数値処理と数式処理を融合した 制御系 CAD 言語 MATX

Computer Aided Control System Design Language MATX with Numerical Computation and Symbolic Manipulation

古賀 雅伸 * ・古田 勝久 *

Masanobu KOGA* and Katsuhisa FURUTA*

*Tokyo Institute of Technology, Meguro-ku, Tokyo

* 東京工業大学 東京都目黒区大岡山 2-12-1

邦 文 要 旨

本研究では、ベクトルや行列演算の記述性に優れ、数値処理と数式処理を統一的に扱うことができる、制御系 CAD 言語 MATX の開発を行なった。数値処理と数式処理の長所と短所を考察することにより、それぞれの短所を補い長所を生かす数値 – 数式混合処理が可能となった。制御系の解析・設計において、１変数の多項式や有理関数そしてそれらを成分として持つ多項式行列や有理関数行列が、重要な役割を果たすことに注目して、これらの数式処理と数値行列の演算などの数値処理とを融合した。これにより、システム理論に必須である数値行列の様々な演算や、多項式、有理関数、等の数式の処理を理論中の表記とほとんど同じ形式で記述でき、制御系の解析・設計だけでなく、システム理論の新しい設計アルゴリズムの開発や検証などの支援にも役立つ。

英 文 要 旨

This paper describes a high-performance computer aided control system design language MATX which integrates numerical analysis, matrix computation, and symbolic manipulation in an easy way to use. One-variable polynomials, rational-functions, and symbolic matrices comprised of those elements are essential to the study of system theory. We take advantage of this fact and combine numerical computation and one-variable symbolic manipulation facility. Since MATX allows the definition and manipulation of these objects in a natural symbolic fashion, problem solutions are expressed almost exactly as they are written mathematically. Therefor, the user can solve a lot of control system design problems in much shorter time than describing programs in other programming languages. MATX will play an important role in developing and examining new design algorithms in system theory.

Key Words : CAD system, symbolic manipulation, numerical computation, matrix computation

1. はじめに

制御工学およびシステム工学での計算機援用設計 (CAD) システムの歴史は長く、BASIC 言語で書かれた比較的小規模なものから FORTRAN 等で書かれたかなり大規模なシステム [1),2)] まで種々開発され実用に供している。初期の CAD システム [3),4)] は、数値計算によるものであり、制御系のパラメータが記号のままで与えられた場合は取り扱うことができなかった。その後、制御系の解析・設計の際にはパラメータを記号のままで処理した方が都合の良い場合があることが指摘され [5),6)]、記号処理に適した CAD システムが数式処理言語を用いて開発された。しかしながら、数式処理言語は様々な数式の代数的計算には有効であるが、一般に数値計算の効率が悪い。そのため、REDUCE や muSIMP のような数式処理言語と C や FORTRAN のような数値処理言語を併用することによって、数式・数値の混合計算が可能な、制御系の計算環境を構築する試みがなされるようになった [7),8)]。

制御系の設計において、外乱やプラントパラメータの変化に対する閉ループのシミュレーションを行なうことは重要であるので、制御系設計に計算機を使用することはシミュレーションから始まった。代表的なものに、非線形システムのシミュレーションを行なうために開発された SIMNON[9)] がある。残念ながら、SIMNON は行列演算を効率良く記述することができなかった。制御系の解析・設計におけるベクトルや行列演算を簡単に行ないたいという要望から、行列を基本データ構造とする言語 MATLAB[10)] や MATRIXx[11)] が生まれた。

MATLAB は、 LINPACK や EISPACK 等の信頼性のある行列計算アルゴリズムを元に構築された行列演算言語であり、紙に書いた行列演算をほとんどそのままプログラムとして使用できるという記述性の良さから、近年多く使用されるようになった。しかしながら、MATLAB はデータの型を「区別しない」言語で、唯一のデータ型は行列であり、多項式や有理関数などの他の型のデータの演算や操作をするには、特別な演算子を使うか関数を呼び出さなければならない。このため、メモリ効率が悪く、スカラ値のループ計算が遅いだけでなく、多項式や有理関数を伴う計算のプログラムの可読性が悪い。

本研究では、ベクトルや行列演算の記述性に優れ、数値処理と数式処理を統一的に扱うことができる、制御系 CAD 言語 MATX の開発を行なった。これにより、システム理論に必須である数値行列の様々な演算や、多項式、有理関数、多項式行列、有理関数行列等の数式の処理を理論中の表記とほとんど同じ形式で記述でき、制御系の解析・設計だけでなく、システム理論の新しい設計アルゴリズムの開発や検証などの支援にも役立つと考えられる。本論文では、 MATX の数値処理と数式処理の融合について記述するが、 MATX のシステム環境や内部構造については、文献 12) で説明される。

