C++による Jordan 標準形の計算

電気電子工学科　立花康夫

Jordan Canonical Form Calculation using C++

Yasuo Tachibana

Abstruct

In this paper we treat the calculation method of Jordan canonical form for a square matrix with complex elements. We construct a class of C++ for 80 bit long double precision floating point complex data using object oriented function and overload of operators, and a class for polynomial data with complex coefficients. We show that the manipulation of polynomials is carried out easily as integer calculation using these classes. The elementary divisors are obtained using the fundamental operations for a matrix with polynomial elements. Simple elementary divisors are calculated using complex Newton-Raphson method. The similar transform matrix is derived from root vectors. We present some example of calculation in order to show the effectiveness of the proposed method.

Key Words: Jordan canonical form, Numerical analysis, Matrix, Programming, C++

1. はじめに

この論文では複素数要素の正方行列を Jordan 標準形に変換する計算を C++プログラミングにより実施する方法について論じる。制御系の解析では Kalman の状態空間での扱いにより系の内部的構造を明確に記述し、入力より出力の生ずる過程を説明するようになった。そのために例え 1 入力 1 出力の系であっても高次の系であれば多次元の変数を同時に扱う必要が生じた。そのために数学的な記述の手段として線形空間(ベクトル空間)における表現が不可欠となり、それらの変換として行列が随所に出現することになった。ベクトル空間の基底をうまく選べば、これらの変換行列は都合の良い標準形とすることができ、理論の展開がスムーズに進展する。特に正方行列の標準形ではいくつかの有用な形が用いられる[1]。例えば行列の固有方程式の係数がそのまま行列上の係数として出現する自然標準形（コンパニオン行列）や固有値がそのまま対角線上に現れる Jordan 標準形（Jordan 行列）等である。固有値が全て異なる場合には Jordan 標準形は完全な対角線形になる。従って、ベクトル空間がいくつかの独立な 1 変数の空間の集まりとなり、動作の説明が大変わかりやすいものとなる。さて、実数または複素数を要素とする正方行列を Jordan 標準形に変換する方法は基本的には行列の固有値とその固有ベクトルおよび根ベクトルを求める問題となる[2]。線形空間に関する数学的な理論展開の上で見れば固有値は固有方程式を求めると、この複素係数の多項式のゼロ点となる。この上で対応する固有ベクトルと根ベクトルを求めることになる。しかし、数値計算上の常識からすれば、このような方法は誤りであるとされている。例えば、岩波講座応用数学[方法 2]「線形計算」（森、杉原、室田著）の第 6 章固有値問題の解法 I[3] (べき乗法)の冒頭には次のように記されている。すなわち「線形代数学における Jordan 標準形にまつわる理論により、固有値と固有ベクトルの数学的性質は十分明らかになっている。… 例えば、与えられた行列から特性方程式の係数を求め、その後に代数方程式の数値解法を利用して固有値を求めるというやり方は典型的な誤りである。云々」となっている。実際には古くから行われている任意のベクトルから始めてその行列による変換を繰り返すことによ

り絶対値の最大の固有値と対応する固有ベクトルを逐次近似していくAscoli-Arzelaの定理[4]を基本とする方法や、ユニタリ変換の反復によりSchur分解を実現する方法等に代表される方法が採られる[5]。しかし、これらのいずれの方法も数値的あるいは数値計算的に妥当であると言うのは計算を実数、場合によっては複素数を浮動小数点数として表現するデータ型に基づいて行っているからである。従来のFortranやBasicまたはPascal、場合によってはC等を用いてこれらの問題を解くためのプログラムを作るとすれば全てが数値的な扱いの処理となる。しかもFortranを除けば複素数型というものは存在しないから実数まで全ての処理を分解してやらなければプログラムできなかった。このような点を考慮すると上記の引用文献のような主張が正しいことになる。ところで、最近ではパーソナルコンピュータ（以後PCと略称する）の性能がいちじるしく向上し、オペレーティングシステム（以後OSと略称する）も米国マイクロソフト社のWindows98やWindows NT等の32ビットOSとなって特に大規模で大容量の計算でない限り数値計算はPC上で十分実用的な結果が出せるようになってきた。さらにグラフィカルユーザインターフェースGUIの効果的な使用が不可欠な前提となるに及んでプログラム言語が大挙してオブジェクト指向機能を取り入れた。この結果、従来から広く用いられていたCもC++となり生まれ変った。C++にはCへのオブジェクト指向機能の付加と共にもう一つ演算子のオーバロードという機能が付加された。この2つの機能の付加はC++を数値計算用のプログラム言語として決定的に魅力あるものにすることになった。すなわち、この論文においては、従来の複素数までのデータ型を使った計算からこれらの機能を用いて複素係数多項式のデータ型を作り、これを使った数値計算を考えてJordan標準形の計算に用いる。さて、このような背景の下で64ビット浮動小数点数を基礎とする複素数型やAnsiString型等はC++の中に標準で定義されている。しかし、この論文ではPCの中に用意されている80ビットの浮動小数点プロセッサの扱う80ビットの長倍精度浮動小数点数型を基礎にした長倍精度複素数型とそれを用いた複素係数多項式型を新たに作って用いる。また、Jordan標準形への変換過程は理論的には多項式行列の処理による単因子を求める処理、単純単因子すなわち単因子の全ゼロ点とその多重度を求めることから構成されている。これらの叙述はすべて多項式の演算に基礎をおいているので、従来の数値計算的観点からアルゴリズム化することは複雑になり過ぎるのでプログラムにはほとんど用いられることはなかった。この論文では多項式演算が普通の整数と同じに扱えるプログラム環境を揃えたことで、この基本的な理論的方法をプログラムのアルゴリズムとして用いることができる。Jordan標準形への変換は固有値の多重度が2以上の場合には数値計算的にはかなり難しい計算であることが知られている。例えばMATLABのように数値計算に特化したソフトウエア体系の中でもJordan標準形は難しい計算のひとつとされている[6]。この論文で用いる多項式演算を基礎とする方法では固有値の多重度を明確に見出すことができるので、可能な限り信頼度の高いJordan標準形への変換が得られる。いくつかの計算例を与えて手法の有効性を示す。

2. PC上での浮動小数点数

PC特に米国Intel社のPentiumプロセッサなどでは浮動小数点演算プロセッサがCPUの中に組み込まれている。このプロセッサは80ビットの浮動小数点演算が可能のように作られている。すなわち、短精度浮動小数点数（32ビット）でも倍精度浮動小数点数（64ビット）でも長倍精度浮動小数点数（80ビット）でも同じに演算する。すなわち、データの出し入れにかかる時間はそれぞれ異なっても正味の演算時間は同じ程度になる。ワークステーション等では通常64ビットの倍精度浮動小数点数による計算が基本であるので、PCでも多くのソフトウエアが計算は倍精度浮動小数点数で実施している。ところで倍精度浮動小数点数はPCの場合では約15桁の精度と 2.2×10^{-308}～1.7×10^{308} のダイナミックレンジを持つ。一方PC特有の長倍精度浮動小数点数では約19桁の精度と 1.0×10^{-4932}～1.7×10^{4932} のダイナミックレンジを持つ。この高い精度と極端に広い

ダイナミックレンジは多少の時間がかかる点をがまんすると数値計算的には好ましいものである。この長倍精度の浮動小数点数を利用できるようなコンパイラとしてこの論文では米国Borland社のBorland C++ V5.0 1[7]を用いる。

3. 長倍精度複素数演算

C++の中には2つの倍精度浮動小数点数から構成されるcomplexというデータ型（class）が存在する。具体的にはプログラムの冒頭にcomplex.hというヘッダファイルをインクルードすればこの倍精度複素数演算が通常の実数の演算と同様に利用できる。またcomplex.hの中には各種の関数、特に複素数を変数とする初等関数なども含まれている。さらに複素数と実数の組み合わせの種種の演算も可能のようになっている。これはC++の持つオブジェクト指向の機能と演算子のオーバロードの機能により実現されるものである。この論文の主題である複素数要素の正方行列のJordan標準形への変換ではcomplexにより準備されたすべての機能は必ずしも必要としない。むしろ、2つの実数を長倍精度の浮動小数点数として表現したほうが都合が良い。そこで、この論文では新たに長倍精度複素数を一つの型として定義（class）してこの目的を達成する。このclassの定義などはecomplex.hというヘッダファイルにまとめプログラムの冒頭にてインクルードすることで長倍精度複素数演算数が可能となるように工夫してある。すなわち、複素数$z = x + jy$について実数部xと虚数部yは共に長倍精度浮動小数点数として表される。**付録1**にはこの論文で用いたecomplex.hの全リストを示す。特に複素変数の初等関数は$\mathrm{sinl}(x)$と通常の関数名の後ろにlを付してある。

