbb{B}^T \mathbf{\Lambda} = 0 \\ or \\ \mathbf{u} = R^{-1} \mathbb{B}^T \mathbf{\Lambda} \end{gathered} \tag{2.4}$$

が導かれます。　最後に $\mathbb{X}$ についての変分からは、

$$\frac{\delta \tilde{\mathcal{O}}}{\delta \mathbb{X}^T} = Q\mathbb{X} - \frac{d\mathbf{\Lambda}}{dt} - \mathbb{A}^T \mathbf{\Lambda} = 0 \tag{2.5}$$

が導かれます。

ここで、 Eq.(2.4) が状態フィードバックの制御則になることを要求して、

$$\mathbf{\Lambda} = -\mathbb{P}\mathbb{X} \tag{2.6}$$

で新たな量, $\mathbb{P}$ を導入します。

この定義から、

$$\frac{d\mathbf{\Lambda}}{dt} = -\frac{d\mathbb{P}}{dt}\mathbb{X} - \mathbb{P}\left(\mathbb{A}\mathbb{X} - \mathbb{B}R^{-1}\mathbb{B}^T\mathbb{P}\mathbb{X}\right) \tag{2.7}$$

[1] Ricatti 代数方程式については Liapnov 関数の方式の方がわかり易いかもしれない。これは、第 1.4.2 章の議論の裏返しでもある。

です。これを Eq.(2.5) に代入すると、

$$Q\mathbb{X} + \frac{d\mathbb{P}}{dt}\mathbb{X} + \mathbb{P}\left(A\mathbb{X} - \mathbb{B}R^{-1}\mathbb{B}^T\mathbb{P}\mathbb{X}\right) + \mathbb{A}^T\mathbb{P}\mathbb{X} = 0 \tag{2.8}$$

となります。初期値に関わらず、この条件が成り立つためには、

$$Q + \frac{d\mathbb{P}}{dt} + \mathbb{P}\mathbb{A} + \mathbb{A}^T\mathbb{P} - \mathbb{P}\mathbb{B}R^{-1}\mathbb{B}^T\mathbb{P} = 0 \tag{2.9}$$

をみたす $\mathbb{P}$ が存在すれば良いことになります。定常状態では $\frac{d\mathbb{P}}{dt} = 0$ ですから、この式, Eq.(2.9) は

$$Q + \mathbb{P}A + \mathbb{A}^T\mathbb{P} - \mathbb{P}\mathbb{B}R^{-1}\mathbb{B}^T\mathbb{P} = 0 \tag{2.10}$$

と Ricatti の代数方程式に帰着されました。この方程式が正定値対称な解 $\mathbb{P}$ をもてば、それが $I[\mathbb{X}, \mathbf{u}, t]$ を最小化するという意味で、最適なかつ安定な状態フィードバックを決めることがわかりました。 この時の制御則は、

$$\mathbf{u} = R^{-1}\mathbb{B}^T\mathbf{\Lambda} = -R^{-1}\mathbb{B}^T\mathbb{P}\mathbb{X} \tag{2.11}$$

となります。

B.2 ハミルトン行列

式 Eq.(2.3) ‘ をラグランジアンと考えて, ハミルトン形式に書き換えることを考えてみます。$\mathbb{X}$ に対する"正準運動量"$\mathbf{P}$ を、

$$\begin{aligned}\mathbf{P} &= \frac{\delta\tilde{\mathcal{O}}}{\delta\frac{d\mathbb{X}^T}{dt}} \\ &= \mathbf{\Lambda}\end{aligned} \tag{2.12}$$

と定義すると、ハミルトニアン $\mathcal{H}$ は,

$$\begin{aligned}\mathcal{H} &= \mathbf{P}^T\frac{d\mathbb{X}}{dt} - L \\ &= \mathbf{P}^T\frac{d\mathbb{X}}{dt} - \left\{\frac{1}{2}\mathbb{X}^TQ\mathbb{X} + \frac{1}{2}\mathbf{u}^TR\mathbf{u} + \mathbf{P}^T\left(\frac{d\mathbb{X}}{dt} - \mathbb{A}\mathbb{X} - \mathbb{B}\mathbf{u}\right)\right\} \\ &= -\frac{1}{2}\mathbb{X}^TQ\mathbb{X} - \frac{1}{2}\mathbf{u}^TR\mathbf{u} + \mathbf{P}^T\left(\mathbb{A}\mathbb{X} + \mathbb{B}\mathbf{u}\right)\end{aligned} \tag{2.13}$$

