JAPLA 研究会資料 2007/5/26

Ｊの正規表現プログラミング III

記号処理により行列式を直接展開して固有値を求める

西川　利男

０．はじめに

先月の JAPLA 例会で、中野嘉弘氏により「J 言語と高等数学－固有値問題（直接法）を主に」と題して、固有値問題の特性方程式を直接求めるダニレフスキー法が紹介された[1]。この方法により特性方程式として高次多項方程式が得られれば、J の強力な関数 p. を用いてその根は直ちに求めることが出来る。また、行列の対称、非対称を考慮することも不要であり、非対称行列にも有効である。この方法を中野ネフスキー法とも名付け、従来からの固有値の計算法であるべき乗法、ヤコビ法、QR 法などに代わってもっと使われるべきと主張されているが、筆者も同感である。

筆者は先に J の正規表現を利用した数式処理を報告したが[2]、ダニレフスキー法によらず、記号処理のみによってもっと直接的に行列式を展開する方法を開発した。

１．数学における記号処理の重要性

はじめに数学における記号処理の意義につき少し述べておきたい。およそ数学演算なるものの実際の内容は一般に考えられているような計算ではなく、もっぱら記号の操作であると筆者は考えている。これを一覧として示すと次のようになる。

・数値の計算……加減乗除、平方根、立方根、…
・記号の構造操作
　・代入、同類項の簡約
　・カッコ操作（演算順序の変更）…カッコでくくる、カッコをほどく
　　年配の方であれば、「考え方」数学での「ビンヅメ」、「カンヅメ」の語を思い出すであろう。
　・因数分解　　｜
　・行列、行列式　　｜　これらは計算ではなく、記号の操作である！！
　・微分　　｜
　・複素数　　｜

ここであらためて「代数」なる語を考えてみる。「数を何に代えるのか？－文字（記号）に代える」のである。これにより数学は長足の進歩をとげることが出来たのである。

２．行列式の直接展開による特性方程式と固有値

行列の固有値を求める問題は、次のような特性行列式を解くことに帰着する。
例えば3×3の場合はつぎのようになる。

$$\begin{vmatrix} a_{11}-x & a_{12} & a_{13} \\ a_{21} & a_{22}-x & a_{23} \\ a_{31} & a_{32} & a_{33}-x \end{vmatrix} = 0$$

この行列式は、余因子小行列式(cofactor minor)を用いて、次のように展開される。これはラプラス展開と呼ばれる。

$$(a_{11}-x)\begin{vmatrix} a_{22}-x & a_{23} \\ a_{32} & a_{33}-x \end{vmatrix} - (a_{12})\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33}-x \end{vmatrix} + (a_{13})\begin{vmatrix} a_{21} & a_{22}-x \\ a_{31} & a_{32} \end{vmatrix} = 0$$

そして、これにより得られる x の高次多項方程式

$$(a_{11}-x)(a_{22}-x)(a_{33}-x) - (a_{11}-x)(a_{23})(a_{32}) - (a_{12})(a_{21})(a_{33}-x) + (a_{12})(a_{23})(a_{31}) + (a_{13})(a_{21})(a_{32}) - (a_{13})(a_{22}-x)(a_{31}) = 0$$

を解くことで固有値は得られる。

Jにおいては高次多項方程式は強力な関数p.を使えば容易に求められる。したがって前半の行列式から多項式の展開をどうすればよいか、にかかっている。中野嘉弘氏の方法はダニレフスキーのアルゴリズムによってこの展開を行うものである。

先の行列式の a_{11}, a_{12}, ‥は数値であり、x は未知数を示す文字であり、このままではJでプログラミングするわけには行かない。

３．記号処理による直接展開法－考え方と原理

筆者の直接展開法は次のとおりである。

(1) はじめにすべて文字だけから成る次のような行列式

$$\begin{vmatrix} A\ B\ C \\ D\ E\ F \\ G\ H\ I \end{vmatrix}$$

に対して展開を行い、高次多項式を表示する一種の文字列を得る。

(2) この展開式は次数が高くなるにしたがい、多重のカッコとぼう大な数の項から成ることになり、これをどう処理するかが決め手になる。筆者の方法はこれを正規表現などを使った記号処理で行った。

(3) 上で得られた展開式において、各文字に対して

$A = a_{11} - x$, $B = a_{12}$, $C = a_{13}$,..

