数理解析研究所講究録 1456 巻 2005 年 159-166

SCILAB へのべき級数の導入とその制御系設計への応用

北本卓也
T. KITAMOTO
山口大学
YAMAGUCHI UNIVERSITY*

1 序論

Mathematica や Maple 等の数式処理システムは近年かなり著名になり、これらの数式処理の特色を生かして工学上の問題を解決する研究も行われるようになってきた。しかしながら、これらの応用はまだ研究段階にとどまっており、現場レベルでの数式処理システムの応用はまだきわめて少ない状況にある。特に、制御系設計の分野においてはいまだに MATLAB 等の数値計算を基礎とするシステムが主役の座を占めており、数式処理システムの活躍の場は少ない。この理由としては、1つにはよく言われるように数式処理システムが柔軟性に欠けることが挙げられる。具体的には、工学上の問題が主に浮動小数点数を取り扱うのに対して、数式処理システムのアルゴリズムの多くが誤差を全く含まない整数、または有理数をベースにしているため、浮動小数点数を含む演算がそのままの形では行えないことである。これらの問題を解決するために、近年では数値数式融合演算に関する研究が盛んになってきている。

しかしながら、制御系設計の研究者の話を聞いてみると、数式処理システムが活用されない原因はその「数式処理の柔軟性の欠如」だけにあるのではなく、MATLAB のような制御系設計専門のパッケージが少ないことにもあることがわかる。具体的には、ロバスト制御系設計では今では基本的な道具となった LMI（線形行列不等式）や μ 解析など計算を行うパッケージが MATLAB にはあるが、*Mathematica* や Maple などの汎用数式処理システムにはそれに該当するものがない (*Mathematica* には Control System Professional 2 というパッケージがあるが、これは最新の制御系設計理論の結果を含んでいるとは言えず、LMI や μ 解析など計算は行えない)。また、制御系設計の研究者の多くは MATLAB を使うことに慣れており、*Mathematica* や Maple などの他の文法をもつシステムには不慣れなため近寄りづらいことも制御系設計の研究者を数式処理システムから遠ざける遠因であろう。

そこで本稿では、MATLAB とよく似た文法とほとんど同じ機能（パッケージ）を併せ持つ SCILAB に注目し、この SCILAB にべき級数演算を導入することにより制御系設計の研究者にとって近づきやすく、使いやすい "擬似" 数式処理システムを作成する（"擬似" とつけたのは、このシステムでは通常の数式処理システムとは数式の取り扱いが異なるからである。このシステムでは打ち切りべき級数のみを取り扱い、普通の意味での数式は取り扱うことが出来ない)。

なお、現在では MATLAB から数式処理システム Maple を直接使うためのパッケージ (Extended symbolic math toolbox) が開発され、以前に比べはるかに容易に MATLAB から Maple にアクセスできるようになっている。Extended symbolic math toolbox を用いた制御系の MATLAB ツールも開発されつつあり、この Extended symbolic math toolbox を用いて本稿のようなべき級数演算パッケージを Maple 上で実現

*kitamoto@yamaguchi-u.ac.jp

するということも技術的には可能である（このようなパッケージは制御系設計者から見たインターフェースがMATLABであり、制御系設計者から見たハードルが低いという点で本稿のパッケージと長所を共有するので有用ではないかと思われる）。

本稿は以下の構成をとる。まず、2章で制御系設計の研究者の標準的な道具となっているMATLABの概要を説明する。次に3章でMATLABの代替ソフトウェアとして今回取り上げるSCILABについて概説し、そのSCILABへのべき級数演算の導入について述べる。4章ではそのべき級数を係数とする多項式やべき級数を要素とする行列のSCILABへの導入について説明する。5章では、SCILABへ導入したべき級数演算のベンチマークテストを実施し、6章では、べき級数演算を活用した制御系の設計例を示す。最後に7章で結論を述べる。

2 MATLABについて

MATLABは行列を簡単に取り扱えるシステムとして昔から広く使われている有料のソフトウェアである。特に制御系設計の研究者、実務者の間では制御系設計の標準的なCADソフトウェアの役割を果たしている。行列の取り扱いが簡単になるような工夫がされており、行列演算が得意で（MATLABでは通常の数は1×1の行列として扱われる）、浮動小数点演算に基礎を置いている（整数型もあることはあるが、ほとんど使われない）。プログラム言語としてはインタプリタ型の言語であり、一言で言うと*Mathematica*から数式処理機能を除いて、制御系設計のための専用パッケージを付け加えたようなものである。プログラム言語の文法は手続き型言語の極めてオーソドックスなもので、プログラム言語として特に際立つものはなく、GNUのOctaveや本稿で取り挙げるSCILABなどのクローンがたくさん存在する。