2. 数値処理と数式処理

2.1. 数値処理

制御系の CAD が開発されるにつれ、制御系設計アルゴリズムの計算を数値的に安定に行なうことの重要性が認識され、多くの数値的に安定なアルゴリズムが提案された [13]。特に、最適レギュレータやオブザーバなどの線形制御理論を中心とする制御系の解析・設計に必須である LU 分解、 QR 分解、 QZ 分解、固有値問題、一般固有値問題、等の行列演算を数値的に安定に行なうアルゴリズムをまとめたものに LINPACK[15] や EISPACK[15),16)] がある。これらのパッケージは、FORTRAN で記述されており、簡単に入手できるので、多くの制御系 CAD の内部で利用されている [1),2)]。数値処理言語で書かれた CAD には、以下の特徴がある。

- 数値計算が高速である。
- 複雑な処理に対する計算法が十分に発達している。(固有値問題に対する数値的に安定なアルゴリズムが存在する。)
- パラメータを記号のまま扱う問題は苦手である。

2.2. 数式処理

いっぽう、数式処理を制御系の設計に利用する試みが始まったのは、１０年ほど前である [17),18)]。ほとんどの数式処理による制御系 CAD は、汎用の数式処理言語の上に、設計に必要な関数を付け加えることによって構築された [5),6)]。

汎用の数式処理システムのうち古くから使われているものには、 MACSYMA, REDUCE, CAMAL, SCRATCHPAD がある。その後、 μ-MATH , SMP , MAPLE 等が開発され、近年になって Mathematica[19] が Wolfram Research,Inc. によって開発された。数式処理言語で書かれた CAD には、以下の特徴がある。

- 数値計算による誤差がない。
- 特定の数値に依存しない本質的な性質を議論できる。(処理結果のパラメータに数値を代入すれば、一度に多くの数値解が得られる。)
- 中間結果が爆発することがある。
- 複雑な処理に対する計算法が十分に発達していない。(固有値問題が不得意である。)
- 数値計算の効率が悪い。

2.3. 数値 – 数式混合処理

数値処理と数式処理の両者の短所を補い長所を生かした数値 – 数式ハイブリド CAD を開発するには 2 つの方法がある。１つは、数値処理言語と数式処理言語の両方を利用して数値処理システムと数式処理システムを開発し、両者のインターフェースを作成する方法であり、もう１つは、数式処理と数値処理を融合させた言語を設計し、その言語を用いて数値 – 数式システムを開発する方法である。前者の場合、開発時間は短いが、数値処理系と数式処理系の間でデータの転送が必要であるので、データ転送の回数や量が多くなるとシステムの処理速度が遅くなる。後者の場合、数式処理と数値処理を対等に混用でき、システムは単一プロセスからなっているので、データ転送の必要がないので、データ量の増加による処理速度の低下はおこりにくい。

一般に、数値処理と数式処理を融合させた言語を開発するには時間がかかる。しかし、システム工学における代表的なシステム記述方法である伝達関数や伝達関数行列は、記号 s や z を主変数とした１変数多項式や１変数有理関数から構成されるので、１変数の数式のみを扱う数式処理を数値処理に融合させた言語は、システム制御工学において利用価値が高いと考えられる。そして、そのような言語は、開発時間が短く、数式処理も高速であ

る。そこで、本研究では、数値処理と 1 変数の数式処理を融合させた制御系 CAD 言語を設計して、それを用いて制御系 CAD を開発する。

3. プログラミング言語 MATX

MATX の行列の演算や成分の操作法の重要なアイデアは、Cleve Moler が 1982 年に開発した MATLAB 言語に由来する [10),20)]。MATX は、MATLAB といくつかの特徴的性質を共有しているが、けっして MATLAB の一方言などではない。MATLAB は、データの型を「区別しない」言語である。唯一のデータ型は行列であり、他の型のデータの演算や操作をするには、特別な演算子を使うか関数を呼び出さなければならない。

