

# 行列式の計算法異考

## 固有値問題の脇き道

中野嘉弘 (84才・札幌市)

山下紀幸 (82才・横浜市)

FAX 専 011-588-3354

TEL/ FAX 045-851-3721

yoshihiro@river.ocn.ne.jp

行列式の求値で、J言語の 基本的関数 `det =: -/. *` を用いずに、やる方法を色々考えた。

### 0. は し が き

今まで見逃されて来た固有値問題の直接法 (文献 1 a ~ 1 f)、即ち、我らの Naigen法、さらに、最古参の Frame 法、あるいは、同じ内容かもしれないが、仏露の大家 Leverrier-Faddeev 法と格闘している内に、気が付いた記事を述べる。予定して居た固有ベクトル計算法等の追加記事は次回以降に廻す。

行列式の計算では、J言語は断然、有利である。それは、J言語の基本的関数の `det =: -/. *` のおかげである。同じような基本関数を提供するのは、J言語の他には、SHARP APL 位かも知れぬ (文献 2)。小生愛用の英国製の Dyadic APL では残念ながら、この流儀は出来なかった。かの有名な APL2 では、確認テストの暇がなかった。数年前に折角、インストールして置いた教育版を動かさないので、専門家からの情報を期待したい。

最近、我が JAPLA 会友・横浜市の山下紀幸氏より、FAX 情報が入電した (文献 3)。その内容は「APL にはサラスの展開による 3 次までの行列式計算法はあるが、J言語の基本関数のように、何次にでも使える J言語の関数 `det` の流儀は無いので、APL 言語で作って見たとの、自身の古い報告を再発見したと云うものだ。行列の次数を遞減する事を反復して、低次の行列式の求値に転渡する事を考えたのだ。一昔前のものだが、今回、J言語に直して提供したい。原理はガウスの消去法と同じ。」とあった。これが、きっかけで、今回の報告とはなった。

一般的に、行列式の計算法には、

1) Recursive 反復法 と 2) Elimination 消去法 とが可能である。

この事は、「文献 2」の Joseph De Kerf も述べて居る。

しかし、山下 FAX の話題により、消去法を先ず、取り上げる。

### 1. 消 去 法

1) 山下流 APL 関数 `DET N` (N 次用) より

$\nabla R \leftarrow DET N A; K; N$

[1]  $N \leftarrow 1 \uparrow \rho A$

[2] LP:  $A1 \leftarrow A[1;1]$

[3]  $A[1; ] \leftarrow A[1; ] \div A[1; 1]$

```

[4] K ← 2
[5] LP1: A[K; ] ← A[K; ] - A[K; 1] × A[1; ]
[6] → (N ≥ K ← K + 1) / LP1
[7] A ← A[1 ↓ ι N; 1 ↓ ι N]
[8] A[1; ] ← A[1; ] × A1
[9] → (2 ≤ N ← 1 ↑ ρ A) / LP
[10] □ ← A ∇

```

テスト行列

```

ya2 ← 2 2 ρ 6 2 2 3
yb3 ← 3 3 ρ 3 1 4 1 5 9 2 6 5
yc5 ← 5 5 ρ (ι 5), ((ι 5)*2), ((ι 5)*3), ((ι 5)*4), ((ι 5)*5)

```

APL 言語での計算結果 :

```

DETN ya2 → 14
DETN yb3 → -90
DETN yc5 → 34560

```

1' ) J 言語プログラム (J601版 中野) jdet0 の結果例 :

```

      jdet0 ya2   jdet0 yb3   jdet0 yc5
6 2      3 1 4     1 2 3 4 5
2 3      1 5 9     1 4 9 16 25
0 sec    2 6 5     1 8 27 64 125
14       0 sec    1 16 81 256 625
          _90      1 32 243 1024 3125
                          0 sec
                          34560