4. 長倍精度複素係数多項式の演算

この論文では多項式行列（多項式を要素とする行列）に関するいくつかの基本的な演算処理により複素数要素の正方行列をJordan標準形に変換するのである。そのために多項式の演算が容易にできる必要がある。そこで、前節で与えた長倍精度複素数を係数とする多項式データ型（class）を作りこの目的に沿うように工夫する。このデータ型の定義はecpolynom.hと名づけたヘッダファイルにまとめてある。**付録2**にはこのプログラムの全リストを示す。すなわち、λの多項式

$$f(\lambda) = f_m + f_{m-1}\cdot\lambda + \cdots + f_0\cdot\lambda^m \tag{1}$$

に関して次数mをint型変数、係数$f_0, f_1, \cdots, f_m$をecomplex型変数として扱う。演算子のオーバロードの定義において多項式同士の和、差、積、および商や剰余の演算が可能のようにしてある。すなわち、多項式同士の演算がint型変数や浮動小数点型の変数とまったく同じように演算子*,+,-,/,%等により実施できることになる。例えば

$$\left.\begin{aligned} F_1(\lambda) &= 5 + j2 + (3+j)\lambda + j4\lambda^2 \\ F_2(\lambda) &= 10 + j3 + (2+j6)\lambda + (5+j2)\lambda^2 + (3+j)\lambda^3 \end{aligned}\right\} \tag{2}$$

として

$$\left.\begin{aligned} F_3(\lambda) &= (4+j3)F_1(\lambda) & F_4(\lambda) &= F_1(\lambda) + F_2(\lambda) \\ F_5(\lambda) &= F_1(\lambda) - F_2(\lambda) & F_6(\lambda) &= F_1(\lambda)F_2(\lambda) \\ F_7(\lambda) &= F_1(\lambda) / F_2(\lambda) & F_8(\lambda) &= F_2(\lambda)\%F_1(\lambda) \end{aligned}\right\} \tag{3}$$

とすると、これはFig.1のようにコード化される。この実行結果はFig.2のようになる。

```
////////////////////////////////////////////////////////////////////////
// ECPLYTST.CPP  長倍精度複素数係数多項式計算テスト 立花康夫 1998.9.14  //
////////////////////////////////////////////////////////////////////////
//
//  インクルードファイル
#include <windows.h>
#include <stdio.h>
#include <math.h>
#include "ecpolynom.h"  // 長倍精度複素数係数多項式定義ヘッダファイル
//
//  関数プロトタイプ宣言
void ECPLYTST(int id); // 長倍精度複素数係数多項式計算テストプログラム
void OutResult(FILE *Fout, ecpolynom *F, char *mes);     // 計算結果出力
//
////////////////////////////////////////////////////////////////////////
                        メインプログラム
////////////////////////////////////////////////////////////////////////
int PASCAL WinMain(HINSTANCE HCurInstance, HINSTANCE HPrevInstance,
                   LPSTR lpCmdLine, int nCmdShow) {

    int     id;

    if (HCurInstance==NULL) { id=0; }
    if (HPrevInstance==NULL) { id=0; };
    if (lpCmdLine==NULL) { id=0; };
    if (nCmdShow==NULL) { id=0; };

    ECPLYTST(id);       // 計算テストプログラム

    return 0;
}
////////////////////////////////////////////////////////////////////////
// ECPLYTST  計算テストプログラム                立花康夫 1998.9.14    //
////////////////////////////////////////////////////////////////////////
void ECPLYTST(int id) {

    char
    Outfile[]="d:¥¥TACHI¥¥TMP¥¥CPP32¥¥ECPLYTSTA¥¥ECPLYTSTA.OUT";
                                                    // 出力ファイル名
    FILE          *Fout;                            // 出力ファイルポインタ

    if (id==0) {
        MessageBox(NULL,"POLYTST多項式試験を開始します。立花康夫",
                "ECPLYTST",MB_OK | MB_ICONEXCLAMATION);
```

```
    }

    if ((Fout=fopen(Outfile,"wt"))==NULL) {
        MessageBox(NULL,"出力ファイルがひらけません。","ECPLYTST",
                        MB_OK | MB_ICONEXCLAMATION);
        exit(1);
    }                                           // 出力ファイルオープン

    // ------------------------------------------------------------------

    ecpolynom F1,F2,F3,F4,F5,F6,F7,F8,F9;       // 多項式変数

    F1.PutOrder(2);                             // F1(λ)J=5+j2+(3+j)λ+j4λ^2
    F1.PutCoef(0,ecomplex(5,2));                // F1(λ)定義
    F1.PutCoef(1,ecomplex(3,1));
    F1.PutCoef(2,ecomplex(0,4));

    F2.PutOrder(3);          // F2(λ)J=10+j3+(2+j6)λ+(5+j2)λ^2+(3+j)λ^3
    F2.PutCoef(0,ecomplex(10,3));               // F2(λ)定義
    F2.PutCoef(1,ecomplex(2,6));
    F2.PutCoef(2,ecomplex(5,2));
    F2.PutCoef(3,ecomplex(3,1));

    F3=ecomplex(4,3)*F1;                        // 実際の多項式の計算
    F4=F1+F2;
    F5=F1-F2;
    F6=F1*F2;
    F7=F2/F1;
    F8=F2%F1;
    F9=F7*F1+F8;                                // F9=F2

    // ------------------------------------------------------------------

    fprintf(Fout,"長倍精度複素数係数多項式演算試験結果");
    OutResult(Fout,&F1,"F1          ");   // F1(λ)～F8(λ)  出力
    OutResult(Fout,&F2,"F2          ");
    OutResult(Fout,&F3,"F3=(4+3j)*F1");
    OutResult(Fout,&F4,"F4=F1+F2    ");
    OutResult(Fout,&F5,"F5=F1-F2    ");
    OutResult(Fout,&F6,"F6=F1*F2    ");
    OutResult(Fout,&F7,"F7=F2/F1    ");
    OutResult(Fout,&F8,"F8=F2%F1    ");
    OutResult(Fout,&F9,"F9=F7*F1+F8 ");

    fclose(Fout);

    printf("¥n¥n処理が終了しました。何かキーを押して下さい。");
    MessageBox(NULL,"POLYTST の計算が終了しました。","ECPLYTST",
                    MB_OK | MB_ICONEXCLAMATION);

}
//////////////////////////////////////////////////////////////////////////
//  OutResult  計算結果出力                        立花康夫 1998.9.14    //
//////////////////////////////////////////////////////////////////////////
void OutResult(FILE *Fout, ecpolynom *F, char *mes) {

    int         i;              // 汎用インデックス

    fprintf(Fout,"¥n¥n%s   n=%3d",mes, F->GetOrder());

    for (i=0;i<=F->GetOrder();i++) {
        fprintf(Fout,"¥n%3d %25.19Lf %25.19Lf",
                        i,real(F->GetCoef(i)),imag(F->GetCoef(i)));
    }
}
```

Fig.1 長倍精度複素数係数多項式試験プログラム

```
長倍精度複素数係数多項式演算試験結果

F1              n=  2
  0     5.0000000000000000000     2.0000000000000000000
  1     3.0000000000000000000     1.0000000000000000000
  2     0.0000000000000000000     4.0000000000000000000

F2              n=  3
  0    10.0000000000000000000     3.0000000000000000000
  1     2.0000000000000000000     6.0000000000000000000
  2     5.0000000000000000000     2.0000000000000000000
  3     3.0000000000000000000     1.0000000000000000000

F3=(4+3j)*F1    n=  2
  0    14.0000000000000000000    23.0000000000000000000
  1     9.0000000000000000000    13.0000000000000000000
  2   -12.0000000000000000000    16.0000000000000000000

F4=F1+F2        n=  3
  0    10.0000000000000000000     3.0000000000000000000
  1     7.0000000000000000000     8.0000000000000000000
  2     8.0000000000000000000     3.0000000000000000000
  3     3.0000000000000000000     5.0000000000000000000

F5=F1-F2        n=  3
  0   -10.0000000000000000000    -3.0000000000000000000
  1     3.0000000000000000000    -4.0000000000000000000
  2    -2.0000000000000000000    -1.0000000000000000000
  3    -3.0000000000000000000     3.0000000000000000000

F6=F1*F2        n=  5
  0    44.0000000000000000000    35.0000000000000000000
  1    25.0000000000000000000    53.0000000000000000000
  2     9.0000000000000000000    80.0000000000000000000
  3     2.0000000000000000000    30.0000000000000000000
  4     0.0000000000000000000    26.0000000000000000000
  5    -4.0000000000000000000    12.0000000000000000000

F7=F2/F1        n=  1
  0     1.9310344827586206900    -0.1724137931034482759
  1    -0.3674197384066587396     1.0642092746730083240

F8=F2%F1        n=  1
  0     6.4768133174791914390    -8.5493460166468489900
  1     7.2568370986920332940     2.4696789536266349580

F9=F7*F1+F8     n=  3
  0    10.0000000000000000000     3.0000000000000000000
  1     2.0000000000000000000     6.0000000000000000000
  2     5.0000000000000000000     2.0000000000000000000
  3     3.0000000000000000000     1.0000000000000000000
```