となります。更に $\mathbf{u} = R^{-1}\mathbb{B}^T\mathbf{P}$ を代入することで、

$$\begin{aligned}\mathcal{H} &= -\frac{1}{2}\mathbb{X}^TQ\mathbb{X} - \frac{1}{2}\mathbf{P}^T\mathbb{B}R^{-1}\mathbb{B}^T\mathbf{P} + \mathbf{P}^T\left(\mathbb{A}\mathbb{X} + \mathbb{B}R^{-1}\mathbb{B}^T\mathbf{P}\right) \\ &= \frac{1}{2}\mathbf{P}^T\mathbb{B}R^{-1}\mathbb{B}^T\mathbf{P} - \frac{1}{2}\mathbb{X}^TQ\mathbb{X} + \mathbf{P}^T\mathbb{A}\mathbb{X}\end{aligned} \tag{2.14}$$

と書けます。

この時、"正準方程式"は、

$$\begin{aligned}\frac{d\mathbb{X}}{dt} &= \frac{\delta\mathbf{H}}{\delta\mathbf{P}^T} = \mathbb{A}\mathbb{X} + \mathbb{B}R^{-1}\mathbb{B}^T\mathbf{P} \\ \frac{d\mathbf{P}}{dt} &= -\frac{\delta\mathbf{H}}{\delta\mathbb{X}^T} = Q\mathbb{X} - \mathbb{A}^T\mathbf{P}\end{aligned} \tag{2.15}$$

となります[2]。行列表示では、

$$\frac{d}{dt}\begin{pmatrix}\mathbb{X}\\ \mathbb{P}\end{pmatrix}=\begin{pmatrix}\mathbb{A}, & \mathbb{B}R^{-1}\mathbb{B}^T\\ Q, & -\mathbb{A}^T\end{pmatrix}\begin{pmatrix}\mathbb{X}\\ \mathbb{P}\end{pmatrix} \tag{2.16}$$

です。　この拡大された状態方程式は、初期条件 $\mathbb{X}_0$ に対して、安定な固有値に対応する固有ベクトル ξ_i だけを使って、

$$\begin{pmatrix}\mathbb{X}_0\\ \mathbf{P}_0\end{pmatrix}=\Sigma_i a_i\xi_i=\begin{pmatrix}\mathbf{\Xi}_x\\ \mathbf{\Xi}_P\end{pmatrix}\ni \tag{2.17}$$

となる様に $\mathbf{P}_0$ を選んでやれば、この系は安定になることがわかります。

$$\mathbf{P}_0=\mathbf{\Xi}_P{\mathbf{\Xi}_x}^{-1}\mathbb{X}_0 \tag{2.18}$$

および $\mathbf{\Xi}$ の各列ベクトルはハミルトン行列の固有ベクトルであることに注意すると[3]。

$$\begin{pmatrix}\mathbb{A}, & \mathbb{B}R^{-1}\mathbb{B}^T\\ Q, & -\mathbb{A}^T\end{pmatrix}\mathbf{\Xi}=\mathbf{\Xi}\mathbf{\Lambda} \tag{2.19}$$

$$\begin{aligned}\mathbb{A}\mathbf{\Xi}_x+\mathbb{B}R^{-1}\mathbb{B}^T\mathbf{\Xi}_P&=\mathbf{\Xi}_x\mathbf{\Lambda}\\ \mathbb{Q}\mathbf{\Xi}_x-\mathbb{A}^T\mathbf{\Xi}_P&=\mathbf{\Xi}_p\mathbf{\Lambda}\end{aligned}$$

です. 上の二つの式から Λ を消去することで、

$$\mathbb{Q}-\mathbb{A}^T\mathbf{\Xi}_P{\mathbf{\Xi}_x}^{-1}=\mathbf{\Xi}_P{\mathbf{\Xi}_x}^{-1}\mathbb{A}+\mathbf{\Xi}_P{\mathbf{\Xi}_x}^{-1}\mathbb{B}R^{-1}\mathbb{B}^T\mathbf{\Xi}_P{\mathbf{\Xi}_x}^{-1} \tag{2.20}$$