MATLABが制御系設計の標準的なCADとして使われている一番大きな理由は専用パッケージが充実していることである。例えば最新の制御系設計の結果を用いた「ロバストコントロールパッケージ」、「LMI(行列線形不等式)」、「μ解析パッケージ」などがある。また、Simulabを用いれば、ブロック線図から簡単にシミュレーションを行えるようになっている。ただし、これらのパッケージは1つ1つが別売りされており、一揃いの制御系設計の機能をMATLABでそろえようとすると、合計金額は高価になってしまう。

3 SCILABについて

3.1 概要

SCILABはINRIA(フランス国立コンピュータ科学・制御研究所)で作成された高機能は行列演算システムであり、ソースコードも公開されているフリーソフトウェアである。MATLABとよく似た文法、ほぼ同じ行列演算機能(LAPACK等のパッケージを利用)を持っている。制御系設計パッケージが充実しており、MATLABの「ロバストコントロールパッケージ」「LMI(行列線形不等式)パッケージ」「μ解析パッケージ」と同機能のパッケージを持っている。また、MATLABのSimulabと同様にブロック線図からシミュレーションを行えるシミュレータSCICOSを含んでおり、制御系設計の面でMATLABと対等に渡り合えるだけの機能を持っている。

3.2 べき級数型の導入

3.2.1 SCILAB における新しいデータ型を導入する機能

SCILAB は C++ のように新しいデータ型やそれに対する演算を導入する機能を持っている。ここでいう新しいデータ型とは、C++ のクラスや C 言語の構造体と同様に複数のデータを組み合わせて 1 つのデータとしてみなしたものである。例えば、2 つの実数型データを組み合わせて複素数という新しい型を作ったりすることが出来る。

新しいデータ型は tlist という関数を用いて作られる。tlist は以下の形で呼び出される。

```
tlist([‘型の名前’,‘データ 1 の名前’,…,‘データ n の名前’], データ 1, データ 2, …, データ n)
```

tlist の一番初めの引数は文字列を要素とするベクトルであり、一番初めの要素が新しいデータ型の名前を表している（この名前はプログラムの作成者が自由に定めることが出来る）。ベクトルの 2 番目以下の要素は新しいデータ型が含むデータの名前である。引数の 2 番目以下の データ 1, …, データ n が実際に含まれるデータを示している。例えば、複素数 $1+2i$ は次のように定義できる。

```
tlist([‘complex’,‘real’,‘imag’], 1.0, 2.0)
```

上のように定義した複素数を変数 c に代入したとすると、その実部、虚部はそれぞれ c.real、c.imag でアクセスできる。

3.2.2 SCILAB へのべき級数の実現とその方針

前節で説明した機能を用いて、打ち切りべき級数を SCILAB に導入した（打ち切りべき級数のデータ型の名前は “ps” とした）。べき級数の実現方法を決める方針として、次のものを定めた。

(i) 多変数のべき級数を取り扱い、打ち切り次数は全次数を用いる。

(ii) 計算をなるべく効率的に行う。

(iii) 通常の数や行列となるべく同じように取り扱えるようにする。

(ii) の計算効率の向上の観点から、新しく作成するべき級数 “ps” 型では打ち切りべき級数を数式として取り扱わず、その係数のみをベクトルとして保持することにした。例えば、べき級数 $\alpha_0+\alpha_1 x+\alpha_2 y+\alpha_3 x^2+\alpha_4 xy+\alpha_5 y^2$ はベクトル $[\alpha_0\ \alpha_1\ \alpha_2\ \alpha_3\ \alpha_4\ \alpha_5]$ として保持される。このようにべき級数をベクトルとして保持した際の問題は、べき級数同士の乗除算である。というには、2 変数以上のべき級数を全次数で打ち切った場合、その乗算は不規則になるからである。例えば、x, y を変数とするべき級数を全次数 3 次で打ち切った場合、

$$\begin{aligned} p_1 &= \alpha_0+\alpha_1 x+\alpha_2 y+\alpha_3 x^2+\alpha_4 xy+\alpha_5 y^2 \\ p_2 &= \beta_0+\beta_1 x+\beta_2 y+\beta_3 x^2+\beta_4 xy+\beta_5 y^2 \end{aligned}$$