Fig.2 長倍精度複素数係数多項式試験結果

5. 多項式の行列の基本変換

複素数体を**C**とする。要素が**C**に含まれるような行列を**C**-行列と呼ぶことにする。$n \times n$ の任意の**C**-行列を**A**とする。この論文では**A**を Jordan 標準形に変換する問題を扱うわけである。この処理を実施するために変数λに対して行列$\lambda\mathbf{I}-\mathbf{A}$にいくつかの変換を施すことになる。この行列は対角線上の要素だけがλの1次の多項式である。しかし、変換の繰り返しで、高次の多項式が要素として出現する。そこで、係数が**C**に含まれるようなλの多項式を要素にする$n \times n$行列

$$\mathbf{F}(\lambda)=\begin{bmatrix} f_{11}(\lambda) & \cdots & f_{11}(\lambda) \\ \vdots & \ddots & \vdots \\ f_{11}(\lambda) & \cdots & f_{11}(\lambda) \end{bmatrix} \tag{4}$$

を考える。このような行列を多項式の行列あるいはλ-行列と呼ぶ。このλ-行列に対して次のような変換を考える。

1° ある行にゼロと異なる**C**の数を乗ずること。
2° ある行に任意の多項式を掛けてそれを別のある行に加えること。
3° ある列にゼロと異なる**C**の数を乗ずること。
4° ある列に任意の多項式を掛けてそれを別のある列に加えること。

を基本の操作という。特に1°と2°は行操作、3°と4°は列操作という。線形代数学上の基本的な結果として多くのことが知られている。λ-行列**F**に基本操作を施してλ-行列**G**が得られたとすると**F**と**G**は同値であるという。この同値関係は推移的、対称的および反射的であるので、これによりλ-行列を分類することができる。すなわち、標準形というものを考えることができる。この論文で作成した ecploynom 型を用いればλ-行列の基本操作は実数要素の行列に対する基本操作（1°と3°で**C**を実数に、2°と4°で任意の多項式を任意の実数に置き換えたもの）とま

ったく同じように実行できる。ところで、行操作1°と2°を繰り返し用いると2つの行の入れ替えができる。例えば1行と2行を交換するには

$$\begin{bmatrix} f_{11} & \cdots & f_{1n} \\ f_{21} & \cdots & f_{2n} \\ \cdots & \cdots & \cdots \end{bmatrix} \overset{[1]}{\Rightarrow} \begin{bmatrix} f_{11} & \cdots & f_{1n} \\ f_{21}+f_{11} & \cdots & f_{2n}+f_{1n} \\ \cdots & \cdots & \cdots \end{bmatrix} \overset{[2]}{\Rightarrow}$$

$$\begin{bmatrix} -f_{21} & \cdots & -f_{2n} \\ f_{21}+f_{11} & \cdots & f_{2n}+f_{1n} \\ \cdots & \cdots & \cdots \end{bmatrix} \overset{[3]}{\Rightarrow}$$

$$\begin{bmatrix} -f_{21} & \cdots & -f_{2n} \\ f_{11} & \cdots & f_{1n} \\ \cdots & \cdots & \cdots \end{bmatrix} \overset{[4]}{\Rightarrow} \begin{bmatrix} f_{21} & \cdots & f_{2n} \\ f_{11} & \cdots & f_{1n} \\ \cdots & \cdots & \cdots \end{bmatrix} \quad (5)$$

とすれば良い．ここで、[1]は1行目を2行目に加える。[2]では2行目に-1をかけて1行目に加える。[3]では1行目を2行目に加える。[4]では1行目に-1を掛ける。同じように3°と4°により列の入れ替えができる。次に$n\times n$のλ-行列の標準対角行列

$$\begin{bmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & f_1(\lambda) & & & & \\ & & & & \ddots & & & \\ & & & & & f_k(\lambda) & & \\ & & & & & & 0 & \\ & & & & & & & \ddots \\ & & & & & & & & 0 \end{bmatrix} \quad (6)$$

を考える。$f_1(\lambda)$,…, $f_k(\lambda)$ のλの最高次の係数は1とし、$f_1(\lambda)$は$f_2(\lambda)$を、$f_2(\lambda)$は$f_3(\lambda)$を、そして、$f_{k-1}(\lambda)$ は$f_k(\lambda)$を整除する(割り切る)。すべてのλ-行列は基本操作により標準対角行列に変換できしかも一意的である。この際に用いられる演算は基本操作と多項式の商と剰余を求める演算の繰り返しである。この過程の一部を考えてみる。λ-行列**F**の要素がすべて0であれば既に標準対角形になっている。そこでいずれか1つの要素が0でないものが存在する。そのようなものの中で次数の最小のものを$f_{ij}(\lambda)$とする。このときi行と1行を交換し、次にj列と1列を交換すれば$f_{ij}(\lambda)$は1行1列目の要素となる。そこではじめから1行1列目の要素$f_{11}(\lambda)$は次数最小であるとする。次に$f_{i1}(\lambda)$を$f_{11}(\lambda)$で割った時の商と剰余を$q_i(\lambda)$，$r_i(\lambda)$とすると

$$f_{i1}(\lambda)=f_{11}(\lambda)q_i(\lambda)+r_i(\lambda) \quad (i=2,\cdots,n) \quad (7)$$

となる。そこで、1行目に$-q_i(\lambda)$を掛けてi行目に加えると$(i,1)$要素は$r_i(\lambda)$となる。もし、$r_i(\lambda)$が0でなければ$f_{11}(\lambda)$が最小次数であることに反する。従って$r_i(\lambda)=0$でなければならない。すなわちλ-行列**F**は

$$\mathbf{H}=\begin{bmatrix} f_{11}(\lambda) & 0 & \cdots & 0 \\ 0 & h_{22}(\lambda) & \cdots & h_{2n}(\lambda) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & h_{n2}(\lambda) & \cdots & h_{nn}(\lambda) \end{bmatrix} \quad (8)$$

の形のλ-行列**H**と同値となる。ここで$h_{ij}(\lambda)$は$f_{11}(\lambda)$で割り切れる。このような過程を繰り返すことにより標準対角形に達することができる。

6. 単因子と単純単因子

標準対角形行列を書き直して

$$\mathbf{D}=\begin{bmatrix} d_1(\lambda) & & \\ & \ddots & \\ & & d_n(\lambda) \end{bmatrix} \quad (9)$$

のようにする。このとき、$d_1(\lambda),d_2(\lambda),\cdots,d_n(\lambda)$を$\lambda$-行列**D**の単因子と呼ぶ。このとき、行列**D**の$k\times k$ $(k=1,2,\cdots,n)$小行列式の最大公約数を$D_k(\lambda)$とすると

$$D_k(\lambda)=d_1(\lambda)d_2(\lambda)\cdots d_n(\lambda) \quad (k=1,2,\cdots,n) \quad (10)$$

であることが容易にわかる。$D_k(\lambda)$はλ-行列**D**の行列式因子という。これはまた

$$D_k(\lambda)/D_{k-1}(\lambda)=d_k(\lambda) \quad (k=2,3,\cdots,n) \quad (11)$$

とも表される。

次に単因子$d_k(\lambda)$を

$$d_k(\lambda)=\{e_1(\lambda)\}^{n_1}\{e_2(\lambda)\}^{n_2}\cdots\{e_{i_k}(\lambda)\}^{n_{i_k}} \quad (12)$$

のように因数分解する。ここで$e_1(\lambda),e_2(\lambda),\cdots,e_{i_k}(\lambda)$は互いに素で最高次の係数が1であるような既約多項式であり、実際にはλの1次式である。$\{e_1(\lambda)\}^{n_1},\{e_2(\lambda)\}^{n_2},\cdots,\{e_{i_k}(\lambda)\}^{n_{i_k}}$ $(k=1,2,\cdots,n)$を**D**の単純単因子と呼ぶ。例えば$d_1(\lambda)=1,d_2(\lambda)=(\lambda+1),d_3(\lambda)=\lambda^2(\lambda+1)^2$ならば単純単因子は$\lambda^2,\lambda+1,(\lambda+1)^2$となる。単純単因子の具体的な計算は8節で述べる。

7. 相似変換

C-行列**A**と**B**が相似であるとは正則な**C**-行列**T**が存在して

$$\mathbf{A}=\mathbf{T}^{-1}\mathbf{B}\mathbf{T} \tag{13}$$

となることをいう。ところですべての**C**-行列**A**はJordan標準形

$$\mathbf{J}=\begin{bmatrix}\mathbf{J}_1 & & \\ & \ddots & \\ & & \mathbf{J}_s\end{bmatrix} \tag{14}$$