なる文字列の置き換えを行う。つまりこのとき、a_{11}, a_{12}, ‥はたとえば、'2', '3.5', ‥のような数値を表す文字列である。

(4) 再び生じる多重カッコの取りはずしと項の整理などの前処理を行った後、数式としての高次多項式のべき乗カッコ、同類項の整理、簡約には筆者が発表したＪの数式処理システム formulax.ijs のプログラムにより行った。
(5) 最後に、上で得られた多項式の係数部分を取り出し、数値を表す文字列を実行(".)により数値に戻し、関数 p. により多項方程式の解を求めて、これを固有値とする。

４．Ｊのプログラムの実際と途中経過

４．１　文字行列式の展開

まず、文字の行列式を次のようにして作る。

```
A_Z =: (65+i.26){a.
a_z =: (97+i.26){a.
Alphabet =: A_Z, a_z

X =: 2 2$Alphabet
Y =: 3 3$Alphabet
Z =: 4 4$Alphabet
W =: 5 5$Alphabet
V =: 6 6$Alphabet
```

たとえば次のようになる。

```
   Y
ABC
DEF
GHI
   Z
ABCD
EFGH
IJKL
MNOP
```

後に示す関数 expand により展開すると次のようになる。

```
   expand Y
+A(EI-FH)-B(DI-FG)+C(DH-EG)
   expand Z
+A(+F(KP-LO)-G(JP-LN)+H(JO-KN))-B(+E(KP-LO)-G(IP-LM)+H(IO-KM))+C(+E(JP-LN)
-F(IP-LM)+H(IN-JM))-D(+E(JO-KN)-F(IO-KM)+G(IN-JM))
```

このように次数が高くなるにしたがい、多重カッコと項数はぼう大に増える。したがって多重カッコの取り外しが最大の難関となる。

ここで関数 expand は以下のように定義される。なお関数 cofact は余因子小行列式を取り出すサブプログラムである。

```
NB. Expansion of Determinant represented in Alphabet Characters
expand =: 3 : 0
M =. y.
N =. {. $ M
if. N = 2
  do.
    (((<0,0){M),(<1,1){M),'-',(((<0,1){M),(<1,0){M)
    return.
  else.
    j =. 0
    P =. ''
    while. j < N
      do.
        if. 2|j
          do.
            P =. P,'-', ((<0, j){M), '(',(expand (0,j) cofact M), ')'
          else.
            P =. P,'+', ((<0, j){M), '(',(expand (0,j) cofact M), ')'
          end.
        j =. j + 1
        end.
    P
end.
)

cofact =: 3 : 0         NB. cofactor minor eg. (2,2) cofact Z
:
(<<<0{x.){(<<<1{x.){"1 y.
)
```