一方、MATX はデータの型を「区別した」言語である。データ型には、整数、実数、複素数、文字列、多項式、有理関数、行列、配列、リストがあり、型宣言名は、Integer, Real, Complex, String, Polynomial, Rational, Matrix, Array, List である。また、行列、配列、多項式、有理関数、リストは次数指定を必要としない。型の異なるデータの演算においては、**Fig.1** に示す “自動型変換規則” にしたがって、型変換が行なわれる。MATX の文法の根幹となる制御の流れや関数やプログラム構造は、C 言語から受け継いでいる。C 言語は、PDP-11 上で Dennis M.Ritchie によって開発されたものである [21)]。MATX はすべて C 言語で記述されており、特に、構文解析部は構文解析部生成言語として有名な **yacc** を用いて作成されている。

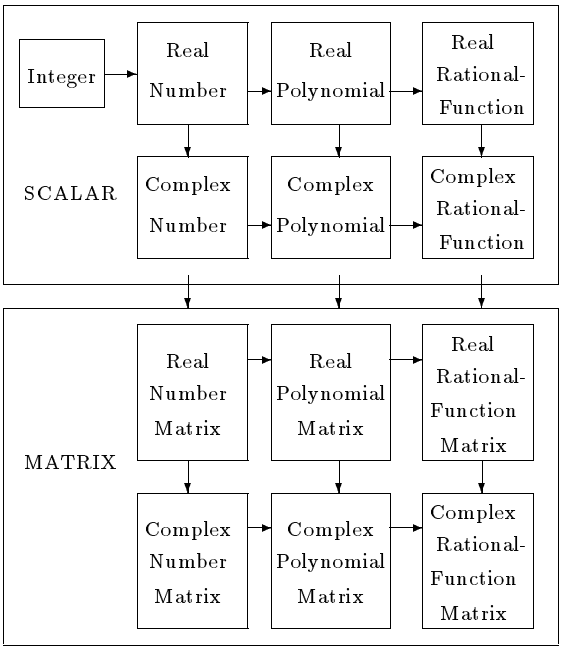

Fig. 1 The Automatic Class Conversion Rule

インタプリタ matx は、**Fig.2** に示すようにクラスライブラリ **MaTX-Lib** を使って作成されている。MaTX-Lib は、複素数クラス、多項式クラス、有理関数クラス、行列クラス、リストクラスの C 言語のクラス・ライブラリの集合である。特に、行列クラス・ライブラリは、高レベル・ライブラリと低レベル・ライブラリから構成されている。

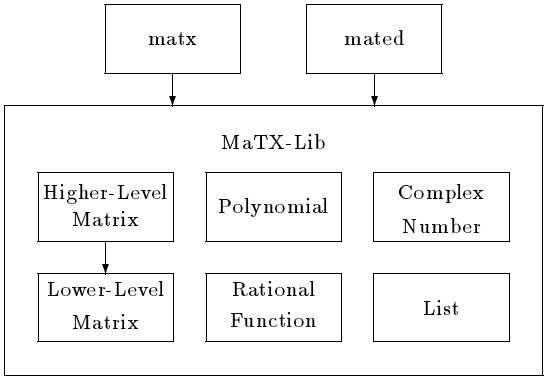

Fig. 2 The Architecture of MATX

低レベル・ライブラリは、約 300 個の関数から構成され、データの高速アクセスを目的とし、型のチェック等を行なわないため、使用時には注意が必要である。データの演算とメモリ効率を高めるための最適化もこのライブラリで行なわれている。高レベル・ライブラリは、低レベル・ライブラリの上に作られていて、約 200 個の関数から構成されている。高レベル・ライブラリの目的は、データの抽象化を行なうユーザ・インタフェースを提供することである。データの保存に必要なメモリ領域を管理し、データの属性に応じて適当な低レベル・ライブラリの関数を呼び出す。

4. 行列処理の実現

MATX は、行列演算を簡単に行なえるように設計されている。

4.1. 行列の入力方法

小さな行列の最も簡単な記述方法は、行列の成分をスペース、タブ、改行、カンマ “,” のいずれかで区切って書き並べ、四角ブラケット “[” と “]” で囲むことである。四角ブラケット “[” と “]” は、行の始まりと終りを示すためにも使われる。各成分の間には、任意個のスペースと改行を挿入することができるので、プログラムが読み易くなるように記述することができる。例えば、 3×3 の実数行列は、次のように

```
A = [[1 2 3][4 5 6][7 8 9]];
```