と相似となる。$\mathbf{J}_k$ $(k=1,2,\cdots,s)$は

$$\mathbf{J}_k=\begin{bmatrix}\rho_k & 1 & & \\ & \rho_k & \ddots & \\ & & \ddots & 1 \\ & & & \rho_k\end{bmatrix} \tag{15}$$

という形をしていてJordan細胞と呼ぶ。ρ_kは**A**の固有値の一つである。線形代数の知識からすれば**C**-行列**A**と**B**が相似であることはλ-行列$\lambda\mathbf{I}-\mathbf{A}$と$\lambda\mathbf{I}-\mathbf{B}$（**I**は$n\times n$単位行列である）が同値であることが必要十分な条件である。ここで$\lambda\mathbf{I}_k-\mathbf{J}_k$の（$\mathbf{I}_k$は$\mathbf{J}_k$と同じ形の単位行列である）行列式因子を計算してみれば直ちにわかるが、この行列の単純単因子は唯ひとつ$(\lambda-\rho_k)^{n_k}$である。ところで

$$\lambda\mathbf{I}-\mathbf{J}=\begin{bmatrix}\lambda\mathbf{I}_1-\mathbf{J}_1 & & \\ & \ddots & \\ & & \lambda\mathbf{I}_s-\mathbf{J}_s\end{bmatrix} \tag{16}$$

であり、従って$\lambda\mathbf{I}-\mathbf{A}$の単純単因子は$(\lambda-\rho_1)^{n_1},(\lambda-\rho_2)^{n_2},\cdots,(\lambda-\rho_s)^{n_s}$となる。すなわち、単純単因子がわかればその段階でJordan標準形が確定する。次に、**C**-行列**A**のJordan標準形**J**が求められたとして相似変換**U**すなわち

$$\mathbf{J}=\mathbf{U}^{-1}\mathbf{A}\mathbf{U} \tag{17}$$

となる**U**はどのようにして求めるかを考える。議論を明確にするために**J**は単一の細胞のみからなる、すなわち、

$$\mathbf{J}=\begin{bmatrix}\rho & 1 & & \\ & \rho & \ddots & \\ & & \ddots & 1 \\ & & & \rho\end{bmatrix} \tag{18}$$

の場合を考える。この際、

$$(\rho\mathbf{I}-\mathbf{A})^k\mathbf{a}=\mathbf{0}\ \ (k=1,2,\cdots,n) \tag{19}$$

となる零でないベクトル**a**を高さkの根ベクトルという。$k=1$の場合は固有ベクトルである。(18)式の場合、最も高い根ベクトル$\mathbf{a}_n$はゼロベクトルでないように任意に与えることができる。これから、

$$\left.\begin{aligned}\mathbf{a}_{n-1}&=\mathbf{A}\mathbf{a}_n-\rho\mathbf{a}_n\\ \mathbf{a}_{n-2}&=\mathbf{A}\mathbf{a}_{n-1}-\rho\mathbf{a}_{n-1}\\ &\cdots\\ \mathbf{0}&=\mathbf{A}\mathbf{a}_1-\rho\mathbf{a}_1\end{aligned}\right\} \tag{20}$$

のように$\mathbf{a}_1,\mathbf{a}_2,\cdots,\mathbf{a}_n$を求める。このようにすると

$$\mathbf{A}[\mathbf{a}_1,\mathbf{a}_2,\cdots,\mathbf{a}_n]=[\mathbf{a}_1,\mathbf{a}_2,\cdots,\mathbf{a}_n]\mathbf{J} \tag{21}$$

が得られる。$\mathbf{a}_1,\mathbf{a}_2,\cdots,\mathbf{a}_n$は線形独立であることがわかっているので、

$$\mathbf{U}=[\mathbf{a}_1,\mathbf{a}_2,\cdots,\mathbf{a}_n] \tag{22}$$

は正則行列となり相似変換行列となる。

8. 単純単因子の具体的な計算

次に単因子$d_k(\lambda)$の単純単因子を計算する方法を考える。添え字kを省略して

$$d(\lambda)=\{e_1(\lambda)\}^{n_1}\{e_2(\lambda)\}^{n_2}\cdots\{e_t(\lambda)\}^{n_t} \tag{23}$$

とする。この多項式のすべての零点を求めることを考える。すべての零点を単根としてのみ有する多項式を

$$p(\lambda)=e_1(\lambda)e_2(\lambda)\cdots e_t(\lambda) \tag{24}$$

とする。次に$p(\lambda)$は$d(\lambda)$と$d'(\lambda)$の最大公約式を$q(\lambda)$とすると、

$$p(\lambda)=d(\lambda)/q(\lambda) \tag{25}$$

である。最大公約式$q(\lambda)$は商と剰余の計算を繰り返し実行するEuclidの互除法により求められる。これも我々の用意したecpolynom型の変数を使って整数の場合と同じような手軽さで実施できる。そして、$p(\lambda)$の零点はNewton-Raphson法

$$\lambda_k=\lambda_{k-1}-\frac{p(\lambda_{k-1})}{p'(\lambda_{k-1})}\quad(k=1,2,\cdots) \tag{26}$$

により容易に求められる。$p(\lambda)$が単根しか持たないので、効率的な収束が期待できる。初期値λ_0として$R\cdot\exp\{j2\pi(2\mu+1)/2\ell\}$ $(\mu=0,1,\cdots,\ell-1)$等と選べばほぼ同時にすべてのゼロ点が手に入る。ここで$|\lambda|\le R$にはすべての零点が含まれるとする。Rの評価については$p(\lambda)$の係数から種々の値が提案されている[8]。一つの零点の重複度は微分系列

$d(\lambda), d'(\lambda), \cdots, d^{(s)}(\lambda)$ を計算し、どこで零で無くなるかを見れば判別することができる。冒頭に述べたように **C**-行列 **A** の固有方程式 $|\lambda\mathbf{I}-\mathbf{A}|=0$ は n 次の方程式となり、いきなりその数値解法を試みても n が大きければ高次の方程式の解法となり困難が伴う。しかし、単因子は

$$|\lambda\mathbf{I}-\mathbf{A}| = d_1(\lambda)d_2(\lambda)\cdots d_n(\lambda) \qquad (27)$$

を満たす。そこで、通常、各 $d_k(\lambda)$ は n よりも低次の多項式となり求根に際して取り扱いやすく精度の点でも有利に働くと考えられる。

8. 計算例

前節までに述べたように C++上で ecomplex と ecpolynom クラスを用いて多項式演算に基礎をおいたアルゴリズムにより **C**-行列の Jordan 標準形への変換プログラムを作成し、それを用いて計算した例を示す。

例

$$\mathbf{A}=\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ -2 & 0 & 0 & 2 & 3 \\ -1 & 0 & 1 & 1 & 1 \\ -1 & 1 & -1 & 2 & 1 \\ -1 & -2 & 1 & 1 & 4 \end{bmatrix} \Rightarrow \mathbf{J}=\begin{bmatrix} 2 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