が成り立つことがわかります。すなわち、$-\mathbf{\Xi}_P{\mathbf{\Xi}_x}^{-1}$ は Ricatti 方程式 Eq.(2.10) の解となっています。

また、正準方程式の時間発展に伴って、

$$\begin{aligned}\mathbf{P}(t)&=\mathbf{\Xi}_P{\mathbf{\Xi}_x}^{-1}\mathbb{X}(t)\\ \mathbf{u}&=R^{-1}\mathbb{B}^T\mathbf{\Xi}_P{\mathbf{\Xi}_x}^{-1}\mathbb{X}(t)\end{aligned} \tag{2.21}$$

が成り立つことも確認できます。

[2] $\begin{pmatrix}A & B\\ C & D\end{pmatrix}$ と $\begin{pmatrix}A & -B\\ -C & D\end{pmatrix}$ は同じ固有値を持つことに注意すれば、これらのハミルトン行列は文献 [A-7][minami:2019] などの表示と等価である。

[3] ここで、$\mathbf{\Xi}_x$ が正則であることを仮定している。これは可制御性から証明されるべきであるが、未着。

付 録C　状態空間

C.1 応答関数から状態方程式へ

応答関数が proper であれば、状態方程式に書き換えることが可能。

$$\begin{aligned} \frac{d\vec{X}}{dt} &= \mathfrak{A}\vec{X} + \mathfrak{B}\vec{u} \\ \vec{y} &= \mathfrak{C}\vec{x} + \mathfrak{D}\vec{u} \end{aligned} \tag{3.1}$$

C.2 解の積分表示

状態方程式

$$\frac{d\vec{X}}{dt} = \mathfrak{A}\vec{X} + \mathfrak{B}\vec{u} \tag{3.2}$$

の形式的な解は、初期条件を $\vec{X(0)} = \vec{X_0}$ として、

$$\vec{X(t)} = e^{\mathfrak{A}t}\vec{X_0} + \int_0^t d\tau e^{\mathfrak{A}(t-\tau)}\mathfrak{B}\vec{u(\tau)} \tag{3.3}$$

と書ける。実際、

$$\begin{aligned} \frac{d\vec{X(t)}}{dt} &= \mathfrak{A}e^{\mathfrak{A}t}\vec{X_0} + \mathfrak{A}\int_0^t d\tau e^{\mathfrak{A}(t-\tau)}\mathfrak{B}\vec{u(\tau)} + \mathfrak{B}\vec{u(t)} \\ &= \mathfrak{A}\vec{X(t)} + \mathfrak{B}\vec{u(t)} \end{aligned} \tag{3.4}$$

である。

C.3 離散系の状態方程式など

離散系の状態方程式を連続系のそれ、 Eq.(3.2) 、に習って、

$$\begin{aligned} \vec{X_n} &= \mathbb{A}\vec{X_{n-1}} + \mathbb{B}\vec{u_n} \\ \vec{Y_n} &= \mathbb{C}\vec{X_n} + \mathbb{D}\vec{u_n} \end{aligned} \tag{3.5}$$

とする[1]。

[1] u の添え字を n とするか、$n-1$ とするかは、よく考える必要がある。

連続系の状態方程式の形式解 Eq.(3.3) を使うと、

$$
\begin{aligned}
X(t \vec{+} \Delta t) &= e^{\mathfrak{A}(t+\Delta t)}\vec{X_0} + \int_0^{t+\Delta t} d\tau e^{\mathfrak{A}(t+\Delta t-\tau)}\mathfrak{B}u(\vec{\tau}) \\
&= e^{\mathfrak{A}\Delta t}\left(e^{\mathfrak{A}t}\vec{X_0} + \int_0^{t+\Delta t} d\tau e^{\mathfrak{A}(t-\tau)}\mathfrak{B}u(\vec{\tau})\right) \\
&= e^{\mathfrak{A}\Delta t}\left(e^{\mathfrak{A}t}\vec{X_0} + \int_0^{t} d\tau e^{\mathfrak{A}(t-\tau)}\mathfrak{B}u(\vec{\tau})\right) + \int_0^{\Delta t} d\tau e^{\mathfrak{A}(\Delta t-\tau)}\mathfrak{B}u(t+\tau) \\
&= e^{\mathfrak{A}\Delta t}X(\vec{t}) + \int_0^{\Delta t} d\tau e^{\mathfrak{A}(\Delta t-\tau)}\mathfrak{B}u(t+\tau)
\end{aligned}
\tag{3.6}
$$