あるいは、もっと自然な様式

```
A = [[1 2 3]
     [4 5 6]
     [7 8 9]];
```

```
 Name :  [ A ]
 Size :  [  50, 10 ]    Filename : A

           (  3)         (  4)         (  5)         (  6)
( 18)  -1.957854E-01  9.554574E-02 -1.957854E-01  9.554574E-02
( 19)  -1.953817E-01  9.751419E-02 -1.953817E-01  9.751419E-02
( 20)  -1.949779E-01  9.948264E-02 -1.949779E-01  9.948264E-02
( 21)  -1.945741E-01  1.014511E-01 -1.945741E-01  1.014511E-01
( 22)  -2.084542E-01  8.378548E-02 -2.084542E-01  8.378548E-02
( 23)  -2.080504E-01  8.575394E-02 -2.080504E-01  8.575394E-02
( 24)  -2.076466E-01  8.772239E-02 -2.076466E-01  8.772239E-02
( 25)  -2.072429E-01  8.969085E-02 -2.072429E-01  8.969085E-02
( 26)  -2.068391E-01  9.165930E-02 -2.068391E-01  9.165930E-02
( 27)  -2.064353E-01  9.362776E-02 -2.064353E-01  9.362776E-02
( 28)  -2.060315E-01  9.559621E-02 -2.060315E-01  9.559621E-02
( 29)  -2.056277E-01  9.756466E-02 -2.056277E-01  9.756466E-02
( 30)  -2.052239E-01  9.953312E-02 -2.052239E-01  9.953312E-02
( 31)  -2.048201E-01  1.015015E-01 -2.048201E-01  1.015015E-01
( 32)  -2.044164E-01  1.034700E-01 -2.044164E-01  1.034700E-01
```

Fig. 3　Matrix Editor (Real Matrix)

で記述でき、結果は

```
=== [A] : (  3,  3) ===
           (  1)         (  2)         (  3)
(  1)  1.000000E+00  2.000000E+00  3.000000E+00
(  2)  4.000000E+00  5.000000E+00  6.000000E+00
(  3)  7.000000E+00  8.000000E+00  9.000000E+00
```

となる。

4.2.　行列エディタ

行列の大きさが大きくなればなるほど、行列の入力・編集作業は面倒になる。入力中にちょっとした間違いを犯しても、もう一度最初からやり直さなければならない。そういった面倒な作業からユーザを解放するために行列エディタ (mated) が開発された。ユーザがアルファベットや特殊文字などの不適当な文字をタイプしても、行列エディタが常にタイプされた文字を判断しているので、問題は生じない。行列エディタの特徴を以下に示す。

1. 任意の大きさの行列を入力・編集でき、大きさを任意に変更できる。
2. 大きな行列を編集する時、画面が上下左右にスクロールする。
3. ウィンドウシステムやグラフィック端末を必要とせず、 curses ライブラリがサポートするキャラクタ端末で使用可能。
4. 実行列だけでなく、複素行列も入力・編集可能。ただし、数式行列の編集機能は未開発。
5. 単位行列の入力、零行列へのリセット、複素行列への変換、転置行列への変換が容易。
6. 行削除、列削除、ペースト、領域の複写が可能。
7. ファイルへの書き込みと、ファイルからの読み込みができる。
8. Emacs 風と vi 風の両方のキー配置が使用可能。

Fig.3 は、行列エディタを使って、 (50×10) の実行列の (18 行 ~ 32 行)(3 列 ~ 6 列) を入力している様子を示している。

4.3.　行列演算と配列演算

行列演算は、 MATX にとって基本的なものであり、(計算機の文字集合の制限はあるが、) 教科書や論文中で書かれる数学的表現とほとんど同じ形で記述することができる。特別な文字ティルデ “~” とプライム “’” は、逆行列と転置行列をそれぞれ意味する。そして、シャープ “#” は、複素共役転置を表し、行列 A が実行列 (実数行列、実多項式行列、実有理関数行列) のとき、 A# は転置行列を意味する。行列の和と差は、 “+” と “-” によって表現され、行列の積は、 “*” によって表現される。 MATX には、行列の割算の演算子が 2 つあり、 “/” と “\” である。もし、行列 $\mathbf{A}$ が正則な正方行列ならば、 “$\mathbf{A}\backslash\mathbf{B}$” と “$\mathbf{B}/\mathbf{A}$” は、形式的には行列 A の逆行列を行列 B の左からと右から乗じることに対応する。 “$\mathbf{A}\hat{}\mathrm{p}$” という表現は、行列 A の p 乗を表し、行列 A が正方で p が 0 以上の整数のとき定義される。