の計算例を Fig.4 に示す。

```
行列の Jordan 標準形
ケース A OK(A)       OK(G)
                                        1998-9-28  9.37.24

Jordan 標準形計算
入力行列      A  n=    5

   i    j ord    k          real{a(i,j)(k)}          imag{a(i,j)(k)}
   1    1    0    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )
   2    1    0    0 (  -2.0000000000000000e+00     0.0000000000000000e+00  )
   3    1    0    0 (  -1.0000000000000000e+00     0.0000000000000000e+00  )
   4    1    0    0 (  -1.0000000000000000e+00     0.0000000000000000e+00  )
   5    1    0    0 (  -1.0000000000000000e+00     0.0000000000000000e+00  )
   1    2    0    0 (   0.0000000000000000e+00     0.0000000000000000e+00  )
   2    2    0    0 (   0.0000000000000000e+00     0.0000000000000000e+00  )
   3    2    0    0 (   0.0000000000000000e+00     0.0000000000000000e+00  )
   4    2    0    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )
   5    2    0    0 (  -2.0000000000000000e+00     0.0000000000000000e+00  )
   1    3    0    0 (   0.0000000000000000e+00     0.0000000000000000e+00  )
   2    3    0    0 (   0.0000000000000000e+00     0.0000000000000000e+00  )
   3    3    0    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )
   4    3    0    0 (  -1.0000000000000000e+00     0.0000000000000000e+00  )
   5    3    0    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )
   1    4    0    0 (   0.0000000000000000e+00     0.0000000000000000e+00  )
   2    4    0    0 (   2.0000000000000000e+00     0.0000000000000000e+00  )
   3    4    0    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )
   4    4    0    0 (   2.0000000000000000e+00     0.0000000000000000e+00  )
   5    4    0    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )
   1    5    0    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )
   2    5    0    0 (   3.0000000000000000e+00     0.0000000000000000e+00  )
   3    5    0    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )
   4    5    0    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )
   5    5    0    0 (   4.0000000000000000e+00     0.0000000000000000e+00  )

固有値絶対値上限  trace=   3.741657386773941386e+00

単因子       n=  5

no=  1/  5
nord=    0
    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )

no=  2/  5
nord=    0
    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )

no=  3/  5
nord=    0
    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )

no=  4/  5
nord=    0
    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )

no=  5/  5
nord=    5
    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )
    1 (  -8.0000000000000000e+00     0.0000000000000000e+00  )
    2 (   2.5000000000000000e+01     0.0000000000000000e+00  )
    3 (  -3.7999999999999999e+01     0.0000000000000000e+00  )
    4 (   2.8000000000000000e+01     0.0000000000000000e+00  )
    5 (  -7.9999999999999998e+00     0.0000000000000000e+00  )

特性多項式
n=    5
    0 (   1.0000000000000000e+00     0.0000000000000000e+00  )
    1 (  -8.0000000000000000e+00     0.0000000000000000e+00  )
    2 (   2.5000000000000000e+01     0.0000000000000000e+00  )
    3 (  -3.7999999999999999e+01     0.0000000000000000e+00  )
    4 (   2.8000000000000000e+01     0.0000000000000000e+00  )
    5 (  -7.9999999999999998e+00     0.0000000000000000e+00  )


単純単因子根と次数       n=  5

sd[    1]=(    1.0000000000000000629e+00        0.0000000000000000e+00    )
ord[  1]=  2

sd[    2]=(    1.9999999999999999916e+00        0.0000000000000000e+00    )
ord[  2]=  3

固有値の分類       n=  5
        異なる固有値の数       nev=  2
no=  1/  2       単純単因子の個数  sdev=  1
 固有値=(  1.0000000000000000629e+00     0.0000000000000000e+00  )
  i  sordev:単純単因子の次数
  1  2
no=  2/  2       単純単因子の個数  sdev=  1
 固有値=(   1.9999999999999999916e+00     0.0000000000000000e+00  )
  i  sordev:単純単因子の次数
  1  3

固有ベクトルの分類       n=  5       異なる固有値の数       nev=  2

固有ベクトルと根ベクトルの分類       n=  5       異なる固有値の数       nev=  2

eno=  1 /  2  番固有値       根ベクトルの最大の高さ hno=  2
 固有値=(  1.0000000000000000629e+00     0.0000000000000000e+00  )
 高さ  1の根ベクトルの1次独立なものの数  hord=  1
   no=  1/  1 番目の根ベクトル
  1    (   9.9999999999999371e-01     0.0000000000000000e+00  )
  2    (  -1.8821749714348356698e-16     0.0000000000000000e+00  )
  3    (  -3.7643499428669671395e-16     0.0000000000000000e+00  )
  4    (   9.9999999999999974489e-01     0.0000000000000000e+00  )
  5    (   6.2883726004159257117e-17     0.0000000000000000e+00  )

eno=  1 /  2  番固有値       根ベクトルの最大の高さ hno=  2
 固有値=(  1.0000000000000000629e+00     0.0000000000000000e+00  )
 高さ  2の根ベクトルの1次独立なものの数  hord=  1
   no=  1/  1 番目の根ベクトル
  1    (   1.0000000000000003764e+00     0.0000000000000000e+00  )
  2    (   9.9999999999999937725e-01     0.0000000000000000e+00  )
  3    (  -7.5286998857393430326e-16     0.0000000000000000e+00  )
  4    (   0.0000000000000000000e+00     0.0000000000000000e+00  )
  5    (   1.0000000000000000000e+00     0.0000000000000000e+00  )

eno=  2 /  2  番固有値       根ベクトルの最大の高さ hno=  3
 固有値=(   1.9999999999999999916e+00     0.0000000000000000e+00  )
 高さ  1の根ベクトルの1次独立なものの数  hord=  1
   no=  1/  1 番目の根ベクトル
  1    (   2.1033522146218786602e-16     0.0000000000000000e+00  )
  2    (   5.0000000000000001893e-01     0.0000000000000000e+00  )
  3    (   4.9999999999999999790e-01     0.0000000000000000e+00  )
  4    (   5.0000000000000000631e-01     0.0000000000000000e+00  )
  5    (   2.1033522146218786602e-16     0.0000000000000000e+00  )

eno=  2 /  2  番固有値       根ベクトルの最大の高さ hno=  3
 固有値=(   1.9999999999999999916e+00     0.0000000000000000e+00  )
 高さ  2の根ベクトルの1次独立なものの数  hord=  1
   no=  1/  1 番目の根ベクトル
  1    (   4.9999999999999997686e-01     0.0000000000000000e+00  )
  2    (  -2.1033522146218786602e-16     0.0000000000000000e+00  )
  3    (  -5.0000000000000001052e-01     0.0000000000000000e+00  )
  4    (  -2.5240226575462543222e-16     0.0000000000000000e+00  )
```

```
  5   (   4.99999999999999369e-01     0.00000000000000000e+00  )

eno=  2 /  2  番固有値       根ベクトルの最大の高さ hno=  3
 固有値=(   1.99999999999999916e+00     0.00000000000000000e+00  )
 高さ  3の根ベクトルの1次独立なものの数  hord=  1

   no=  1/  1 番目の根ベクトル

   1   (  -4.99999999999998107e-01     0.00000000000000000e+00  )
   2   (   4.99999999999999369e-01     0.00000000000000000e+00  )
   3   (   1.00000000000000000e+00     0.00000000000000000e+00  )
   4   (   0.00000000000000000e+00     0.00000000000000000e+00  )
   5   (   0.00000000000000000e+00     0.00000000000000000e+00  )
T   m=    5    n=    5

   i    j          real(a(i,j)(k))          imag(a(i,j)(k))
   1    1  (   2.10335221462187860 2e-16    0.00000000000000000e+00  )
   2    1  (   5.00000000000000189 3e-01    0.00000000000000000e+00  )
   3    1  (   4.99999999999999790e-01      0.00000000000000000e+00  )
   4    1  (   5.00000000000000631e-01      0.00000000000000000e+00  )
   5    1  (   2.10335221462187860 2e-16    0.00000000000000000e+00  )
   1    2  (   4.99999999999997686e-01      0.00000000000000000e+00  )
   2    2  (  -2.10335221462187860 2e-16    0.00000000000000000e+00  )
   3    2  (  -5.00000000000001052e-01      0.00000000000000000e+00  )
   4    2  (  -2.52402265754625432 2e-16    0.00000000000000000e+00  )
   5    2  (   4.99999999999999369e-01      0.00000000000000000e+00  )
   1    3  (  -4.99999999999998107e-01      0.00000000000000000e+00  )
   2    3  (   4.99999999999999369e-01      0.00000000000000000e+00  )
   3    3  (   1.00000000000000000e+00      0.00000000000000000e+00  )
   4    3  (   0.00000000000000000e+00      0.00000000000000000e+00  )
   5    3  (   0.00000000000000000e+00      0.00000000000000000e+00  )
   1    4  (   9.99999999999999371e-01      0.00000000000000000e+00  )
   2    4  (  -1.88217497143483569 8e-16    0.00000000000000000e+00  )
   3    4  (  -3.76434994286967139 5e-16    0.00000000000000000e+00  )
   4    4  (   9.99999999999997489e-01      0.00000000000000000e+00  )
   5    4  (   6.28837260041592571 7e-17    0.00000000000000000e+00  )
   1    5  (   1.00000000000000376 4e+00    0.00000000000000000e+00  )
   2    5  (   9.99999999999993725e-01      0.00000000000000000e+00  )
   3    5  (  -7.52869988573934302 6e-16    0.00000000000000000e+00  )
   4    5  (   0.00000000000000000e+00      0.00000000000000000e+00  )
   5    5  (   1.00000000000000000e+00      0.00000000000000000e+00  )

(T)^(-1)  m=    5    n=    5

   i    j          real(a(i,j)(k))          imag(a(i,j)(k))
   1    1  (  -1.99999999999999750 4e+00    0.00000000000000000e+00  )
   2    1  (  -2.00000000000000016 5e+00    0.00000000000000000e+00  )
   3    1  (   6.69603261727047532 8e-16    0.00000000000000000e+00  )
   4    1  (   9.99999999999997476e-01      0.00000000000000000e+00  )
   5    1  (   1.00000000000000031 40e+00   0.00000000000000000e+00  )
   1    2  (   1.99999999999998735e+00      0.00000000000000000e+00  )
   2    2  (  -2.00000000000000084 0e+00    0.00000000000000000e+00  )
   3    2  (  -1.99999999999999579e+00      0.00000000000000000e+00  )
   4    2  (  -1.00000000000000249 8e+00    0.00000000000000000e+00  )
   5    2  (   9.99999999999999358e-01      0.00000000000000000e+00  )
   1    3  (  -1.99999999999998738e+00      0.00000000000000000e+00  )
   2    3  (   5.89805981832113948 1e-16    0.00000000000000000e+00  )
   3    3  (   1.99999999999999998e+00      0.00000000000000000e+00  )
   4    3  (   9.99999999999997463e-01      0.00000000000000000e+00  )
   5    3  (   6.28837260041592769 1e-17    0.00000000000000000e+00  )
   1    4  (   1.99999999999999750 4e+00    0.00000000000000000e+00  )
   2    4  (   1.99999999999999913e+00      0.00000000000000000e+00  )
   3    4  (  -4.18935719448398839 4e-16    0.00000000000000000e+00  )
   4    4  (   5.03503488902268259 2e-16    0.00000000000000000e+00  )
   5    4  (  -1.00000000000000251 1e+00    0.00000000000000000e+00  )
   1    5  (   1.51701567974171780 4e-15    0.00000000000000000e+00  )
   2    5  (   4.00000000000000502e+00      0.00000000000000000e+00  )
   3    5  (   1.99999999999999160e+00      0.00000000000000000e+00  )
   4    5  (   2.51101223147642826 9e-16    0.00000000000000000e+00  )
   5    5  (  -9.99999999999999987e-01      0.00000000000000000e+00  )

J=(T)(-1)･A･T  m=    5    n=    5

   i    j          real(a(i,j)(k))          imag(a(i,j)(k))
   1    1  (   2.00000000000000016 8e+00    0.00000000000000000e+00  )
   2    1  (  -3.62074829544800340 4e-32    0.00000000000000000e+00  )
   3    1  (  -4.41471291502164138 4e-31    0.00000000000000000e+00  )
   4    1  (   1.26201132877312769 2e-16    0.00000000000000000e+00  )
   5    1  (  -1.26201132877312729 1e-16    0.00000000000000000e+00  )
   1    2  (   1.00000000000000000e+00      0.00000000000000000e+00  )
   2    2  (   1.99999999999999916e+00      0.00000000000000000e+00  )
   3    2  (   0.00000000000000000e+00      0.00000000000000000e+00  )
   4    2  (   0.00000000000000000e+00      0.00000000000000000e+00  )
   5    2  (   0.00000000000000000e+00      0.00000000000000000e+00  )
   1    3  (   0.00000000000000000e+00      0.00000000000000000e+00  )
   2    3  (   1.00000000000000000e+00      0.00000000000000000e+00  )
   3    3  (   1.99999999999999916e+00      0.00000000000000000e+00  )
   4    3  (   0.00000000000000000e+00      0.00000000000000000e+00  )
   5    3  (   0.00000000000000000e+00      0.00000000000000000e+00  )
   1    4  (   2.51534904016637251 1e-16    0.00000000000000000e+00  )
   2    4  (   8.67361737988832762 0e-19    0.00000000000000000e+00  )
   3    4  (  -2.50667542278648677 6e-16    0.00000000000000000e+00  )
   4    4  (   9.99999999999999371e-01      0.00000000000000000e+00  )
   5    4  (  -3.86991548200310204 2e-33    0.00000000000000000e+00  )
   1    5  (   0.00000000000000000e+00      0.00000000000000000e+00  )
   2    5  (   0.00000000000000000e+00      0.00000000000000000e+00  )
   3    5  (   0.00000000000000000e+00      0.00000000000000000e+00  )
   4    5  (   1.00000000000000000e+00      0.00000000000000000e+00  )
   5    5  (   1.00000000000000062 9e+00    0.00000000000000000e+00  )

計算の条件

n=  5    itrmax=200
eps= 1.00000000e-30 epsz= 1.00000000e-10
```