の加算の結果は

$$p_1+p_2 = \alpha_0+\beta_0+(\alpha_1+\beta_1)x+(\alpha_2+\beta_2)y+(\alpha_3+\beta_3)x^2+(\alpha_4+\beta_4)xy+(\alpha_5+\beta_5)y^2$$

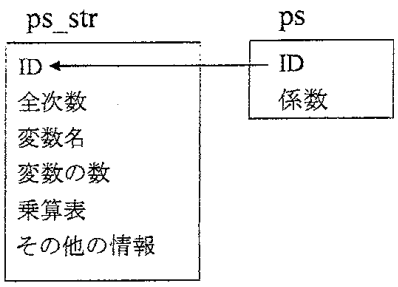

図 1: データ構造

となり、その係数の計算は規則的になるが、乗算の結果は

$$\begin{aligned} p_1 \times p_2 \quad = \quad & \alpha_0\beta_0 + (\alpha_0\beta_1 + \alpha_1\beta_0)x + (\alpha_2\beta_0 + \alpha_0\beta_2)y + (\alpha_0\beta_3 + \alpha_1\beta_1 + \alpha_3\beta_0)x^2 \\ & + (\alpha_0\beta_4 + \alpha_1\beta_2 + \alpha_2\beta_1 + \alpha_4\beta_0)xy + (\alpha_0\beta_5 + \alpha_2\beta_2 + \alpha_5\beta_0)y^2 \end{aligned}$$

となり、係数同士の演算がかなり不規則になってしまう。乗算の際の係数同士の演算規則は、ベクトルの各要素がべき級数のどの項に対応しているかがわかっていれば決められるが、べき級数の乗算が起こるたびに、この係数同士の演算規則を計算していたのでは (ii) の効率性に反する。よって係数同士の演算規則を乗算表としてあらかじめ計算しておくことにした。また、新しく定義するデータ型「べき級数」のデータとしてこの乗算表や変数名、変数の数などのべき級数の構造に関する情報を含めると、「べき級数」のデータ量が多くなりコピーを作るときの手間がかかるため、これも上の (ii) の効率性に反する (データのコピーは代入を行う際や、計算を行う際のテンポラリー変数の作成などで頻繁に使われるので、この操作が効率的であることは非常に重要である)。よって、新しいデータ型 "ps_str" を新たに作成し、こちらにべき級数の乗算表や変数名や変数の数などの基本情報とべき級数の種類を表す自然数 ID を保持し、べき級数の型 "ps" には、係数をベクトルで保存したものとこの ID のみを保持するようにした (図 1 を参照)。下は実際にべき級数を生成する例である。

```
id = ps_str(['x','y'],2);
p1 = ps(id,[1,2,3,-1,2,-2]);
p2 = ps(id,[3,4,-1,2,-1,1]);
```

1 行目の関数 ps_str() はべき級数の基本情報 (乗算表など) を生成し、それに対する ID を返す関数である。1 行目の命令で、x, y を変数とし打ち切り全次数を 2 としたべき級数の基本情報を生成し、それに対応する ID を変数 id に代入している。2 行目の関数 ps() は、その ID と係数のベクトルを元に実際にべき級数を生成する関数である。上の例では、べき級数 $1+2x+3y-x^2+2xy-2y^2$ を変数 p1 に代入している。3 行目の命令は、2 行目の命令と同様にべき級数 $3-4x-y+2x^2-xy+y^2$ を変数 p2 に代入している。

方針 (iii) については、次節で述べる「べき級数の演算子の導入」で対応した。

3.3 べき級数の演算子の導入

SCILAB では、C++ のオペレータオーバーローディングのように新しく導入した型に対する演算子を定義できる。SCILAB のシステムは新しく定義された型に対する演算が行われると、自動的にあらかじめ決

表 1: 演算子と対応する文字

演算	演算子	文字
加算	+	a
減算	-	s
乗算	*	m
除算	/	r
べき乗	^	p
共役転置	'	t
等式	==	o
等式の否定	<>	n
不等式	<	1
不等式	>	2
不等式	<=	3
不等式	>=	4
取り出し	()	e
代入	()	i