```

```

      jdet0 =: 3 : 0
wr A =. y
      t =: 6! : 1''
      k =. 0
      sig =. 1
label_lp.
      n =. # A
      a0 =. (<0 0) { A
      A =. A E2 0, (% a0)
      k =. 1
label_nxt.
      ak =. (<k 0) { A
      A =. A E3 k, 0, (-ak)
      k =. k + 1
if. k < n do. goto_nxt. end.
      A =. 0 omitrc A
      A =. A E2 (0, a0)
if. 2 <: (#A) do. goto_lp. end.
      wr (':(6!:1'')- t), ' sec '
      A * sig
)

```

E2, E3 等は、配列要素の取り出し (from) と 修正 (amend) の関数で、一昔前に、我が JAPLA の西川会長によって、与えられた (元来は J - Dictionary の例)。

Script 類は、末尾に掲示した。

上記のテスト行列 ya2, yb3, yc5 では、APL からの直訳プログラム jdet0 で OK であったが、データ行列の成分、A[1;1] がゼロになるような、不都合な場合には、結果の行列式値はゼロとなり、使えない。どうも、実行中に DOMAIN エラーを発生するらしい。J 言語の版によっては、エラー表示法が多少、異なるらしいが？ 実は、既に APL 言語の段階でも、はっきり DOMAIN エラーなのである。

例えば エル5 l5=: latin 5 (関数 latin は、山下による。末尾に記載。)

```
jdet0 l5      jdet l5      jdet l6      jdet l9
1 2 3 4 5      62.93 sec    98.872 sec    155.493 sec
2 3 4 5 1      1875          _27216        2.15234e8
3 4 5 1 2
4 5 1 2 3      det l5      det l6      det l9
5 1 2 3 4      1875          _27216        2.15234e8
0 sec
0
```

(データ行列が エル2 l2 から l4 までは、jdet0 関数でも正しく計算するが、latin 5 以上では不可。改良関数 jdet では latin 13 等などでも、OK！)

対策としては、A[1;1] 成分がノンゼロになる如く、データ行列の行を交換するなどの操作を加える必要がある。山下の APL プロ自身は、その対策を欠いている。中野の改良プログラムを jdet とし、J 言語の基本的関数 det の結果と比較した。上首尾の結果は上記。しかし、演算時間が若干長くなったのが気になる。山下マシンより、300倍も高速なマシンによるものであるから、なおさらである。改良プログラム jdet の内容は、いささか長くなるので、本稿の末尾に纏めて置いた。

本稿を清書中の11月20日に、異論的な山下FAX (0が断然多いデータ行列では、行の交換は無理ではないか?) が到来した。(文献6)

山下データ (tdg8、8次、各行共、ノン 0 が 2 ~ 3ケのみ) にギョットだったが、実は、行交換の必要が全く無い簡単な問題に過ぎなかった。pivot が 0 にならない型に過ぎぬ。改良以前のプログラムでOK！(文献7)

```
jdet0 Tdg8
2 _1 0 0 0 0 0 0
_1 2 _1 0 0 0 0 0
0 _1 2 _1 0 0 0 0
0 0 _1 2 _1 0 0 0
0 0 0 _1 2 _1 0 0
0 0 0 0 _1 2 _1 0
0 0 0 0 0 _1 2 _1
0 0 0 0 0 0 _1 2
0 sec
9
```

## 2. 正しく動く APL プログラム

前節1. の例では、元来の APL プログラム自身が不十分であった。当然、APLプログラム自身を改良すべきであるが、実は既に、(文献2)の著者 De Kerf 自身が与えている。中野が Dyag APL でチェックした範囲では、前節の如き難点は発生せず、全て、正しい計算をしてくれた。即ち、前節の改良版に対応する如く、始めから作られているものと評価出来た。

冒頭の「はしがき」で紹介した如く、その2例のプログラム、即ち「recursive 反復法」と「elimination 消去法」の2法が呈示されている。再録して置こう。