Fig.4 Jordan標準形への変換計算例

9. まとめ

制御系の解析で良く用いられる複素数要素正方行列の Jordan 標準形への変換の計算法について論じた。C++プログラミングによりオブジェクト指向機能と演算子のオーバロード機能を用いることによりPCに本来備わっている80ビットの浮動小数点演算機能を有効に利用する長倍精度複素数クラスとそれを係数とする長倍精度複素数多項式クラスを作成した。この二つの型の導入により多項式演算が整数演算と同様な形態で簡単に実施できるような環境ができることを示した。これを基にして**C**-行列を Jordan 標準形への変換について証明の論述にのみ用いられていたλ-行列の基本操作に基づく単因子の作成を実際のプログラムとして簡単にコード化できることを示した。単因子とその微分多項式の最大公約式は Euclid の互除法を ecpolynom 型変数の上で行い簡単に求めらる。単因子と最大公約式の商は単根のみ有し、これに複素数 Newton-Raphson 法を適用してその全零点を安定に求めることができることを示した。また、多重度も微分系列から求まり単純単因子をすべて定めることができ、Jordan 標準形が定まることを示した。さらに根ベクトルの計算から Jordan 標準形への正則な相似変換行列も求められることを示した。このように従来は反復法を主体にして行われていた計算法とは異なり、C++というプログラム言語の特性を生かして計算法を提案した。いくつかの例を示し計算の有効性を示した。

最後に、この研究は平成7年以来著者が卒業研究のテーマとして何人かの卒業研究生のテーマとして検討してもらった結果をまとめたものであることを述べ、各卒業生への謝意を表したいと思う。

文献

(1) Golub,G.H, Wilkinson, J.H,"Ill-conditioned eigensystems and the computations of the Jordan canonical form",SIAM Review, 18. ,578-619, 1976

(2) ア・イ・マリチェフ「線型代数学Ⅰ」(柴岡泰光訳) 商工出版, 1960.3

(3) 森　正武、杉原正顕、室田一雄「線形計算」岩波講座応用数学[方法 2], 1994.2

(4) 吉田耕作「積分方程式論」岩波全書 117, 1950.10

(5) J Stear, R. Bulirsh,"Introduction to Numerical Analysis", Springer Verlag, 1991.5

(6) "Using MATLAB Version 5", Math Works, 1996.12

(7) 「Borland C++ プログラマーズガイド V5.0」Borland, 1996.8

(8) 高木貞次「代数学講義」共立出版, 1930.7

付録 1 長倍精度複素数演算ヘッダファイル ecomplex.h

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//    ecomplex.h  長倍精度複素数計算ヘッダファイル                    //
//                                                                    //
//     extended type                          1993-4-24    立花康夫   //
//                                            1996-10-26   改訂       //
//                                            1997-12-23   32bit 版   //
//                                                                    //
////////////////////////////////////////////////////////////////////////
//
class ecomplex {
     long double real; // 実数部分
     long double imag; // 虚数部分

     public:

     ecomplex() { real=0; imag=0; }; // 初期値を指定しないコンストラクタ

     ecomplex(long double initr, long double initi) {
          real=initr;
          imag=initi;
     }; // 指定コンストラクタ

     long double Getreal() { return real; };          // 実数部設定
     long double Getimag() { return imag; };          // 虚数部設定
     void Putreal(long double rval) { real=rval; };   // 実数部取り出し
     void Putimag(long double ival) { imag=ival; };   // 虚数部取り出し
     ecomplex conj() {                                // 共役作成
          return ecomplex(real, -imag);
     };

     friend long double real(ecomplex & arg);   // 実数部取り出し関数
     friend long double imag(ecomplex & arg);   // 虚数部取り出し関数
     friend ecomplex conj(ecomplex & arg);      // 共役作成関数
     friend long double absl(ecomplex & arg);   // 絶対値取り出し関数
     friend ecomplex expl(ecomplex & arg);      // 指数関数
     friend ecomplex cosl(ecomplex & arg);      // cos 関数
     friend ecomplex sinl(ecomplex & arg);      // sin 関数

     ecomplex operator +(ecomplex arg) {        // + オーバーロード
          return ecomplex(real+arg.real, imag+arg.imag);
     };
     ecomplex operator +(long double arg) {     // + オーバーロード
          return ecomplex(real+arg, imag);
     };
     ecomplex operator -(ecomplex arg) {        // - オーバーロード
          return ecomplex(real-arg.real, imag-arg.imag);
     };
     ecomplex operator -(long double arg) {     // - オーバーロード
          return ecomplex(real-arg, imag);
     };
     ecomplex operator *(ecomplex arg) {        // * オーバーロード
          return ecomplex(real*arg.real-imag*arg.imag,
                    real*arg.imag+imag*arg.real);
     };
     ecomplex operator *(long double arg) {     // * オーバーロード
          return ecomplex(real*arg, imag*arg);
     };
     ecomplex operator /(ecomplex arg) {        // / オーバーロード
          long double x;
          x=arg.real*arg.real+arg.imag*arg.imag;
          return ecomplex((real*arg.real+imag*arg.imag)/x,
             (imag*arg.real-real*arg.imag)/x);
     };
     ecomplex operator /(long double arg) {     // / オーバーロード
          return ecomplex(real/arg, imag/arg);
     };
     ecomplex operator +() {                    // +() オーバーロード
          return *this;
     };
     ecomplex operator -() {                    // -() オーバーロード
          return ecomplex(-real, -imag);
     };
     ecomplex operator +=(ecomplex arg) {       // += オーバーロード
          real+=arg.real;
          imag+=arg.imag;
          return ecomplex(real, imag);
     };
     ecomplex operator -=(ecomplex arg) {       // -= オーバーロード
          real-=arg.real;
          imag-=arg.imag;
          return ecomplex(real, imag);
     };
     ecomplex operator *=(ecomplex arg) {       // *= オーバーロード