MATX は、行列と配列を区別する。行列と配列のデータ構造は全く一致していて、複数のデータを 1 次元的または 2 次元的に保存するという性質は変わらない。行列の演算では、代数的な行列演算が行なわれるのに対して、配列の演算では、配列の各成分に対して演算が行なわれ、演算結果は各成分の演算結果を成分とする配列になる。計算の途中で一時的に行列を配列として演算を行なうための便利な配列演算子 `.*`, `./`, `.\`, `.^` がある。クラス変換関数 `Matrix()` と `Array()` によって、行列と配列は相互に変換できる。便利な添字記法を使えば、列や行や各成分や部分行列を簡単に参照することができる。

4.4. 行列関数

LU 分解、Schur 分解、QR 分解、QZ 分解、特異値分解、固有値問題、一般固有値問題、等の行列演算を数値的に安定に行なうアルゴリズムは、最適レギュレータやオブザーバなどの線形制御理論を中心とする制御系の解析・設計に必須である。MATX が備えている関数群の一部を Table 1 に示す。アルゴリズムや機能についての詳しい説明は割愛するが各関数は大雑把にいって備考欄に示す機能を持っている。ただし、`A` と `B` は行列であり、`p` と `m` と `n` は、整数である。

Table 1. Matrix Function

Function Name	Remark
Rows(A)	Row number
Cols(A)	Column number
rank(A)	Rank
max(A)	Maximum element
min(A)	Minimum element
maxsing(A)	Maximum singular value
minsing(A)	Minimum singular value
det(A)	Determinant
sum(A)	Sum of all elements
mean(A)	Mean of all elements
trace(A)	Trace
frobnorm(A)	Frobenius norm
infnorm(A)	Infinity norm
norm(A, p)	p = 1: 1-norm p = 2: 2-norm
vectorprod(A, B)	Vector product
exp(A)	Exponential function
sqrt(A)	Square root function
vec2diag(A)	Column-vector to diagonal matrix
diag2vec(A)	Diagonal matrix to Column-vector
kernel(A)	Kernel (Null space)
conj(A)	Complex conjugate
rand(m, n)	m x n Uniformly random matrix
size(A)	Row number and column number
eig(A)	Eigen value problem
eig(A, B)	Generalized eigen value problem
svd(A)	Singular value decomposition
lu(A)	LU-decomposition
hess(A)	Hessenberg-decomposition
schur(A)	Schur-decomposition
qr(A)	QR-decomposition
qz(A, B)	QZ-decomposition

5. 数式処理の実現

多項式、有理関数、多項式行列、有理関数行列は、ボード線図、ナイキスト線図、等の周波数領域でのシステムの解析・設計において重要である。近年、既約分解表現あるいは多項式代数法を用いた制御理論が活発に研究されている。多項式代数法による制御系の解析・設計技法は伝達関数の既約分解からユニラテラル、バイラテラル、ベズー方程式などで代表されるいくつかの代数方程式に帰着され、システム制御工学における数式処理の重要性が増している。MATX は、数式オブジェクト (多項式、有理関数、それらを成分にもつ行列) の定義や操作を自然な様式で記述できるように設計されていて、数式オブジェクトは実数及び複素数の係数を持つことができる。

5.1. 多項式と有理関数

MATX の数式オブジェクトは 1 変数であるので、多項式の係数は昇べきの順に横ベクトルとして保存され、有理関数は分子多項式と分母多項式の対として保存される。多項式を記述するには、多項式変数を定義し、その変数を利用して数式を記述する。例えば、

```
s = $;
A = 4 * s^2 + 5 * s + 6;
```

の結果、

```
A = 4 s^2 + 5 s + 6
```

となる。“`s = $;`” は多項式変数の定義であり、出力に用いられる多項式変数は、常に文字 “`s`” である。クラス変換関数 `Polynomial()` と `Matrix()` によって横ベクトルと多項式を相互に変換することができる。多項式の係数は、横ベクトルの成分と昇べきの順に一致する。例えば、

```
B = Polynomial([6 5 4]);
```

の結果、多項式 `B` は、多項式 `A` と一致する。多項式の係数を参照するには、多項式の名前の次に次数を丸括弧 “`(`” と “`)`” で囲む。

```
A(2) = 1.0; A(1) = 2.0; A(0) = 3.0;
```

の結果、

```
A = s^2 + 2 s + 3
```

となる。一方、有理関数は一対の多項式の割算として記述し、

```
s = $;
C = (s + 1)/(4*s^2 + 5*s + 6);
```

の結果

```
          s + 1
C = ---------------
    4 s^2 + 5 s + 6
```

となる。有理関数の分子と分母は、関数 Nu() と De() によって取り出すことができる。例えば、 Nu(C) は 1 次の多項式 “s + 1” を、 De(C) は 2 次の多項式 “4s^2 + 5 s + 6” をそれぞれ取り出す。 MATX では、多項式と有理関数を、実数や複素数のように行列の成分として使うことができる。これは、 MATX の便利な特徴の 1 つで、システム制御工学の研究に威力を発揮する。多項式行列と有理関数行列について次に述べる。