められた名前の関数を呼ぶようになっている。例を挙げると、変数 x と y の加算を定義するためには %<x の型名>_a_<y の型名> という名前の関数を定義する。ここで <x の型名>、<y の型名> はそれぞれ変数 x と y の型名 を表す。例えばべき級数 (ps 型) 同士の加算を定義するためには %ps_a_ps() という名前の関数を定義する。p1, p2 を ps 型 の変数とすると、p1+p2 という表現は自動的に %ps_a_ps(p1,p2) で置き換えられ、計算が実行される。ps 型はべき級数の種類を表す ID と係数のベクトルからなるので、%ps_a_ps() の定義は次のようにすればよい。

```
function c = %ps_a_ps(a,b)
if (a.id ~= b.id) then
  error("%ps_a_ps: ids are different!!");
end
c=ps(id,a.coef+b.coef);
endfunction
```

上の関数では、まず ID が一致するかどうかをチェックし、一致しなければエラーメッセージを出力して演算を終了する。ID が一致している場合は、与えられた引数と同じ ID をもち、与えられた引数の係数ベクトルの和を係数ベクトルに持つべき級数を新たに作成し、それを関数値として返している。

べき級数同士の減算、乗算、除算については %ps_a_ps() の a(加算を表す) をそれぞれ s(減算を表す)、m(乗算を表す)、r(除算を表す) で置き換えた名前の関数 %ps_s_ps()、%ps_m_ps()、%ps_r_ps() を同様に定義すればよい (乗算と除算の場合の計算は、前節で述べた乗算表を用いて行う)。このほか、べき乗 ^ (p を用いる) や等号演算子 == (o を用いる) などの演算子も定義できる。表 1 に演算子とそれに対応する文字の代表的なものの一覧表を示す。

関数 %ps_a_ps() はあくまでべき級数同士の加算を定義しているので、べき級数と数値の加算には使えない。よって p1 を ps 型とするとき、p1+1 というようなべき級数と数値の加算を行うためには %ps_a_s()

(数値は s 型である) を別に定める必要がある。このため、このようなべき級数と数値の演算を行う関数も定義した (ちなみに 1+p1 を演算を行う関数は%ps_a_s() でなく、%s_a_ps() である)。

4 多変数多項式、多項式行列の導入

SCILAB ではすでに poly 型という多項式を扱う型が存在するが、1 変数多項式に限られており、主変数の係数が数値であるものしか取り扱えない。1 変数多項式ではべき級数演算を生かすことが出来ないので、主変数の係数がべき級数である新しい型を定義し、multpoly 型と名づけた。また、SCILAB ではもちろん行列は取り扱えるが、要素が数値であるものしか取り扱えないので、要素がべき級数である新しい型を定義し、こちらは psmat 型と定義した。multpoly 型は、データとして主変数名と係数 (ps 型) のリストの 2 つをデータを保持することにした。psmat 型は、データとして行数、列数、要素 (ps 型) のリストの 3 つをデータとして保持する。

実際に multpoly 型のデータを作るときには、 multpoly(変数名, 係数のリスト) とする。例えば、次の命令は x を主変数とする 2 次の多項式 p0 + p1*x + p2*x^2 を作り、f に代入する。(ただし、p0、p1、p2 が ps 型の変数)

```
f = multpoly('x',list(p0,p1,p2))
```

また、psmat 型のデータを作るときには、 psmat(行数、列数、要素のリスト) とする。例えば、次の命令は、要素がべき級数である 2×2 の行列 $\begin{bmatrix} p11 & p12 \\ p21 & p22 \end{bmatrix}$ を作り、変数 m に代入する。(ただし、p11、p12、p21、p22 が ps 型の変数)

```
m = psmat(2,2,list(p11,p12,p21,p22))
```

multpoly 型に対する加減乗除等の演算子をべき級数の時と同様に定義し、記号的ニュートン法でべき級数根を計算する関数 multpoly_newton() を作成した。以下にそのソースコードを示す (説明のため、一列目に本来のソースコードには含まれていない行番号入れている)。

```
1:  function xn = multpoly_newton(p,x0)
2:  xn = x0;
3:  tdeg = ps_str_data(p.coef(1).id).tdeg;
4:  pd = p';
5:  i = 1;
6:  while (i-1)<tdeg
7:    i = 2*i;
8:    xn = xn-p(xn)/pd(xn);
9:  end;
10: endfunction
```

一番目の引数 p は multpoly 型の変数であり、multpoly_newton() はこの p のべき級数根を計算する。二番目の引数 x0 は数値であり、求めるべき級数根の定数部である。2 行目の xn = x0 で x0 をべき級数根 xn を x0 で初期化する。3 行目ではべき級数の全次数を tdeg に代入している。4 行目の pd = p' は p の