          long double re;

          re=real*arg.real-imag*arg.imag;
          imag=real*arg.imag+imag*arg.real;
          real=re;

          return *this;
     };
     ecomplex operator /=(ecomplex arg) {       // /= オーバーロード

          long double re,u;

          u=arg.real*arg.real+arg.imag*arg.imag;
          re=(real*arg.real+imag*arg.imag)/u;
          imag=(-real*arg.imag+imag*arg.real)/u;
          real=re;

          return *this;
     };
     ecomplex & operator = (const long double &r) { // 代入演算子

          real=r;
          imag=0;

          return *this;
     };

};
//
//     実行部  implementation
//
#include <stdio.h>
#include <math.h>
//
inline long double real(ecomplex & arg) {   // 実数部取り出し関数

     return arg.real;
}

inline long double imag(ecomplex & arg) {   // 虚数部取り出し関数

     return arg.imag;
}

inline ecomplex conj(ecomplex & arg) {      // 共役作成関数

     return ecomplex(arg.real, -arg.imag);
}

inline long double absl(ecomplex & arg) {   // 絶対値取り出し関数

     return sqrtl(arg.real*arg.real+arg.imag*arg.imag);

}

inline ecomplex expl(ecomplex & arg) {      // 指数関数

     long double r,t;

     r=expl(arg.real);
     t=arg.imag;

     return ecomplex(r*cosl(t),r*sinl(t));
}

inline ecomplex cosl(ecomplex & arg) {      // cos 関数

     long double r,t,c,s,hc,hs;

     r=arg.real;
     t=arg.imag;

     c=cosl(r);
     s=sinl(r);
     hc=coshl(t);
    hs=sinhl(t);
```

```
    return ecomplex(c*hc,-s*hs);
}

inline ecomplex sinl(ecomplex & arg) {      //  sin 関数

    long double r,t,c,s,hc,hs;

    r=arg.real;
    t=arg.imag;

    c=cosl(r);
    s=sinl(r);
    hc=coshl(t);
    hs=sinhl(t);

    return ecomplex(s*hc,c*hs);
}
//
////////////////////////////////////////////////////////////////////////////
//////////////////////////  ECOMPLEX終了  //////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//
```

付録 2　長倍精度複素数多項式演算
ヘッダファイル ecpolynom.h

```
////////////////////////////////////////////////////////////////////////////
//                                                                        //
// ecpolynom.h     長倍精度複素係数多項式計算ヘッダファイル               //
//                                              1996-11-30 立花康夫       //
//                                                                        //
////////////////////////////////////////////////////////////////////////////
/
//
#pragma warn -inl               // インライン警報を無視
//
#include <stdio.h>
#include <math.h>
#include "ecomplex.h"
//
#define epsiron     1.e-15
#define MAXCHR      100         // 読み込み数字数最大値
#define MAXCHR1     101
#define _MAXPOLD    100         // 無指定多項式次数最大値
//
//
class ecpolynom {

    int         n;              // 次数
    ecomplex *a;                // 係数  a0･x^n+a1･x^(n-1)+･･･+an  a[i]=ai

    public:  // メンバ関数の定義

    ecpolynom(int iv=_MAXPOLD):n(iv) {    // 大きさを指定するコンストラクタ
        int i;                             // 汎用インデックス

        a=new ecomplex[n+1];
        for (i=0;i<=n;i++) {
            a[i]=0;
        }

    };

    ecpolynom(int iv, ecomplex av[]) {     // 初期値指定コンストラクタ
        int i;                             // 汎用インデックス

        n=iv;
        a=new ecomplex[n+1];
        for (i=0;i<=n;i++) {
            a[i]=av[i];
        }

    };

    ecpolynom(const ecpolynom &C) {        // コピーコンストラクタ

        int i;

        n=C.n;
        a=new ecomplex[n+1];
        for (i=0;i<=n;i++) {
            a[i]=C.a[i];
        }
};

ecpolynom & operator = (const ecpolynom &C) {  // 代入演算子

    int i;

    if (this==&C) return *this;

    delete[] a;

    n=C.n;
    a=new ecomplex[n+1];
    for (i=0;i<=n;i++) {
        a[i]=C.a[i];
    }
    return *this;
};

int  GetOrder() { return n; };                 // 次数取り出し

ecomplex GetCoef(int iv) { return a[iv]; };    // 係数取り出し

void Arrange() {                               // 最高次係数を 1 とする。

    int i;
    long double u,v;
    ecomplex    c;

    u=sqrtl(real(a[0]*a[0].conj()));
    c=a[0];
    if (u>epsiron) {
        for (i=0;i<=n;i++) {
            a[i]=a[i]/c;
        }
    }
}

long double GetAbslCoef() {                    // 係数の自乗の和の平方根

   int i;
   long double u,v,w;

   u=0;
   for (i=0;i<=n;i++) {
       v=real(a[i]*a[i].conj());
       u+=v;
   }
   w=sqrtl(u);

   return w;

};

long double GetEps() {

    long double eps;

    eps=epsiron;
    return eps; }                              // ε取り出し

void PutOrder(int nv) { n=nv; };               // 次数設定
void PutCoef(int iv, ecomplex av) { a[iv]=av;};    // 係数設定
void PutCoef(int iv, long double xv) { a[iv]=ecomplex(xv,0); }

ecpolynom ecpolynom::Proper();                 // プロパー化

ecomplex ecpolynom::GetValue(ecomplex x);
                                               // 多項式の値取り出し
ecomplex ecpolynom::GetValue(long double x);
                                               // 多項式の値取り出し


ecpolynom operator +(ecpolynom & arg);         // + オーバーロード
ecpolynom operator -(ecpolynom & arg);         // - オーバーロード
ecpolynom operator *(ecpolynom & arg);         // * オーバーロード
ecpolynom operator /(ecpolynom & arg);         // / オーバーロード
ecpolynom operator %(ecpolynom & arg);         // % オーバーロード

friend void PolProper(int n, int *nv, ecomplex _ar[], long double eps);
                                               // プロパー化ルーチン

friend ecpolynom operator * (ecomplex scalar, ecpolynom & A);
                                               // scalar * ecpolynom

friend int GetOrder( ecpolynom & arg);         // 次数取り出し
friend ecomplex GetCoef( ecpolynom & arg, int iv);
                                               // 係数ベクトル取り出し

friend void PutOrder(ecpolynom & arg, int iv); // 次数設定
friend void PutCoef(ecpolynom & arg, int iv);
                                               // 係数ベクトル設定
friend ecpolynom Proper(ecpolynom & arg);      // プロパー化関数
friend ecomplex GetValue(ecpolynom &arg, ecomplex x);
                                               // 多項式の値取り出し関数

friend ecomplex GetValue(ecpolynom &arg, long double x);
                                               // 多項式の値取り出し関数

friend void QuotResd(ecpolynom &F, ecpolynom &G, ecpolynom &Q, ecpolynom &R);
```

```
                                        // F=G*Q+R 商と余り計算
    virtual ~ecpolynom() { delete[] a; };       // デストラクタ

};
//
//      実行部
//
//      内部関数定義
//
inline _RTLENTRY void PolProper(int n, int *nv, ecomplex _ar[], long double eps) {

    int           i;                    // 汎用インデックス
    long double   u;

    *nv=0;                              // プロパー化処理
    for (i=0;i<=n;i++) {
    u=sqrt(real(_ar[i]*conj(_ar[i])));
        if (u>eps) {
            *nv=n-i;
            break;
        }
    }
    for (i=0;i<=*nv;i++) {
        _ar[i]=_ar[i+n-*nv];
    }
}

inline _RTLENTRY ecpolynom  ecpolynom::Proper() {   // プロパー多項式への変換

    int i;                              // 汎用インデックス
    int nv;
    long double eps=epsiron;
    ecomplex    *_ar;                   // 作業領域

    ar=new ecomplex[n+1];

    for (i=0;i<=n;i++) {
        _ar[i]=a[i];
    }

    PolProper(n,&nv,_ar,eps);

    return ecpolynom(nv, _ar);