5.2. 数式行列

数式行列、すなわち多項式行列や有理関数行列の操作は実数行列や複素行列といった数値行列の操作と同じである。多項式行列を記述する便利な方法には、少なくとも 2 つある。 1 つは、

```
s = $;
A = [[1 2][3 4]]*s^2 + [[5 6][7 8]]*s
    + [[9 10][11 12]];
```

であり、もう 1 つは

```
s = $;
A = [[  s^2+5*s+9 , 2*s^2+6*s+10]
     [3*s^2+7*s+11, 4*s^2+8*s+12]];
```

で、どちらも同じ多項式行列を作る。前者は、行列を多項式の係数として用い、後者は、 4 つの多項式を陽に行列の成分として使っている。

有理関数行列を記述するには、陽に有理関数を行列の成分として用いる。例えば、

```
s = $;
A = [[(s+1)/(s+2) , (s+3)/(s+4)]
     [(s+5)/(s+6) , (s+7)/(s+8)]];
```

の結果、

```
=== [A] : (  2,  2) ===
      [     (  1)     ]  [     (  2)     ]
            s + 1             s + 3
(  1) ----------------   ----------------
            s + 2             s + 4

            s + 5             s + 7
(  2) ----------------   ----------------
            s + 6             s + 8
```

となる。また、配列オペレータを使うことによって、有理関数行列を簡単に記述することができ、

```
s = $;
X = [[1 3][5 7]];  Y = [[2 4][6 8]];
B = (ONE(2)*s + X) ./ (ONE(2)*s + Y);
```

によって、有理関数行列 B は、 A と一致する。ただし、記号 “./” は、行列の成分毎の割算を行なう配列オペレータを意味し、 ONE(2) は、全ての成分が 1 である 2×2 の行列である。

5.3. 数式の微分と積分

関数 “derivative()” と “integral()” を用いると、数式の微分と積分を求めることができる。第 2 引数に求める導関数 (derivative) あるいは不定積分 (indefinite integral) の階数を指定するが、階数を省略すると 1 階の導関数または不定積分を求める。例えば、

```
A = 12*s^2 + 18*s + 10;
M = [A A];
DM = derivative(M);
IM = integral(M);
```

によって、

```
 DM = [24s + 18, 24s + 18]
 IM = [4s^3 + 9s^2 + 10s, 4s^3 + 9s^2 + 10s]
```

の結果を得る。

5.4. 数式の評価

関数 “eval()” を用いて、数式の変数にいろいろな型の値を代入した結果を求めることができる。配列は、多項式行列の各成分の変数に代入されるので、評価結果はブロック行列になる。例えば、

```
A = 3*s^2 + 4*s + 5;
B = eval(A, 2.0);          C = eval(A, s+1);
D = eval(A, (s+1)/(s+2)); F = eval(A, [1:4]);
```

の結果、

```
B = 25
C = 3 s^2 + 10 s + 12

      12 s^3 + 62 s^2 + 107 s + 62
D = ----------------------------
         s^3 + 6 s^2 + 12 s + 8

F = [12  25  44  69]
```

となる。

6. MATX の実行例

ここでは、いくつかの例題プログラムを紹介する。

6.1. 連続系リカッティ方程式

ハミルトン行列の固有値問題を解くことによって[22]、連続系の線形システムのリカッティ方程式の解を求めるプログラムは、以下のように記述できる。6 行で、ハミルトン行列 H を作り、9 行で、行列 H の固有ベクトルの計算する。eigvec() は、固有ベクトルを固有値の実部の大きさ順に並べて返す関数である。10–11 行で、行列 vec からピース行列 (Piece Matrix)A を使って安定な固有値に対応する固有ベクトルを取り出す。ピース行列は、行列の配列の大きさの単位を指定するために使われる。

```
   Func Matrix CRiccati(A, B, Q, R)
     Matrix A, B, Q, R;
   {
     Matrix H, U, V, vec, P;
05:
     H = [[  A  , -B*R~*B']
          [ -Q  ,    -A'  ]];

     vec = eigvec(H);
10:  U = vec(0, 1, A);
     V = vec(1, 1, A);
     P = V * U~;

     return P;
15:}
```

