Ｊと OpenGL 思考からの線形代数の理解
Gram-Schmidt の QR 過程の図表示と固有値の計算

西川　利男

１．線形代数－Ｊと OpenGL から見てみる

複数の値の集合をまとめて計算したいという要望は自然科学、社会科学を問わずいろいろな分野で必要である。これをコンピュータ・プログラミングとして行うのが、APL やＪであることはいうまでもない。一方、数学として行うのが線形代数であると筆者は思っている。

さて、線形代数で複数の値の集合はベクトルと呼ばれる。そしてこれらに対して演算操作をするのが行列である。Ｊのことばで言えば、ベクトルは名詞であり、行列演算は動詞である。ただし、この動詞は名詞として左引数をとり、これが行列である。

```
( マトリックス ) matp ( ベクトル ) = ( ベクトル )
  名詞        動詞
        動詞        名詞      名詞
```

ここで動詞 *matp* は +/ . * のように定義されている。

線形代数では数値のベクトルと行列に対して、*matp* のような演算規則が定められている。なお、個人的には行列なる語よりマトリックスのほうが良いと思う。このような考え方については以前報告したことがある[1]。今回の報告はこれに続くものである。

ところで、数値の並びだけではなかなかイメージがわきにくいが、視覚で表せばずっと容易になる。これが線形代数の幾何モデルである。

要素２つのベクトルは平面上で、要素３つのベクトルは立体空間内でのそれぞれ矢印のベクトルで表される。これは点の位置を示すものと見たときは位置ベクトルと呼ばれる。行列はここではベクトルを拡大、縮小、回転などして新しい別のベクトルを生成する変換操作に対応する。

ここで OpenGL の考え方からもひとこと述べる。OpenGL では頂点の座標値をもとにモデルを作るがこれが則、ディスプレイ画面に表示されるのではない。レンダリングという過程によって現実のディスプレイ画面の図になるのである。

線形代数におけるベクトルはこれをモデルと考え、行列操作なるレンダリングによってはじめて図として見ることになるとも考えられる。

２．グラム・シュミットの QR 過程―その考え方と計算アルゴリズム

このテーマについては、先月の中野氏および志村氏による報告と解説がある[2],[3]。

---

[1] 西川利男「Ｊグライックスによる固有値・固有ベクトルの体験ツール」
JAPLA 研究会資料 2007/6/23

[2] 中野嘉弘「速報：階級の問題」同報，2010/2/27
[3] 志村正人「Jの簡易反復法による関数のプログラミング」同報，2009/7/30，p.6

先の行列は線形代数では操作を表すＪで言う動詞であったあったが、ここで発想の転換をする。すなわち、今後は単に数値がタテヨコに並んだ集まりとのみ考えることにする。ここでは行列ではなく、配列（アレイ）と呼んだようが良いかもしれない。

つまり、たとえば３行３列の行列ではなく、３列のベクトルが３つあると考える。これは幾何イメージでは、３次元空間に３つのベクトル A1, A2, A3 があり、これらは一般には勝手な方向を向いている。

これから互いに直交するベクトル V1, V2, V3 を求める。このアルゴリズムが一般に Gram-Schmidit 過程と呼ばれるものである。

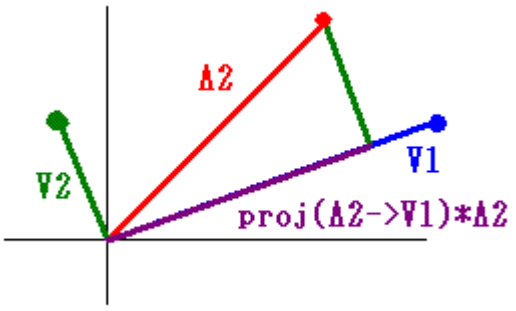
まず、スタートとして、

$V1 = A1$

とする。

つぎにベクトル V1 に対してベクトル A2 の直交する成分のベクトル V2 を求める。これには、ベクトル A2 からベクトル V1 への次の正射影 *proj* を利用する。

$$proj(A2 \rightarrow V1) = \frac{A2 \bullet V1}{V1 \bullet V1}$$



ここで「 • 」はベクトルのドット積（内積、スカラー積）である。そして、次のベクトル演算を行う。

$V2 = A2 - proj(A2 \rightarrow V1) * V1$

この意味を図によって説明する。

ベクトル A2 から A2 の V1 への正射影を引くとはベクトル減算である。つまり、三角形の２辺の合成で作るベクトルの加算に対する減算操作である。その結果出来たベクトルを原点に移動して V2 とし、図のようになる。これにより直交するベクトルができた。

さらにベクトル A3 からの直交ベクトルは、つぎのように２段階で求められる。

$A31 = A3 - proj(A3 \rightarrow V1)$

$V3 = A31 - proj(A3 \rightarrow V2)$

つまり、ベクトル A3 からベクトル V1 への正射影をベクトル減算で求め中間ベクトル A31 を得る。さらにこの A31 からベクトル A3 のベクトル V2 への正射影をベクトル減算して最終ベクトル V3 とするのである。