関数 expand の実行は再帰呼び出しで行われるので、行列の次数にかかわりなく実行される。

４．２　多重カッコの取り外し

まずは関数 expdet により、多重カッコをはずした結果を見てみよう。

```
   expdet X  NB. 2×2
AD-BC
   expdet Y  NB. 3×3
+AEI-AFH-BDI+BFG+CDH-CEG
   expdet Z  NB. 4×4
+AFKP-AFLO-AGJP+AGLN+AHJO-AHKN-BEKP+BELO+BGIP-BGLM-BHIO+BHKM+CEJP-CELN-CFIP
+CFLM+CHIN-CHJM-DEJO+DEKN+DFIO-DFKM-DGIN+DGJM
   expdet W  NB. 5×5
+AGMSY-AGMTX-AGNRY+AGNTW+AGORX-AGOSW-AHLSY+AHLTX+AHNQY-AHNTV-
AHOQX+AHOSV+AILRY-AILTW-AIMQY+AIMTV+AIOQW-AIORV-AJLRX+AJLSW+AJMQX-AJMSV-
AJNQW+AJNRV-BFMSY+BFMTX+BFNRY-BFNTW-BFORX+BFOSW+BHKSY-BHKTX-
BHNPY+BHNTU+BHOPX-BHOSU-BIKRY+BIKTW+BIMPY-BIMTU-BIOPW+BIORU+BJKRX-BJKSW-
BJMPX+BJMSU+BJNPW-BJNRU+CFLSY-CFLTX-CFNQY+CFNTV+CFOQX-CFOSV-
CGKSY+CGKTX+CGNPY-CGNTU-CGOPX+CGOSU+CIKQY-CIKTV-CILPY+CILTU+CIOPV-CIOQU-
CJKQX+CJKSV+CJLPX-CJLSU-CJNPV+CJNQU-DFLRY+DFLTW+DFMQY-DFMTV-
DFOQW+DFORV+DGKRY-DGKTW-DGMPY+DGMTU+DGOPW-DGORU-DHKQY+DHKTV+DHLPY-DHLTU-
DHOPV+DHOQU+DJKQW-DJKRV-DJLPW+DJLRU+DJMPV-DJMQU+EFLRX-EFLSW-
EFMQX+EFMSV+EFNQW-EFNRV-EGKRX+EGKSW+EGMPX-EGMSU-EGNPW+EGNRU+EHKQX-EHKSV-
EHLPX+EHLSU+EHNPV-EHNQU-EIKQW+EIKRV+EILPW-EILRU-EIMPV+EIMQU
   expdet V  NB. 6×6
+AHOVcj-AHOVdi-AHOWbj+AHOWdh+AHOXbi-AHOXch-AHPUcj+AHPUdi+AHPWaj-AHPWdg-
AHPXai+AHPXcg+AHQUbj-AHQUdh-AHQVaj+AHQVdg+AHQXah-AHQXbg-AHRUbi+AHRUch+AHRVai-
AHRVcg-AHRWah+AHRWbg-AINVcj+AINVdi+AINWbj-AINWdh-AINXbi+AINXch+AIPTcj-AIPTdi-
AIPWZj+AIPWdf+AIPXZi-AIPXcf-AIQTbj+AIQTdh+AIQVZj-AIQVdf-AIQXZh+AIQXbf+AIRTbi-
AIRTch-AIRVZi+AIRVcf+AIRWZh-AIRWbf+AJNUcj-AJNUdi-AJNWaj+AJNWdg+AJNXai-AJNXcg-
AJOTcj+AJOTdi+AJOWZj-AJOWdf-AJOXZi+AJOXcf+AJQTaj-AJQTdg-AJQUZj+AJQUdf+AJQXZg-
AJQXaf-AJRTai+AJRTcg+AJRUZi-AJRUcf-AJRWZg+AJRWaf-AKNUbj+AKNUdh+AKNVaj-AKNVdg-
AKNXah+AKNXbg+AKOTbj-AKOTdh-AKOVZj+AKOVdf+AKOXZh-AKOXbf-AKPTaj+AKPTdg+AKPUZj-
AKPUdf-AKPXZg+AKPXaf+AKRTah-AKRTbg-AKRUZh+AKRUbf+AKRVZg-AKRVaf+ALNUbi-ALNUch-
ALNVai+ALNVcg+ALNWah-ALNWbg-ALOTbi+ALOTch+ALOVZi-ALOVcf-ALOWZh+ALOWbf+ALPTai-
ALPTcg-ALPUZi+ALPUcf+ALPWZg-ALPWaf-ALQTah+ALQTbg+ALQUZh-ALQUbf-ALQVZg+ALQVaf-
BGOVcj+BGOVdi+BGOWbj-BGOWdh-BGOXbi+BGOXch+BGPUcj-BGPUdi-BGPWaj+BGPWdg+BGPXai-
BGPXcg-BGQUbj+BGQUdh+BGQVaj-BGQVdg-BGQXah+BGQXbg+BGRUbi-BGRUch-
BGRVai+BGRVcg+BGRWah-BGRWbg+BIMVcj-BIMVdi-BIMWbj+BIMWdh+BIMXbi-BIMXch-
```

```
BIPScj+BIPSdi+BIPWYj-BIPWde
```
（以下２ページにわたって続くが、省略する）

関数 expdet の全体の定義は巻末のプログラム・リストを見ていただくとして、4×4 の文字行列式 Z について、その処理の過程を見てみる。

```
   Z
ABCD
EFGH
IJKL
MNOP
```
まず、関数 expand により行列式を展開する。

```
   y40 =. expand Z
   y40
+A(+F(KP-LO)-G(JP-LN)+H(JO-KN))-B(+E(KP-LO)-G(IP-LM)+H(IO-KM))+C(+E(JP-LN)-
F(IP-LM)+H(IN-JM))-D(+E(JO-KN)-F(IO-KM)+G(IN-JM))
```
次にパターン pat で検索された部分だけのすべてにわたって、カッコをほどく関数 depar を J の正規表現関数 rxapply により作用させ、内部のカッコをほどく。

例えば、

   +F(KP-LO) は +FKP-FLO に、-G(JP-LN) は -GJP+GLN に

のように変換する。ここに正規表現のパターン認識が効果的に使われている。

```
   pat =: '[¥+-][[:alpha:]]¥([[:alpha:]][[:alpha:]]-[[:alpha:]][[:alpha:]]¥)'
   y41 =. pat depar rxapply y40
   y41
+A(+FKP-FLO-GJP+GLN+HJO-HKN)-B(+EKP-ELO-GIP+GLM+HIO-HKM)+C(+EJP-ELN-
FIP+FLM+HIN-HJM)-D(+EJO-EKN-FIO+FKM+GIN-GJM)
```
今度は右カッコ')'を区切り文字としてボックスで区切る。これには行頭に')'を付加し、ボックスカット <;._1 で区切り、区切り文字で失われたカッコ')'を補充する。

```
   y42 =. (<;._1 ')', y41),L:0 ')' y41
   y42
+-------------------------+-------------------------
+-----------------|+A(+FKP-FLO-GJP+GLN+HJO-HKN)|-B(+EKP-ELO-GIP+GLM+HIO-
HKM)|+C(+EJP-ELN-FIP+FL
+-------------------------+-------------------------+-----------------
---------+-------------------------+-+
M+HIN-HJM)|-D(+EJO-EKN-FIO+FKM+GIN-GJM)|)|
---------+-------------------------+-+
```
さらに L:0 機能によりボックス内のそれぞれに対してカッコをほどく関数 deparn を