List 1: Continuous-Time Riccati Equation

6.2. 線形システムの伝達関数

線形システム

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned}$$

の伝達関数行列

$$\begin{aligned} G(s) &= C(sI - A)^{-1}B \\ &= C\frac{\Gamma_{n-1}s^{n-1} + \Gamma_{n-2}s^{n-2} + \cdots + \Gamma_0}{s^n + \alpha_{n-1}s^{n-1} + \alpha_{n-2}s^{n-2} + \cdots + \alpha_0}B \end{aligned}$$

をファディーブのアルゴリズムにより求めるプログラムを以下に示す。8 行で多項式変数の宣言し、9 行でシステムの次数 n を決定。11 行で、“sI - A” の余因子行列の係数行列 Γ_i を保存するための大きな行列 Gamma を用意し、12–13 行で特性方程式 ch をモニック多項式として定義。15–20 行で、Γ_i と多項式の係数をファディーブのアルゴリズムを使って計算し、22–26 行で、それらを使って伝達関数行列を計算する。

```
   Func Matrix Faddeev(A, B, C)
     Matrix A, B, C;
   {
     Matrix Gamma, N;
05:  Polynomial s, ch;
     Integer i, n;

     s = $;
     n = Rows(A);
10:
     Gamma = Z(n,A);
     ch = Polynomial(Z(1,n+1));
     ch(n) = 1.0;

15:  Gamma(n-1,A) = I(n);
     ch(n-1) = - trace(A);
     for (i = n-2; i >= 0; i--) {
       Gamma(i,A) = A*Gamma(i+1,A) + ch(i+1)*I(n);
       ch(i) = - trace(A*Gamma(i,A)) / (n - i);
20:  }

     N = Z(Rows(C), Cols(B));
     for (i = 0; i < n; i++) {
       N = N + C*Gamma(i,A)*B * s^i;
25:  }
     return N / ch;
   }
```

List 2: Transfer Function

6.3. ボードとナイキスト周波数応答

与えられた線形システムのボード線図とナイキスト線図を描くためのデータを計算するプログラムを以下に示す。生成したデータは、グラフィック描画・編集環境 XPLOT[23] を MATX 内から、呼び出すことによってグラフィック表示することができる。XPLOT は、X ウィンドウ上で開発されたソフトウェアで、複数のデータを複数の画面に同時に表示でき、マウスを使った対話型処理だけでなくスクリプトによる一括型処理も可能である。

9 行で、3 次の実有理関数 G を入力し、10 行で、周波数領域を対数空間ベクトル w で指定。12–13 行で、複数の戻り値を返す関数 Bode() と Nyquist() を呼び、15–16 行で、データをファイルに保存。26 行と 40 行で、虚数単位 j の定義し、27 行と 41 行で、有理関数配列の評価。28 行と 29 行で、配列演算を行ない、30 行と 42 行で、2 つの成分を持つリストを返す。

```
   Func void main()
   {
     Polynomial s;
     Rational G;
05:  Array w, ga, ph, re, im;
     List Bode(), Nyquist();

     s = $;
     G = (s^2+3*s+4) / (s^3+5*s^2+6*s+7);
10:  w = logspace(0.01, 100.0);

     {ga, ph} = Bode(G, w);
     {re, im} = Nyquist(G, w);

15:  print [[w][re][im]] >> "nyquist.mat";
     print [[w][ga][ph]] >> "bode.mat";
   }
```

```
   Func List Bode(G, w)
20:  Rational G;
     Array w;
   {
     Complex j;
     Array data, ga, ph;
25:
     j = (0,1);
     data = eval(G, j*w);
     ga = 20.0 * log10(abs(data));
     ph = arg(data) * 180.0 / PI;
30:  return {ga, ph};
   }

   Func List Nyquist(G, w)
     Rational G;
35:  Array w;
   {
     Complex j;
     Array data;

40:  j = (0,1);
     data = eval(G, j*w);
     return {Re(data), Im(data)};
   }
```

List 3: Bode and Nyquist Frequency Response

7. おわりに

以上、現在著者らが開発を行なっている制御系 CAD 言語 MATX について述べてきた。本システムは 6 章の実行例でも示すように、制御系の解析・設計に必要な数式処理と数値処理 (特に行列計算) を統一的に扱うことができる。したがって、数値演算と数式処理の両者の長所を生かしたいわゆる数値 – 数式ハイブリッド CAD の開発が可能である。