● recursive

```
[0] D ← DET1 M;N;I; □ IO
[1] → ((2 ≠ ρ ρ M) (1 ≠ =/ρ M) (1 ↑ ,M)= 1 ↑ ,M)/0
[2] → (2 > N)/0, D ← (+/1 ↑ ,M)+0=N ← 1 ↑ ↓ ρ M
[3] D ← N ρ I ← □ IO ← 1
[4] LAB: D[I] ← DET1 M [1 ↓ ι N; (I ≠ ι N) / ι N]
[5] → (N ≥ I ← I+1)/LAB
[6] D ← -/M[1;]xD
```

◎ elimination

```
[0] D DET2 M;N;I; □ IO
[1] ((2 ≠ ρ ρ M) (1 ≠ =/ρ M) (1 ,M)= 1 ,M)/0
[2] (1=1 ρ M)/0, D +/1 ,M
[3] (0=1 ρ M)/0, D □ IO 1
[4] LAB: M[I,1;] M[1,I (I/M) /I/M] ι /I/M;]
[5] M[;J,1] M[;1,J (M[1;]) ι /M[1;]]
[6] D DxM[1;1]x ~1*+/1 I,J
[7] M(1 1 ↓ M)-((1↓M[1;] ,x1 ↓ M[1;]) M[1;1]
[8] (1 ≤ 1 ↑ ρ M) /LAB
```

しかし、このAPLプログラムをJ言語に、翻訳する事は、それほど、簡単とは思えないので、この後は、別途の思考をする。また、APL Fontの関係で、APLプログラムの組版はあきらめ、別途、本稿末尾に、スキャナー版を掲載する。

### 3. 古屋 の プログラム

すでに、固有値問題の報告の前報(その6)、Frame法で登場した古屋 茂東大教授の著書「行列と行列式」(文献5)に、行列式値を求める計算法が、巻末の付録になっている。それを、利用しない手は無い。早速、利用させて頂く。特に、p.141には、データ・マトリックスの2つの行の間で、ある選ばれた2行2列の要素間の2次の行列式値(クロス積の差)を、逐次に求める手法が力説されていた。J言語の関数 `det = ./.*` の素過程に相当する感じである。古屋先生は、図示はしているが、命名していないので、仮に、`fry` 或いは、クロス・タブの意味で関数 `crt` と名付ける。或いは、消去法の解説の節にあるので、関数 `elim` と名付けよう。

```
fry =: crt =: elim =: 3 : 0
  n1 =. <: n =. # A =. y
  i =. 0
  A0 =. 0 { A
  a0 =. { . A0
  B =. 0
  i =. 1
```

```

while. i < n do.
  Ai =. i { A
  ai =. { Ai
  bi =. (a0* }. Ai) - (ai* }. A0)
  B =. B, bi
  i =. i + 1
end.
B =. (n1, n1) $ }. B
)

```

その動作の例は、 原行列 A

|    |    |    |    |    |
|----|----|----|----|----|
| 2  | 3  | 1  | _3 |    |
| _1 | 2  | 2  | 4  |    |
| 4  | 1  | _3 | 5  |    |
| 5  | _4 | _4 | 1  | から |

```

.....
B =. elim A 7 5 5
              _10 _10 22
              _23 _13 17

```

```

.....
C =. elim B  _20 204
              24 234

```

最後の行列式値は ..... 9676 を得る。

(実際の det A は さらに 9676 / ((2^2)\*7) としての 342 で得られる。)

この古屋流の行列式算法プログラムは、簡単な場合（ピヴォットにゼロが皆無）では

```

fryn0 =. 3 : 0
wr A =. y
t =. 6! : 1''
ny =. # A
i =. 0
a0 =. (<0 0) { A
cf =. a0 ^ nya =. (ny - 2)
B =. elim A
i =. 1
b0 =. (<0 0) { B
cf =. cf * b0 ^ nya =. nya - i
i =. i + 1
while. i < ny do.
C =. elim B
c0 =. (<0 0) { C
if. (ny - i) > 2 do. cf =. cf * c0 end.
B =. C
i =. i + 1
end.
wr (":(6!:1'') - t), ' sec'
D =. B % cf
)