これが Gram-Schmidit 過程のポイントである。

つぎに、具体的な数値例でやってみよう。

```
   DA
1  0 _1
1  2  1
2  2  3
   'A1 A2 A3' =: |: DA
   A1
1  1  2
   A2
0  2  2
   A3
_1  1  3
```

３つのベクトル A1, A2, A3 は３次元空間ではテンデンバラバラの方向である。
上の Gram-Schmidit 過程に従って、計算する。

```
   V1 =. A1
   V1
1 1 2
   P21 =. A2 proj V1
   V2 =. A2 - P21*V1
   V2
_1 1 _4.44089e_16
   P31 =. A3 proj V1
   P32 =. A3 proj V2
   V3 =. A3 + (- P31*V1) + (- P32*V2)
   V3
_1 _1 1
```

得られた V1, V2, V3 をそれぞれ正規化して Q1, Q2, Q3 とする。

```
   Q1 =. V1 % norm V1
   Q2 =. V2 % norm V2
   Q3 =. V3 % norm V3
   Q =. Q1,.Q2,.Q3
   Q
0.408248    _0.707107 _0.57735
0.408248     0.707107 _0.57735
0.816497 _3.14018e_16  0.57735
```

つぎに、元の行列に戻すものとして R 行列を求める。

```
   R1 =. (norm V1), 0, 0
   R2 =. (P21 * norm V1), (norm V2), 0
   R3 =. (P31 * norm V1), (P32 * norm V2), (norm V3)
   R =. R1,. R2,. R3
   R
2.44949 2.44949 2.44949
      0 1.41421 1.41421
      0       0 1.73205
```

以上の過程が QR 分解と呼ばれる計算アルゴリズムである。
ここで使用した J の関数は以下のとおりである。

```
dotp =: +/ @: *
norm =: %:@([ dotp ])
proj =: 4 : 0
(x. dotp y.) % (y. dotp y.)
)
```

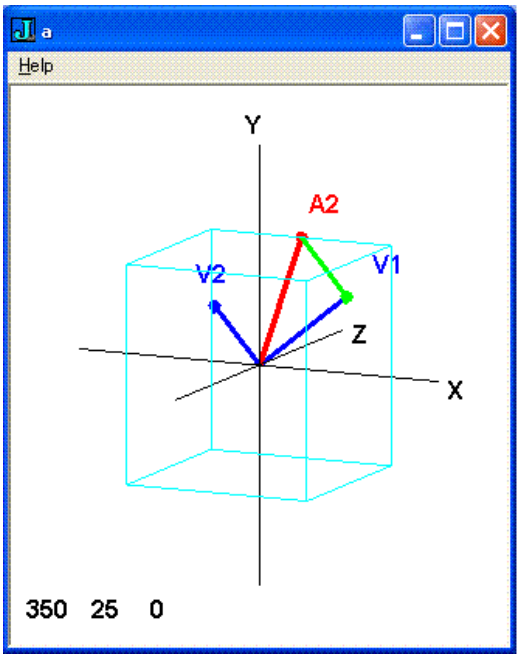
なお、以上の計算は J では、プリミティブ 128!:0 を使えば、一発で求められる。

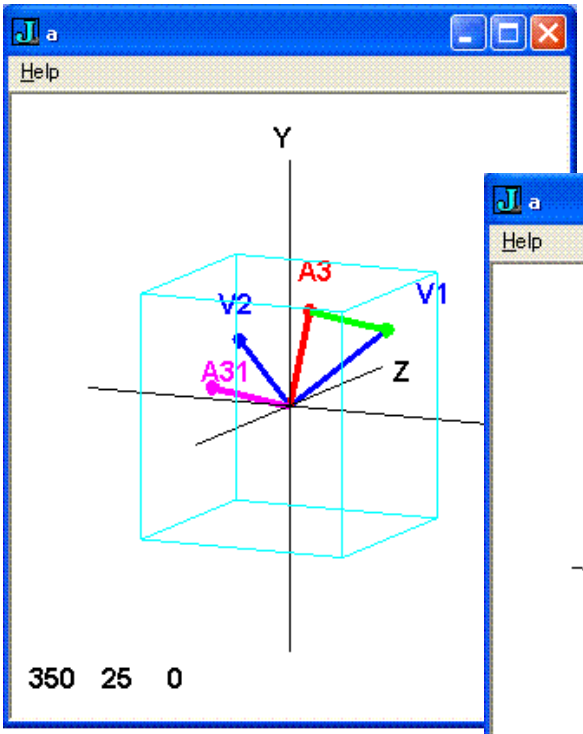
```
   128!:0 DA
+----------------------------+----------------------+
|0.408248    _0.707107 _0.57735|2.44949 2.44949 2.44949|
|0.408248     0.707107 _0.57735|      0 1.41421 1.41421|
|0.816497 _3.14018e_16  0.57735|      0       0 1.73205|
```