制御系の解析・設計において、1 変数の多項式や有理関数そしてそれらを成分として持つ多項式行列や有理関数行列が、重要な役割を果たすことに注目して、これらの数式処理と数値行列の演算などの数値処理とを融合した。本システムでは、複数の変数を持つ数式の処理を行なうことができないので、そのような必要が生じた時は、 Mathematica や Maple のような汎用の数式処理システムを利用しなければならない。しかし、汎用の数式処理システムと MATX の結合は難しい問題ではない。

ところで、行列演算を簡単に行なうために開発された言語に、 MATLAB[10] や MATRIXx[11] があるが、どちらも行列を基本データ構造とし、データの型を「区別しない」言語なので、多項式や有理関数を伴う計算のプログラムの可読性が悪い。そして、これらは汎用の数式処理システムと同じくインタプリタであるので、 FORTRAN や C で記述されたプログラムより処理速度が遅い。我々が開発している MATX もインタプリタ上で処理すると速度は遅いが、コンパイルして実行型プログラムを作成して実行することによって、高速な処理が可能となる。このことは、別の稿で報告する予定である。

参 考 文 献

[1] Katsuhisa Furuta: Computer Aided Design Program for Linear Multivariable Control Systems, IFAC CAD Symposium, 267/272 (1980)

[2] 古田 勝久: 制御系 CAD の現状と動向, 電気学会雑誌, 105/112 (1986)

[3] Dean K. Frederick and Russell P. Kraft: Computer-Aided Control System Analysis and Design Using Interactive Computer Graphics, Control Systems Magazine, 19/23 (1982)

[4] Mituhiko Araki: Industrial Applications in Japan of Computer Aided Design Packages for Control Systems, CADCE'85, 7176 (1985)

[5] 斉藤制海, 高橋岳之, and 阿部健一: 数式処理システム REDUCE を用いた制御系の CAD, 計測自動制御学会論文集, **21**–11, 1239/1246 (1985)

[6] 松本英治 and 井上紘一: 数式処理による線形代数と制御系の正準分解への応用, 計測自動制御学会論文集, **24**–4, 370/376 (1988)

[7] S. Hara, I. Akahori, and M. Kaneda: Computer Aided Control System Analysis Using Symbolic Manipulation with Numerical Computation, Control Theory & Advanced Thecnology, **2**–3, 345/365 (1986)

[8] 高橋直樹, 藤田充, 斉藤制海, and 阿部健一: 数式・数値処理による制御系 CAD, 電気学会論文誌 C, **111**–7, 292/298 (1991)

[9] K. J. Astrom: Computer Aided Modeling, Analysis and Design of Control Systems – A Perspective, Control Systems Magazine, 4/16 (1983)

[10] Cleve Moler: MATLAB – User's Guide Albequerque, USA(1980)

[11] Robert Walker, Charles Gregory Jr., and Sunil Shah: MATRIXx: A Data Analysis, System Identification, Control Design and Simulation Package, Control Systems Magazine, 30/37 (1982)

[12] A. J. Laub: A Schur Method for Solving Algebraic Riccati Equations, IEEE Trans. on AC, **24**–6, (1979)

[13] J.J. Dongarra, C.B. Moler, J.R. Bunch, and G.W. Stewart: LINPACK — Users' Guide, SIAM, Philadelphia (1979)

[14] B.S. Garbow, J.M. Boyle, J.J. Dongarra, and C.B. Moler: Matrix Eigensystem Routines – EISPACK Guide Extension, Springer-Verlag (1977)

[15] B.T. Smith, J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema, and C.B. Moler: Matrix Eigensystem Routines – EISPACK Guide, Springer-Verlag (1976)

[16] 佐々木 建昭: 計算機による数式処理, システムと制御, **26**–4, (1982)

[17] 斉藤制海, 阿部健一, and 高橋岳之: 計算機援用による制御工学 – 制御工学と数式処理の周辺, システムと制御, **29**–12, (1985)

[18] Stephen Wolfram: Mathematica, Addison-Wesley Publishing Company, Inc. (1991)

[19] C. Moler, J. Littel, and S. Bangert: PC-MATLAB – User's Guide N. Main St., Sherborn, MA 01770, USA3.1 edition(1987)

[20] Brian W. Kernighan and Dennis M. Ritchie: The C Programming Language, Prentice-Hall (1978)

[21] J.E.Potter: Matrix Quadratic Solution, SIAM Journal on Appl. Math., **14**, 496/501 (1966)

[22] 古賀 雅伸 and 古田 勝久: 制御系解析・設計支援ツール, 第２９回計測自動制御学会学術講演会, 467/468 (1990)