作用させ、最後に ； によりすべてのボックスをはずせば結果が得られる。

```
   y43 =. ; deparn L:0 y42
   y43
+AFKP-AFLO-AGJP+AGLN+AHJO-AHKN-BEKP+BELO+BGIP-BGLM-BHIO+BHKM+CEJP-CELN-
CFIP+CFLM+CHIN-CHJM-DEJO+DEKN+DFIO-DFKM-DGIN+DGJM
```

２種類のカッコをほどく関数 depar, deparn の定義はプログラムリストにあげてある。

４．３　文字への入力数値データの置き換え（assignment）

中野嘉弘氏の 4×4 の数値例でやってみよう。データの入力を数学の表記と合わせた形で出来るようにして、それを変換する関数 makedet を作った。（プログラムは巻末）

```
  nda4 =: ] ;._2 (0 : 0)
  2  1  3  4
  1 _3  1  5
  3  1  6 _2
  4  5 _2 _1
)
   NDA4 =: makedet nda4
   NDA4
+-----+------+-----+------+
|(2-x)|(1)   |(3)  |(4)   |
+-----+------+-----+------+
|(1)  |(-3-x)|(1)  |(5)   |
+-----+------+-----+------+
|(3)  |(1)   |(6-x)|(-2)  |
+-----+------+-----+------+
|(4)  |(5)   |(-2) |(-1-x)|
+-----+------+-----+------+
```

置換の関数 assign はアルファベットに各データを置き換えるだけで、簡単にループで行っている。

```
assign =: 3 : 0
:
x =. x.
i =. 0
y =. , y.
while. i < #y
```

```
  do.
     x =. ((i{Alphabet);(>i{y)) rxrplc x
     i =. i + 1
end.
wr 'initial_data=', x
x1 =. numbox x
x2 =. numcalc L:0 x1
wr 'det(x)=', x3 =. ; x2
x3
)
```

実行の途中経過を見てみよう。

```
   ZZ =: expdet Z
   ZZ assign NDA4
iniial_data=       +(2-x)(-3-x)(6-x)(-1-x)-(2-x)(-3-x)(-2)(-2)-(2-x)(1)(1)(-1-
x)+(2-x)(1)(-2)(5)+(2-x)(5)(1)(-2)-(2-x)(5)(6-x)(5)-(1)(1)(6-x)(-1-
x)+(1)(1)(-2)(-2)+(1)(1)(3)(-1-x)-(1)(1)(-2)(4)-(1)(5)(3)(-2)+(1)(5)(6-
x)(4)+(3)(1)(1)(-1-x)-(3)(1)(-2)(5)-(3)(-3-x)(3)(-1-x)+(3)(-3-x)(-
2)(4)+(3)(5)(3)(5)-(3)(5)(1)(4)-(4)(1)(1)(-2)+(4)(1)(6-x)(5)+(4)(-3-x)(3)(-
2)-(4)(-3-x)(6-x)(4)-(4)(1)(3)(5)+(4)(1)(1)(4)
```

ところでここで次のステップである数式処理システム formulax を利用するためには,各項を

+ または - の符号　数値　カッコのべき乗

という形にしなければならない。そのための数値部分の掛け算と項の入れ替えの関数が numcalc である。また、その準備として前と同様のボックスカットが必要となり、そのための関数 numbox を作った。

ここでは ')+(' または ')-(' のような文字列を検索した上で、この中の右カッコ')'でボックスカットを行う。そのためには')'の次に文字'!'を挿入し、文字'!'でボックスカットをすればよい。ここでも正規表現処理が有効に利用された。

```
NB. boxcut by at ')' in either pattern ')+(' or ')-('
numbox =: 3 : 0
y =. <;._1 '!', '¥)[¥+-]¥(' ({. , '!'&, @}.) rxapply  y.
)
```

このような処理を行った結果、次のような数式が得られた。

```
det(x)=     +(2-x)(-3-x)(6-x)(-1-x)-4(2-x)(-3-x)-1(2-x)(-1-x)-10(2-x)-10(2-x)-
25(2-x)(6-x)-1(6-x)(-1-x)+4+3(-1-x)+8+30+20(6-x)+3(-1-x)+30-9(-3-x)(-1-x)-
```

```
24(-3-x)+225-60+8+20(6-x)-24(-3-x)-16(-3-x)(6-x)-60+16
+(2-x)(-3-x)(6-x)(-1-x)-4(2-x)(-3-x)-1(2-x)(-1-x)-10(2-x)-10(2-x)-25(2-x)(6-
x)-1(6-x)(-1-x)+4+3(-1-x)+8+30+20(6-x)+3(-1-x)+30-9(-3-x)(-1-x)-24(-3-x)+225-
60+8+20(6-x)-24(-3-x)-16(-3-x)(6-x)-60+16
```