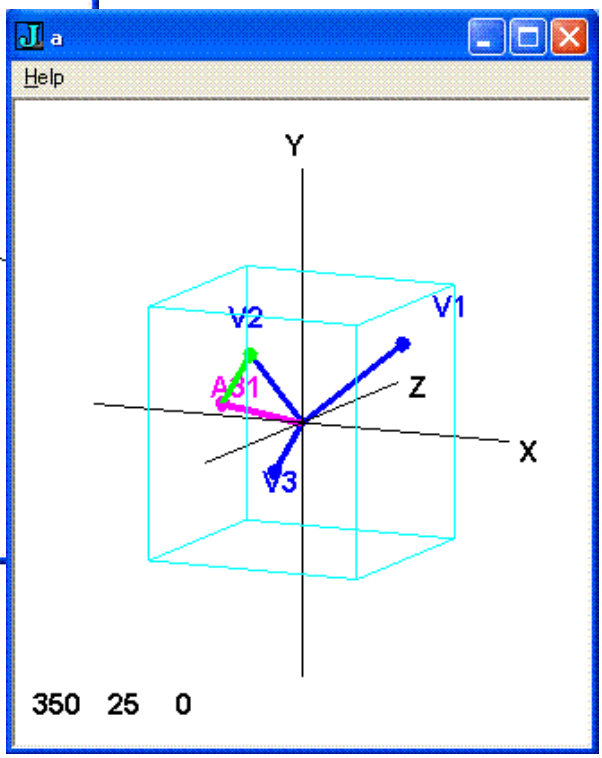
+----------------------------+----------------------+

３．OpenGL によるグラム・シュミット過程のグラフィックス

　上の数値例に対して Gram-Schmidit 過程を OpenGL グラフィックスにより
直交ベクトルの生成するようすを、
ステップを追って図示してみた。
　線形代数において、幾何学イメージが
いかに有用か、お分かりになるだろう。







４．QR法による固有値の計算

Gram-Schmidit過程を繰り返し行って固有値が計算される。（プログラムは後述）

```
   DA
1 0 _1
1 2  1
2 2  3
   10 gramsch DA
===== i:1 ==============
** Q **
  0.408248 _0.707107 _0.577350
  0.408248  0.707107 _0.577350
  0.816497 _0.000000  0.577350
** R **
  2.449490  2.449490  2.449490
  0.000000  1.414214  1.414214
  0.000000  0.000000  1.732051

===== i:2 ==============
** Q **
  0.872872 _0.356348 _0.333333
  0.377964  0.925820  0.000000
  0.308607 _0.125988  0.942809
** R **
  4.582576  0.377964 _0.925820
  0.000000  0.925820  0.377964
  0.000000  0.000000  1.414214

===== i:3 ==============
** Q **
  0.986559 _0.112982 _0.118056
  0.119337  0.991683  0.048196
  0.111629 _0.061637  0.991837
** R **
  3.909695 _1.074029 _2.176768
  0.000000  0.945559  0.542404
  0.000000  0.000000  1.623005
（途中省略）
===== i:10 ==============
** Q **
  0.999994 _0.002160 _0.002627
  0.002165  0.999996  0.001833
  0.002623 _0.001838  0.999995
** R **
  3.031212 _1.414880 _2.869325      （正確な固有値は　3, 1, 2　である。）
  0.000000  0.998489  0.806192
```

```
  0.000000  0.000000  1.982402
```

Jのプログラム・リスト

```
wr =: 1!:2&2
rd =: 1!:1

dotp =: +/ @: *
norm =: %:@([ dotp ])
proj =: 4 : 0
(x. dotp y.) % (y. dotp y.)
)

NB. Sample Matrix Data
DA =: 3 3$1 0 _1 1 2 1 2 2 3

NB. Gram-Sschmidt Process
NB. e.g. gram DA
gram =: 3 : 0
'A1 A2 A3' =: |: y.
V1 =. A1
P21 =. A2 proj V1
V2 =. A2 - P21*V1
P31 =. A3 proj V1
P32 =. A3 proj V2
V3 =. A3 + (- P31*V1) + (- P32*V2)
wr V1,.V2,.V3
Q1 =. V1 % norm V1
Q2 =. V2 % norm V2
Q3 =. V3 % norm V3
Q =. Q1,.Q2,.Q3
wr '*** Q matrix ***'
wr Q
R1 =. (norm V1), 0, 0
R2 =. (P21 * norm V1), (norm V2), 0
R3 =. (P31 * norm V1), (P32 * norm V2), (norm V3)
R =. R1,. R2,. R3
wr '*** R matrix ***'
wr R
Q;R
)
```

```
matp =: +/ . *

NB. Eigen Value Calculation by Gramschmidt QR Iteration
NB. e.g. 10 gramsch DA
gramsch =: 3 : 0
10 gramsch y.
:
A =. y.
i =. 0
while. i<x. do.
  wr '===== i:', (": >: i), ' ==============='
  'Q R' =. gram A
  wr '** Q **'
  wr 10.6 ": Q
  wr '** R **'
  wr 10.6 ": R
  if. 1 = $yn =. rd 1
    do. '** end **'
        return. end.
  A =. R matp Q
  i =. i + 1
end.
)
```