であるから、

$$
\mathbb{A} = e^{\mathfrak{A}\Delta t} \tag{3.7}
$$

であることがわかる、離散系の制御では、:math:‘ t + Dleta t > tau > t‘で $u(\tau) = u(t)$ として良いから、

$$
\mathbb{B} = \int_0^{\Delta t} d\tau e^{\mathfrak{A}(\Delta t-\tau)}\mathfrak{B} \tag{3.8}
$$

となる。

C.4 digital filter と状態方程式

digital filter は、入力 x_n と出力 y_n が

$$
y_n = \Sigma_{k=0}^{M} c_k x_{n-k} + \Sigma_{j=1}^{N} d_j y_{n-j} \tag{3.9}
$$

で定義される。$N \geq M$ の場合はプロパーな関係となって、状態方程式で表現することが可能。

C.5 PID 制御

C.5.1 不完全微分

$$
\begin{gathered}
A = -\frac{1}{T_D}, B = 1, C = -\frac{1}{{T_D}^2}, D = \frac{1}{T_D} \\
\text{の時} \\
y = \frac{s}{T_D s+1}
\end{gathered}
\tag{3.10}
$$

これを不完全微分と呼ぶ。T_D に比べゆっくりとしている間は微分の近似になっている。

信号の微分はプロパーな応答として表現できない (状態方程式では表現できない）ことから、不完全微分を実現では使う必要がある[2]。

[2] PID 制御は少なくとも四つのパラメータ $k_p, k_i, k_d,$ andT_D を含んでいることになる。

C.5.2 オペアンプを使った微分／積分回路

https://detail-infomation.com/differentiation-circuit/

C.5.3 入力と出力

注釈: 入力と出力の関係は因果律を考慮すれば、一般性を失うことなく、

$$y(t) = \int_{-\infty}^{t} d\tau G(t,\tau)u(\tau) \tag{3.11}$$

と書けるだろう。時間の並進対称性を仮定すれば、

$$y(t) = \int_{-\infty}^{t} d\tau G(t-\tau)u(\tau) \tag{3.12}$$

として良い。

このカーネルの Laplace 変換が Proper であることは、カーネルに対して、どういう意味を持っているのか？

プロパーで無いということは、カーネルとの畳み込みが微分を含ということ.

関連図書

[A-1] Python control module. web page. URL: http://python-control.readthedocs.io/en/latest/intro.html#some-differences-from-matlab.

[A-2] インターフェース編集部編.“技術者のための *UNIX* 系 *OS* 入門、第八章 *RT-Linux* による倒立振り子の制御”. CQ 出版社、東京, 2000.

[A-3] 川谷亮治他.“倒立振り子キットマニュアル ”.

[A-4] L. Dalesio. Epics home page. URL: http://www.aps.anl.gov/epics/.

[A-5] 川谷亮治. 線形制御理論テキスト. URL: http://feedback.nagaokaut.ac.jp/cgi-bin/book.cgi.

[A-6] W. Stein et al. Sage mathematics software. URL: http://sagemath.org.

[A-7] 野庭建造、西村英和、平田光男. *MATLAB* による制御系設計. "東京電気大学出版局", 1998.

[A-8] 片山　徹.“線型システムの最適制御 *UTF 2013* デスクリプタシステム入門 *UTF 2013*”. 近代科学社, 東京, 1999. 状態方程式を拡張したデスクリプタシステムの最適制御についての教科書であるが、線型システムの LQ 最適フィードバックなどの解法についても詳しい。.

[A-9] 福井大学、川谷研究室. 様々な倒立振子系に対する安定化制御. web page. URL: http://mech.u-fukui.ac.jp/~Kawa-Lab/kenkyu/pend/pend.htm.