４．４　特性方程式の計算

特性方程式を求める関数は charpoly として定義され、次のように数式処理システム formulax を実行することで、計算される。

```
NB. Characteristic Polynomial
NB. using formulax.ijs
charpoly =: 3 : 0
select. {. $ y.
  case. 2 do. chp =. XX assign y.
  case. 3 do. chp =. YY assign y.
  case. 4 do. chp =. ZZ assign y.
  case. 5 do. chp =. WW assign y.
  case. 6 do. chp =. VV assign y.
end.
R =. run chp    NB. execute run of formulax
wr 'Characteristic Polynomial = ', R
RR              NB. RR is global return of formulax
)
   charpoly NDA4
Characteristic Polynomial = 568+260x-73x^2-4x^3+x^4
568+260x-73xx-4xxx+1xxxx+
```

４．５　固有値の計算

最後に以下の関数 eigen により、固有値が求められる。すなわち formulax の結果（グローバル戻り値）を元に係数を取り出し、J の関数 p. により多項式の根を求めて、最終結果の固有値とする。

```
eigen =: 3 : 0
chapol =. charpoly y.
y =. excoef L:0 sign_cut chapol
eig =. p. ; ". L:0 y
'Eigen Values = ', ": ; 1{eig
```

```
)

NB. sign_cut '8+9x-2x^2+x^3-5x^5'
sign_cut =: 3 : 0
<;._1 '!', ('[¥+-]') ('!'&(,])) rxapply y.
)

NB. extract coefficients
excoef =: 3 : 0
('x' i.~ y.) {. y.
)

   eigen NDA4
Characteristic Polynomial = 568+260x-73x^2-4x^3+x^4
Eigen Values = _8.02858 7.9329 5.66886 _1.57319
```

５．いくつかの実行例

中野嘉弘氏の報告[1]にある数値例を利用させていただく。
筆者の関数 eigen をそれぞれの数値データに実行した結果は次のとおりである。

```
   NDA2
+-----+-----+
|(3-x)|(2)  |
+-----+-----+
|(4)  |(1-x)|
+-----+-----+
   eigen NDA2
Characteristic Polynomial = -5-4x+x^2
Eigen Values = 5 _1

   NDA3
+-----+-----+------+
|(1-x)|(2)  |(2)   |
+-----+-----+------+
|(2)  |(0-x)|(0)   |
+-----+-----+------+
|(2)  |(1)  |(-1-x)|
```

```
+-----+-----+------+
```

eigen NDA3

Characteristic Polynomial = 8+9x-x^3

Eigen Values = 3.37228 _2.37228 _1

NDA5

```
+-----+-----+-----+-----+-----+
|(2-x)|(0)  |(0)  |(1)  |(1)  |
+-----+-----+-----+-----+-----+
|(0)  |(2-x)|(0)  |(0)  |(0)  |
+-----+-----+-----+-----+-----+
|(3)  |(2)  |(0-x)|(0)  |(0)  |
+-----+-----+-----+-----+-----+
|(0)  |(-1) |(0)  |(3-x)|(0)  |
+-----+-----+-----+-----+-----+
|(-1) |(1)  |(1)  |(0)  |(0-x)|
+-----+-----+-----+-----+-----+
```

eigen NDA5

Characteristic Polynomial = 18-21x+20x^2-17x^3+7x^4-x^5

Eigen Values = 3 2.17456 2 _0.0872797j1.17131 _0.0872797j_1.17131

NDA6

```
+-----+-----+-----+-----+------+-----+
|(0-x)|(1)  |(0)  |(0)  |(0)   |(0)  |
+-----+-----+-----+-----+------+-----+
|(-2) |(0-x)|(1)  |(1)  |(0)   |(3)  |
+-----+-----+-----+-----+------+-----+
|(1)  |(1)  |(0-x)|(0)  |(0)   |(-1) |
+-----+-----+-----+-----+------+-----+
|(1)  |(0)  |(0)  |(0-x)|(0)   |(-1) |
+-----+-----+-----+-----+------+-----+
|(-1) |(0)  |(4)  |(2)  |(-1-x)|(1)  |
```

```
+-----+-----+-----+-----+------+-----+
|(0)  |(1)  |(0)  |(0)  |(0)   |(0-x)|
+-----+-----+-----+-----+------+-----+
   eigen NDA6
Characteristic Polynomial = 0-2x^3-2x^4+x^5+x^6
Eigen Values = 1.41421 _1.41421 _1 0 0 0
```