```

これが、ピヴォットにゼロがあって、行の交換の操作を必要とする時には、その為の関数 `zrchgf` を含む改良関数 `fryn` を用いる。それらのプログラムは、長くなる

ので、末尾に纏めて置いた。

計算例： (処理例の最初の出力は、与データ行列)  
fryn0 fad5

```
2 1 0 0 0
1 2 1 0 0
0 1 2 1 0
0 0 1 2 1
0 0 0 1 2
final cf = 576
6 (結果行列式値)
```

比較： det fad5  
6

#### 4. J言語の det 関数 と 消去の elim 関数の類似

先の3. 節で登場した関数 crt (別名 elim) 自身の動作を調べると面白い結果が得られた。J言語の基本的関数 det =: -/. \* と、そっくりなのである。データ行列に エル l2 =. latin 2 等を用い、

```
det l2    det l3          det l4
_3        _18            160

crt l2    crt l3    crt ^:(2) l3    crt ^:(3) l4
_3        _1 _5    _18            _160 (絶対値は等しい)
                _5 _7
```

更に、途中経過をも示せば

```
-/. * l2    crt \ l2          -. * l3    crt \ l3
1 (途中)    0                6 (途中)    0 0    (途中)
_3 (最終)   _3 (最終)       5          0 0
                3
                4          0 0
                _18 (最終)
                                _1 0
                                _1 _5
                                _5 _7
                                _18 (最終)
```

これらの知見から、J言語の det 関数 は「消去法」であろうと推測される。

#### 5. 行列式求値 1 行 プログラム

行列式の求値で最短プログラムは、J言語の det =: -/. \* がトップであろうが、これ以外で、最短のものを探そう。

本稿で、今までグジャグジャ考えたが、実は、我らの答は、始めから、決まっていた。

我らの既報、「J言語と固有値問題」シリーズの特に、(その5)、(その6)で登場した Fadeev法 や Frame法 を中野流に纏めた方法 Feigen関数 のゴールは、与行列の特性多項式の最後(0次)の項、即ち、行列式値であった事を思い出して欲しい。即ち、J言語の det 関数を用いずとも、行列式の計算は可能なのであった。その関数の例を与える。

```

Feigdet =: 3 : 0
t=:6!:1"
In =. unitm n =. # A =. y
i =. 1
pi =. - (trace Bi =. A)
while. i < n do.
i =. i + 1
pi =. -i % ~ (trace Bi =. (A indot (Bi + pi*In) ))
end.
wr (":(6!:1") - t), ' sec'
((1)^n) * pi
)

```

これは、本質的に、次の一行プログラムと同値である。

```
Pk =: 'pk=. (k=>:k) trace0 Bi =. (A indot (Bi+ pk*In))'
```

先ず、pk=k=0 [Bi=. In =.unitm n =. # A=. (データ行列、例えば latin 6 などの準備はして置く。その後、 ". Pk [Enter キー] を反復する。データ行列の回数繰り返せば、結果は 0 になる。その一つ前の出力が、求める行列式値(少なくとも絶対値)である。符号は、(1)^n (次数)で決まる。[Enter キー]の反復を避けたければ、次の一行プログラムを用いれば良い。

```
Pkdet =: 3 : ' while. k <:y do. (" . Pk ) end. )'
```

例： データ6次 A=.エル l6=. latin 6  
結果 Pkdet 5 から \_27216 ( Pkdet 6 からは 0 となる。)  
比較 det l6 から \_27216

## 6. む す び

偉大なる J言語の det 関数を、敢えて用いずに、行列式値を求める算段をした。とどのつまりは、Frame法 や Fadeev法 の「提灯」を持つ事に帰着したが、それが、本来の目的であったのだ。廻り道の楽しみであった。

## 文 献

- 1 - a) 中野嘉弘「J言語と高等数学 (固有値問題直接法)」  
JAPLA報告 2007.4.2 pp.9
- b) 中野嘉弘「J言語と固有値問題(その2)直接法の発展」  
JAPLA報告 2007.5.26 pp.13
- c) 中野嘉弘「J言語と固有値問題(その3)VECTOR誌投稿原稿」  
JAPLA報告 2007.6.23 pp.3