主係数による微分を pd に代入している（' は本来、行列の共役転置を取る演算子なのであるが、multpoly 型では、これを微分演算子として定義している)。5-9 行目でニュートン法の繰り返しによりべき級数根を計算している。8 行目の p(xn) は p の主変数に xn を代入したときの値 (結果は ps 型) を表している（この () は本来、行列やベクトルの要素の取り出しに使われる演算子であるが、ここでは主変数の値の代入に用いている)。

ソースコードからわかるように、通常の数式処理システムとほとんど同じ形でべき級数根を計算するプログラムが書けている。これは SCILAB へのべき級数の導入が自然な形で行えたことを示している。

psmat 型に対しても、multpoly 型と同様に、加減乗除等の演算子を定義し、逆行列のべき級数展開を計算する関数 psmat_inv() を作成した。以下にそのソースコードを示す (上と同様に、1 列目に行番号を入れている)。

```
1:  function pn = psmat_inv(a)
2:  if (a.row ~= a.col) then
3:    error(" Matrix is not square!!");
4:  end
5:  a0 = psmat_const(a);
6:  e = eye(a0);
7:  pn = inv(a0);
8:  tdeg = ps_str_data(a(1,1).id).tdeg;
9:  i = 1;
10: while (i-1)<tdeg
11:   i = 2*i;
12:   pn = pn+pn*(e-a*pn);
13: end;
14: endfunction
```

上のソースコードの説明は省略するが、これも通常の数式処理システムとほとんど同じ形でプログラムが書けている。

5 結論

SCILAB は制御系設計の標準的ツールとなっている MATLAB に匹敵する機能、パッケージを持ったオープンソースのソフトウェアである。SCILAB には、C++ のクラスと同様に新しいデータ型とそれに対する演算を定義する機能がある。本稿では、この機能を用いて SCILAB にべき級数演算パッケージを導入した。このパッケージの特徴は

(a) 多変数のべき級数（全次数で打ち切り）が自然な形で扱える。

(b) べき級数の乗算表をあらかじめ計算しておくことや、べき級数の構造に関するデータを別のデータ型に定義しておく等の工夫により、SCILAB 上のプログラムとしては高速に計算を行う。

(c) べき級数を通常の数と同様に自然な形で扱うことが出来る

である。今後の課題はベンチマークテストによるシステムの効率性の検証と、実際の制御系設計へ適用が挙げられる。

参 考 文 献

[1] Scilab Group: Scilab 入門, ftp://ftp.inria.fr/INRIA/Scilab/contrib/SCIJAPANESE/intro-jp.pdf

[2] Kitamoto, T.: Approximate eigenvalues, eigenvectors and inverse of a matrix with polynomial entries, *Jpn. J. Indus. Appl. Math.*, **11**(1), 1994, 73-85.

[3] 北本: 近似固有値、固有ベクトルとその最適制御への応用, 電子情報通信学会論文誌, **J78-A**(4), 1995, 531–534.

[4] 北本: 近似代数を用いた H_2 最適制御系の解析と設計, 電子情報通信学会論文誌, **J81-A**(2), 1998, 289–292.

[5] 北本: 代数的 Riccati 方程式のべき級数解の計算法について, 電子情報通信学会論文誌, **J81-A**(3), 1998, 445–447.

[6] 北本: 近似固有値の計算法について, 電子情報通信学会論文誌, **J81-A**(4), 1998, 803–806.

[7] 北本：近似逆行列の高次収束への拡張について, 電子情報通信学会論文誌, **J84-A**(2), 2001, 243–245.

[8] Kitamoto, T.: On computation of approximate eigenvalues and eigenvectors, *IEICE Trans. Fundamentals*, **E85-A**(3), 2002, 664–675.

[9] Kitamoto, T.: Computation of the peak of time response in the form of formal power series, *IEICE Trans. Fundamentals*, **E86-A**(12), 2003, 3240–3250.

[10] Kung, H.T., Traub, J.F, All algebraci function can be computed fast, *J. ACM*, **25**, 1978, 245–260.

[11] Lipson, J.D., Newton's method: A great algebraic algorithm, *Proceedings of ACM Symposium on Symbolic and Algebraic Computations*, 1976, 260–270.

[12] Yokoyama, K., Takeshima, T.: On Hensel construction of eigenvalues and eigenvectors of matrices with polynomial entries, *Proceedings of the ISSAC '93*, ACM PRESS, 1993, 218-224.