６．おわりに

行列式の直接展開、カッコ処理において、Jの正規表現の強力さをますます実感した。とくに

あるパターンの文字列の並びを取り出し、
それをボックス化し、
L:0機能(これも極めて有能！)で、それぞれまとめて処理する

という手法は、Jの並行処理をいかんなく発揮し、他の言語ではなし得ない強力なプログラミング手法である。

また、中野嘉弘氏のいわく、『固有値の計算には、「べき乗法、ヤコビ法、QR法」など「えらそうな名前のついた解法」を使うことなく、線形代数の教科書に即つながった「判り切った直接法」を復活させたい』と、まことにそのとおりだと思う。

筆者にいわせれば、固有値問題などというが、これは平方根を求めるのと何の違いもなく、決して特殊な高級なものではない。これらを通じて、線形代数を数学者の手から一般の工学、経済などへの便利な道具として、もっと身近かなものにしてもらいたいと願っている。

文献

[1] 中野嘉弘「J言語と高等数学－固有値問題(直接法)を主に」JAPLA資料2007/4/28
[2] 西川利男「Jの正規表現プログラミングII－数式処理システムへの利用」
JAPLAシンポジウム2004/12/11

（発表直前の改良修正について、次ページに追加します）

７．文字行列式の展開の改良版

プログラムの最終チェック過程において、基本部分に大きな改良点が見つかった。原稿のほとんどが出来上がってしまっていたので、その箇所について追加修正をおこなう。

文字行列式の展開の関数 expand の結果として、多重のカッコが生成し、その処理のためにカッコはずしの関数 depar, deparn, … などを必要とし、かつ次数Ｎによってカッコの多重度が異なるので処理を区別せざるを得ないなど煩わしかった。

改良した関数 exp では再帰呼び出しの部分でカッコをほどいた形で行うようにした。そのため擬似的掛け算の関数 mult を作成した。

つまり、

’A’ mult ’+EI-FH’ => ’+AEI-AFH’

のように、文字列の掛け算でカッコをはずした形を返すようにした。例えば

exp Z

+AFKP-AFLO-AGJP+AGLN+AHJO-AHKN-BEKP+BELO+BGIP-BGLM-BHIO+BHKM+CEJP-CELN-CFIP+CFLM+CHIN-CHJM-DEJO+DEKN+DFIO-DFKM-DGIN+DGJM

これにも正規表現、ボックス化、L:0 などＪのプログラム手法を用いた。この改良により、以前の関数 dpar, deparn, …, exdet は不要になり、処理のスピードアップもはかられた。

以下関数 exp, mult など改良部分を示す。

```
NB. expansion revised version ============================================
exp =: 3 : 0
M =. y.
N =. {. $ M
if. N = 2
  do.
    ’+’, (((<0,0){M),(<1,1){M),’-’,(((<0,1){M),(<1,0){M)
    return.
  else.
    j =. 0
    P =. ’’
    while. j < N
      do.
        if. 2|j
          do.
            P =. P, plus_min ((<0, j){M) mult  (exp (0,j) cofact M)
          else.
            P =. P, ((<0, j){M) mult (exp (0,j) cofact M)
```

```
            end.
        j =. j + 1
        end.
    P
end.
)

cofact =: 3 : 0         NB. cofactor minor eg. (2,2) cofact Z
:
(<<<0{x.){(<<<1{x.){"1 y.
)

mpat =: '[¥+-][[:alpha:]]+'
mult =: 3 : 0
:
y =. (<'') -. ~({."2 mpat rxmatches y.) rxcut y.
; x. multin L:0 y
)

multin =: 3 : 0
:
({. y.), x. , (}. y.)
)

plus_min =: 3 : 0       NB. switch sign '+' <-> '-'
pm0 =. '_' ((('+'&=)#(i.@#))y.) } y.
pm1 =. '+' ((('-'&=)#(i.@#))pm0) } pm0
pm2 =. '-' ((('_'&=)#(i.@#))pm1) } pm1
)
NB. Revised version end ============================================
```