- d) 中野嘉弘「J言語と固有値問題 (その4) チュートリアル等 LAPACKまで」  
JAPLA報告 2007.6.23 pp.9
  - e) 中野嘉弘「J言語と固有値問題 (その5) ファデーエヴァ法へ」  
JAPLA報告 2007.9.22 pp.8
  - f) 中野嘉弘・山下紀幸「J言語と固有値問題 (その6) frame法を主に」  
JAPLA報告 2007.10.27 pp.8
- 2) Joseph De Kerf: "APL Defined Functions for the Calculation of Determinants" VECTOR 1994, Vol.10 No.3, pp.21-22
- 3) 山下 FAX (2007.11.14 13:30): 「JREP08A9 行列式の値 H10.3.14 山下紀幸」
- 4) 山下 FAX (2007.6.3 11:00): データ行列作成関数 latin と rot
- 5) 古屋 茂「行列と行列式」培風館、1957 初版、1959 増補版、1998 増補第65刷
- 6) 山下 FAX (2007.11.20 am. 9:15): データ行列が、やたらに多くの 0 を含む場合、pivot の交換は難儀だから、別法を探りたい。
- 7) 中野 FAX (2007.11.7): 中野流「一手詰め法 (1行プログラム)」の成果を見よ! 順行型 (Feigen流) でも逆行型 (Ifeigen流) でも可能。

【 参考 Script 類 】

```

E1r =: <@|C.| 行全体の交換
E1c =: <@|C."1| 列全体の交換

E2 =: f`g`|} 但し f=: {:@|*{:@|}{|} と g=: {:@|}

E3 =: F`g`|} 但し F=: [:+/(1:,{:@|)*{:@|}{|}

latin =: 3:0 但し rot =: 3:0
rot ^:(i.y) a =. >: i.y (}.y), {y
)

jdet =: 3:0
wr A =.
t=: 6!:1''
k=. 0
sig=. 1
label_lp.
n =. # A
a0 =. (<0 0) { A

if. a0 = 0 do. j =. 0
goto_chg. end.
label_chg0. A =. A E2 0, (% a0)
goto_k1.

label_chg. j =. j + 1
a0 =. (<j, 0) { A
if. j > n do. goto_ed. end.
if. a0 = 0 do. goto_chg. end.
A =. A E1r 0, j
sig =. sig * _1
goto_chg0.

```

```

label_k1.
    k=. 1
label_nxt.
    ak =. (<k, 0) { A
    A =. A E3 k, 0, (-ak)
    k =. k + 1
    if. k < n do. goto_nxt. end.
    A =. 0 omitrc A
    A =. A E2 (0, a0)
    if. 2 <: (#A) do. goto_lp. end.
    wr (":(6!:1")- t), 'sec '
    A * sig
)

```

```

Ifeigen =: 3 : 0      Feigen =: 3 : 0
    t=.6!:1"         t=.6!:1"
In=. unitm n =. # A =. y    In=. unitm n =. # A =. y
Ai=. %. A
Bd=. det y           Bi =. A
sig=.(1)^n          pi =. trace Bi
nans=. p =. sig* Bd    nans=. 1, (-pi)
    i=. 1           i =. 0
Bf=. Bd*In
    while. i < n-1 do.      while. i < n do.
                                i =. i + 1
    isig=.i* sig =.(1)^i
    Bfi =. (Bf indot Ai)    Bi =. (A indot ( Bi - pi * In))
    p=. (trace Bfi) % i    pi =. (trace Bi) % i
    nans =. nans, p        nans =. nans, -pi
    Bf =. Bfi + (p*_1) * In
    i=.i+1
end.                    end.
wr (":(6!:1")-t), 'sec'    wr (":(6!:1")-t), 'sec'
nans=.nans,((trace y)*_1), 1    nans
)

```

諸関数 unitm : to make unit matrix  
 det : to make determinant  
 indot : inner product between matrices  
 trace : to get diagonal sum