}

inline _RTLENTRY ecomplex ecpolynom::GetValue(ecomplex x) {
                                        // 多項式の値取り出し
    int           i;                    // 汎用インデックス
    ecomplex      s;                    // 中間変数

    s=ecomplex(0,0);
    for (i=0;i<=n;i++) {
        s=s*x+a[i];
    }

    return s;

}

inline _RTLENTRY ecomplex ecpolynom::GetValue(long double x) {
                                        // 多項式の値取り出し
    int           i;                    // 汎用インデックス
    ecomplex      s;                    // 中間変数

    s=ecomplex(0,0);
    for (i=0;i<=n;i++) {
        s=s*ecomplex(x,0)+a[i];
    }

    return s;

}

inline _RTLENTRY ecpolynom ecpolynom::operator +(ecpolynom & _arg) {
                                        // + オーバロード
    int         i;                      // 汎用インデックス
    int         nv,nr;
    long double eps=epsiron;
    ecomplex    *_ar;                   // 作業領域

    nv=n;
    if (n<_arg.n) { nv=_arg.n; }

    _ar=new ecomplex[nv+1];

    if (n>_arg.n) {
        for (i=0;i<=n;i++) {
            _ar[n-i]=a[n-i];
            if (i<=_arg.n) {
                _ar[n-i]+=_arg.a[_arg.n-i];
            }
        }
    } else {
        for (i=0;i<=nv;i++) {
            _ar[nv-i]=_arg.a[nv-i];
            if (i<=n) {
                _ar[nv-i]+=a[n-i];
            }
        }
    }

    PolProper(nv,&nr,_ar,eps);

    return ecpolynom(nr, _ar);

}

inline _RTLENTRY ecpolynom ecpolynom::operator -(ecpolynom & _arg) {
                                        // - オーバーロード
    int         i;                      // 汎用インデックス
    int         nv,nr;
    long double eps=epsiron;
    ecomplex    *_ar;                   // 作業領域

    nv=n;
    if (n<_arg.n) { nv=_arg.n; }

    _ar=new ecomplex[nv+1];

    if (n>_arg.n) {
        for (i=0;i<=n;i++) {
            _ar[n-i]=a[n-i];
            if (i<=_arg.n) {
                _ar[n-i]=_ar[n-i]-_arg.a[_arg.n-i];
            }
        }
    } else {
        for (i=0;i<=nv;i++) {
            _ar[nv-i]=-_arg.a[nv-i];
            if (i<=n) {
                _ar[nv-i]=_ar[nv-i]+a[n-i];
            }
        }
    }

    PolProper(nv,&nr,_ar,eps);

    return ecpolynom(nr, _ar);

}

inline _RTLENTRY ecpolynom ecpolynom::operator *(ecpolynom & _arg) {
                                        // * オーバーロード
    int         i,j,k;                  // 汎用インデックス
    int         ns,nv;
    long double eps=epsiron;
    ecomplex    s;                      // 中間実数
    ecomplex    *_ar;                   // 作業領域

    nv=n+_arg.n;

    _ar=new ecomplex[nv+1];

    for (k=0;k<=nv;k++) {
        s=ecomplex(0,0);
        for (j=0;j<=nv;j++) {
            i=k-j;
            if (((i>=0) && (i<=n)) && (j<=_arg.n)) {
                s=s+a[i]*_arg.a[j];
            }
        }
        _ar[k]=s;
    }

    ns=nv;
    PolProper(ns,&nv,_ar,eps);

    return ecpolynom(nv, _ar);

}

inline _RTLENTRY ecpolynom ecpolynom::operator /(ecpolynom & _arg) {
                                        // / オーバーロード
    int         i,j;                    // 汎用インデックス
    int         ns,nv;
    long double eps=epsiron;
    long double r,t;
    ecomplex    s;
    ecomplex    *u;
    ecomplex    *_ar;                   // 作業領域
    char swindows[MAXCHR1];             //  実行経過メッセージ

    _ar=new ecomplex[n+1];
    u=_arg.a;
    PolProper(_arg.n,&ns,u,eps);
    t=real(u[0]*conj(u[0]));
    if (t<0) { t=0; }
    r=sqrt(t);
    if (r>eps) {

        if (n>=ns) {
            nv=n-ns;
            for (i=0;i<=nv;i++) {
                _ar[i]=ecomplex(0,0);
            }
            for (i=0;i<=nv;i++) {
                s=ecomplex(0,0);
                for (j=0;j<=i;j++) {
                    if (j<=ns) {
                        s=s+u[j]*_ar[i-j];
                    }
                }
                _ar[i]=(a[i]-s)/u[0];
            }
        } else {
            nv=0;
            _ar[0]=ecomplex(0,0);
        }

    } else {
        nv=0;
        _ar[0]=ecomplex(0,0);
        MessageBox(NULL,"多項式のわり算で0割が発生しました。","ecpolynom",
                        MB_OK | MB_ICONEXCLAMATION);
    }

    ns=nv;
    PolProper(ns,&nv,_ar,eps);

    return ecpolynom(nv, _ar);

}

inline _RTLENTRY ecpolynom ecpolynom::operator %(ecpolynom & _arg) {
                                        // % オーバーロード

                                        // / オーバーロード
    int         i,j;                    // 汎用インデックス
    int         ns,nv;
    long double eps=epsiron;
    long double r;
    ecomplex s;
    ecomplex *u;
    ecomplex *_ar;                      // 作業領域
    char swindows[MAXCHR1];             //  実行経過メッセージ
```

```
    _ar=new ecomplex[n+1];
    u=_arg.a;
    PolProper(_arg.n,&ns,u,eps);

    r=sqrt(real(u[0]*conj(u[0])));
    if (r>eps) {

        if (n>=ns) {
            if (ns>0) {
                nv=ns-1;
                for (i=0;i<=n;i++) {
                    _ar[i]=a[i];
                }
                for (i=0;i<=n-ns;i++) {
                    s=_ar[0]/u[0];
                    for (j=0;j<=n-i-1;j++) {
                        _ar[j]=_ar[j+1]-s*u[j+1];
                    }
                }
            } else {
                nv=0;
                _ar[0]=ecomplex(0,0);
            }
        } else {
            nv=n;
            for (i=0;i<=n;i++) {
                _ar[i]=a[i];
            }
        }

    } else {
        nv=n;
        for (i=0;i<=n;i++) {
            _ar[i]=a[i];
        }
        MessageBox(NULL,swindows,"多項式のわり算で0割が発生しました。",
                        MB_OK | MB_ICONEXCLAMATION);
    }

    ns=nv;
    PolProper(ns,&nv,_ar,eps);

    return ecpolynom(nv, _ar);


}
inline _RTLENTRY ecpolynom operator * (ecomplex scalar, ecpolynom & A) {
                                                    // スカラ*多項式

    int  i;                         // 汎用インデックス
    int  ns,nv;
    long double eps=epsiron;
    ecomplex *_ar;                  // 作業領域

    _ar=new ecomplex[A.n+1];

    nv=A.n;
    for (i=0;i<=nv;i++) {
        _ar[i]=scalar*A.a[i];
    }

    ns=nv;
    PolProper(ns,&nv,_ar,eps);

    return ecpolynom(nv,_ar);
};

inline _RTLENTRY int GetOrder(ecpolynom & arg) {
                                                    // 次数取り出し
    return arg.n;
}

inline _RTLENTRY ecomplex GetCoef(ecpolynom & arg, int i) {
                                                    // 係数ベクトル取り出し
    ecomplex x;

    x=arg.GetCoef(i);

    return x;
}

inline _RTLENTRY void PutOrder(ecpolynom & arg, int iv) {
                                                    // 次数設定
    arg.n=iv;

}

inline _RTLENTRY void PutCoef(ecpolynom & arg, int i, ecomplex x) {
                                                    // 係数ベクトル設定
    arg.PutCoef(i,x);

}

inline _RTLENTRY ecpolynom Proper(ecpolynom & arg) {
                                                    // プロパー化関数
    return arg.Proper();
}

inline _RTLENTRY ecomplex GetValue(ecpolynom &arg, ecomplex x) {
                                                    // 多項式の値取り出し関数
    return arg.GetValue(x);
}

inline _RTLENTRY ecomplex GetValue(ecpolynom &arg, long double x) {
                                                    // 多項式の値取り出し関数
    return arg.GetValue(x);
}

inline _RTLENTRY void QuotResd(ecpolynom &F,ecpolynom &G,ecpolynom &Q,
                                                    ecpolynom &R);
                                                    // F=G*Q+R 商と余り計算

//
////////////////////////////////////////////////////////////////////////
//                      ecpolynom.h終了 //
////////////////////////////////////////////////////////////////////////
```