付録　Jプログラム・リスト

```
NB. Formula Processing for Eigen Matrix -- form_eigen2.ijs
NB.   revised from form_eigen1.ijs  2007/5/22
NB. programmed by T. Nishikawa 2007/4/30, 2007/5/15
NB. using Formula Proccesing Program formulax.ijs by T.N.
wr=. 1!:2&2

NB. Symbolic Manipulation in Determinant Expansion ==================
A_Z =: (65+i.26){a.
a_z =: (97+i.26){a.
Alphabet =: A_Z, a_z

X =: 2 2$Alphabet
Y =: 3 3$Alphabet
Z =: 4 4$Alphabet
W =: 5 5$Alphabet
V =: 6 6$Alphabet

require 'regex'
NB. Here, exp, mult, ...
NB. Insert Revised Functions, exp, mult , ... from the previous page

XX =: exp X
YY =: exp Y
ZZ =: exp Z
WW =: exp W
VV =: exp V



NB. Calculation in Numerical Input Data ================================
NB. Several Examples

NB. Maple V Data
MDA3 =: 3 3$'(1-x)';'(2)';'(2)';'(0)';'(2-x)';'(1)';'(-1)';'(2)';'(2-x)'
```

```
NB. make determinant from table form data ============================
makedet =: 3 : 0
y =. ". y.
('m';'n') =. $y
DE =: ''
i =. 0
while. i < m do.
  j =. 0
  while. j < n do.
    v =. ": (<i, j){y
    v =. ('_';'-') rxrplc v
    if. i = j
      do.   DE =. DE, < '(', v, '-x',')'
      else. DE =. DE, < '(', v, ')'
    end.
    j =. j + 1
  end.
  i =. i + 1
end.
(m, n)$DE
)

NB. Nakano's Problem Data
NDA2 =: 2 2$'(3-x)';'(2)';'(4)';'(1-x)'

NDA3 =: 3 3$'(1-x)';'(2)';'(2)';'(2)';'(0-x)';'(0)';'(2)';'(1)';'(-1-x)'

nda4 =: ] ;._2 (0 : 0)
  2  1  3  4
  1 _3  1  5
  3  1  6 _2
  4  5 _2 _1
)
NDA4 =: makedet nda4
```

```
nda5 =: ] ;._2 (0 : 0)
  2  0  0  1  1
  0  2  0  0  0
  3  2  0  0  0
  0 _1  0  3  0
 _1  1  1  0  0
)
NDA5 =: makedet nda5

nda6 =: ] ;._2 (0 : 0)
  0  1  0  0  0  0
 _2  0  1  1  0  3
  1  1  0  0  0 _1
  1  0  0  0  0 _1
 _1  0  4  2 _1  1
  0  1  0  0  0  0
)
NDA6 =: makedet nda6

NB. Assignment of Input Data ========================================
require 'user¥formulax.ijs'
NB. XX assign NDA2
NB. YY assign NDA3
assign =: 3 : 0
:
x =. x.
i =. 0
y =. , y.
NB. wr y
while. i < #y
  do.
     x =. ((i{Alphabet);(>i{y)) rxrplc x
     i =. i + 1
end.
NB. wr 'initial_data = ', x
x1 =. numbox x
```

```
x2 =: numcalc L:0 x1
x3 =: ; x2
NB. wr 'det(x) = ', x3
x3
)

NB. multiply numbers, then extract coefficient first
NB. eg. -(1-x)(-2)(3-x)(4) => +8(1-x)(3-x)
NB. processed negative eg. (-3) revised 2007/5/11
NB. zero coefficient cut 2007/5/12
pnum =: '¥([¥+-]*[[:digit:]]+¥.*[[:digit:]]*¥)'
numcalc =: 3 : 0
y =. y.
if. (({.y) = '+') +. (({.y) = '-')
  do.
    sign =. {.y
    y =. }. y
  else. sign =. ''
end.
y =. (<'') -.~ ({."2  pnum rxmatches y) rxcut y
N =. #y
i =. 0
jn =. 0
Y =. ''
while. i < N
  do.
    t =. >i{y
    if. 0 = num =. {. {."2  pnum rxmatch (>i{y)
       do.    Y =. Y, (i+65){a.
              jn =. jn + 1
       else. Y =. Y, (i+97){a.
    end.
    i =. i + 1
end.
yy =. ; (/: Y){y
if. jn=0 do. (sign, t) return. end.
```

```
yyy =. (<'') -.~ ({.~2  pnum rxmatches yy) rxcut yy
numb =. jn {. yyy
form =. jn }. yyy
num =. ; numb
val =. ~. }: }. ('¥)¥(';'*') rxrplc num
if. val=0 do. res =. '' return. end.
res =. sign, (~:val), ;form
res =. ('-_';'+') rxrplc res
res =. ('¥+_';'-') rxrplc res
)

NB. boxcut by at ')' in either pattern ')+(' or ')-('
numbox =: 3 : 0
y =. <;._1 '!', '¥)[¥+-]¥(' ({. , '!'&, @}.) rxapply  y.
)

NB. Characterstic Polynomial ================================================
NB. using formulax.js
charpoly =: 3 : 0
select. {. $ y.
  case. 2 do. chp =. XX assign y.
  case. 3 do. chp =. YY assign y.
  case. 4 do. chp =. ZZ assign y.
  case. 5 do. chp =. WW assign y.
  case. 6 do. chp =. VV assign y.
end.
R =. run chp   NB. execute run of formulax
wr 'Characterstic Polynomial = ', R
RR                          NB. RR is global return of formulax
)

NB. Eigen Value Calculation =================================================
eigen =: 3 : 0
chapol =. charpoly y.
y =. excoef L:0 sign_cut chapol
eig =. p. ; ~. L:0 y
```

```
'Eigen Values = ', ″: ; 1{eig
)


NB. sign_cut '8+9x-2x^2+x^3-5x^5'
sign_cut =: 3 : 0
<;._1 '!', ('[¥+-]') ('!'&(,])) rxapply y.
)

NB. extract coefficients
excoef =: 3 : 0
('x' i.~ y.) {. y.
)
```

```
NB. Old Verision as form_eigen1.js ==============================
NB. Expansion of Determinant represented in Alphabet Characters =====
expand =: 3 : 0        NB. formerly defined as expand0
M =. y.
N =. {. $ M
if. N = 2
  do.
    (((<0,0){M),(<1,1){M),'-',(((<0,1){M),(<1,0){M)
    return.
  else.
    j =. 0
    P =. ''
    while. j < N
      do.
        if. 2|j
          do.
            P =. P,'-', ((<0, j){M), '(',(expand (0,j) cofact M), ')'
          else.
            P =. P,'+', ((<0, j){M), '(',(expand (0,j) cofact M), ')'
          end.
        j =. j + 1
      end.
    P
end.
)

cofact =: 3 : 0        NB. cofactor minor eg. (2,2) cofact Z
:
(<<<0{x.){(<<<1{x.){"1 y.
)

NB. deparenthesize ======================================================
va =: {.~ i.&'-'
vb =: }.@(}. ~ i.&'-')
depar =: 3 : 0
a =. (y. i. '(') {. y.
```

```
b =. }: (>: y. i. '(') }. y.
if. 1 = #a
  do. (a),(va b),'-',(a),(vb b)
  elseif. '-' = ({.a)
    do. (a),(va b),'+',(}.a),(vb b)
    elseif. 1 do. (a),(va b),'-',(}.a),(vb b)
end.
)


plus_min =: 3 : 0
pm0 =. '_' (('+'&=)#(i.@#))y.) } y.
pm1 =. '+' (('-'&=)#(i.@#))pm0) } pm0
pm2 =. '-' (('_'&=)#(i.@#))pm1) } pm1
)

deparn =: 3 : 0        NB. called from 4x4 det
d0 =. (y. i. '(') {. y.
d1 =. }: }. (y. i. '(') }. y.
d2 =. ((}. d1) i. '+') <. ((}. d1) i. '-')
d3 =. (((#d1)%(>:d2)),(>:d2))$d1
if. '+' = ({. d0)
   do.
      d4 =. ({."1 d3),. }."1 d0,"1 (}."1 d3)
   else.
      d4 =. (plus_min {."1 d3),. }."1 d0,"1 (}."1 d3)
end.
, d4
)

deparn2 =: 3 : 0       NB. called from 5x5 det
de0 =. (y. i. '(') {. y.
de1 =. }: }. (y. i. '(') }. y.
de2 =. }: (<;._1 ')', de1),L:0 ')'
de3 =. deparn L:0 de2
deparn de0, '(', (; de3), ')'
```

```
)

deparn3 =: 3 : 0       NB. called from 6x6 det
dep0 =. y.
dep1 =. 3{. ; dep0
dep2 =. }: 3}. ; dep0
dep3 =. deparn2 L:0 }: <;._1 '!', ('¥)¥)') ((,])&'!') rxapply dep2
dep4 =. ((,])&')')L:0 dep1, L:0 dep3
dep5 =. ; deparn L:0 dep4
)

NB. expand and deparenthesize determinant of 2,3,4,5 order
NB. can be adapted for 6 order 2007/5/14
pat =: '[¥+-][[:alpha:]]¥([[:alpha:]][[:alpha:]]-[[:alpha:]][[:alpha:]]¥)'
expdet =: 3 : 0
select. {. $ y.
  case. 2 do.   NB. eg. expdet X
    expand y.
  case. 3 do.   NB. eg. expdet Y
    y30 =. expand y.
    y31 =. (}: <;._1 ')', y30) ,L:0 ')'
    y32 =. ; depar L:0 y31
  case. 4 do.   NB. eg. expdet Z
    y40 =. expand y.
    y41 =. pat depar rxapply y40
    y42 =. (<;._1 ')', y41),L:0 ')'
    y43 =. ; deparn L:0 y42
  case. 5 do.   NB. eg. expdet W
    y50 =. expand y.
    y51 =. pat depar rxapply y50
    y52 =. ('¥)¥)';'!') rxrplc y51
    y53 =. }: (<;._1 '!', y52),L:0 '))'
    y54 =. ; deparn2 L:0 y53
  case. 6 do.   NB. eg. expdet V
    y60 =. expand y.
    y61 =. pat depar rxapply y60
```

```
    y62 =. ('¥)¥)¥)';'!') rxrplc y61
    y63 =. }: (<;._1 '!', y62),L:0 ')))'
    y64 =. ; deparn3 L:0 y63
end.
)
```