Cholesky 分解ノート

桂田 祐史

2008 年 6 月 9 日, 2013 年 8 月 25 日

書きかけである。

- 修正 Cholesky 分解のコード
- 帯行列の場合のコード
- Sylvester の慣性律のきちんとした説明

などは書いておきたい。それから最初のうちは下三角因子 L を求めるように書いておいたが、実際には上三角因子 U を求めるようにしているプログラムが多いので、いっそのこと、最初から U を求めるような説明にしておくのが良いかもしれない。

1 序

広い意味の Cholesky (コレスキー, ホレスキー) 分解とは、対称行列に特化した LU 分解である。

この文書では行列は実行列であるとするが、複素行列の範囲で考えることも可能である (転置の代りに Hermite 共役、実対称の代りに Hermite とするわけである)。

正則行列の LU 分解は線型計算において重要な基本操作であるが、行列が対称である場合は、通常の半分の計算量で LU 分解することができる。本来の Cholesky 分解は正値対称行列専用であるが、適用範囲の広い修正 Cholesky 分解がある。

与えられた実対称行列 A が

$$A = LDU \quad (L \text{ は単位下三角行列}, D \text{ は対角行列}, U \text{ は単位上三角行列})$$

と LDU 分解できたと仮定しよう。すると、

$$A = A^T = U^T D^T L^T = U^T D L^T$$

となることと、LDU 分解の一意性から、

$$L = U^T, \quad U = L^T$$

となる。ゆえに

$$A = LDL^T.$$

特に D の対角成分がすべて正の場合には、平方根 $D^{1/2}$ が取れて、それを L に組み込むことで ($LD^{1/2}$ を改めて L とおくことで)

$$A = LL^T$$

となる。

より詳しく言うと、行列 A が単に実対称正則の場合には

$$PAP^T = LDL^T \quad (P \text{ は置換行列, } L \text{ は単位下三角行列, } D \text{ は対角行列})$$

と分解でき (伊理 [2], [3] を参照せよ)、A のすべての主座小行列式が 0 でない実対称行列の場合には、置換が省略できて

$$A = LDL^T \quad (L \text{ は単位下三角行列, } D \text{ は対角行列})$$

と分解でき (修正 Cholesky 分解)、A が正値対称の場合には

$$A = LL^T \quad (L \text{ は下三角行列})$$

の形に分解できる (本来の Cholesky 分解)。

Trefethen and Bau III [9] や森・杉原・室田 [7], 森 [5] などを参考にした。

2 Cholesky 分解

2.1 定義

正値対称行列 $A = (a_{ij}) \in M(N; \mathbf{R})$ に対して、下三角行列 $L = (\ell_{ij}) \in M(N; \mathbf{R})$ が存在して

$$A = LL^T$$

が成り立つ。これを A の **Cholesky 分解** (Cholesky factorization) と呼ぶ[1]。

2.2 Cholesky 分解の存在証明 (1)

Cholesky 分解の存在を証明するため、LU 分解について復習しよう。

[1]「分解する」という動詞には “decompose” が使われることが多いが、「分解」という名詞には “factorization” が使われることが多いようだ。

補題 **2.1 (LU 分解の存在)** $N \in \mathbf{N}$ とする。

(1) 任意の $A \in GL(N;\mathbf{R})$ に対して、置換行列 $P \in GL(N;\mathbf{R})$, 下三角行列 $L \in GL(N;\mathbf{R})$, 上三角行列 $U \in GL(N;\mathbf{R})$ が存在して、

(1) $$PA = LU$$

が成り立つ。

(2) すべての主座小行列式[a]が 0 でないような任意の $A \in GL(N;\mathbf{R})$ に対して、下三角行列 $L \in GL(N;\mathbf{R})$, 上三角行列 $U \in GL(N;\mathbf{R})$ が存在して、

(2) $$A = LU$$

が成り立つ。

(3) 上の (1), (2) における LU は

$$LU = L'DU' \quad (L' \text{ は単位下三角行列}, U' \text{ は単位上三角行列}, D \text{ は対角行列})$$

と書ける。この表わし方は一意的である (「LDU 分解の一意性」)。

[a] MATLAB 風の表記で、$\det A(1:r, 1:r)$ を A の r 次主座小行列式という。

正値対称行列 $A = (a_{ij}) \in GL(N;\mathbf{R})$ の主座小行列式はすべて $\neq 0$ であるから、上の命題から LDU 分解できることが分かる。すなわち適当な単位下三角行列 $L = (\ell_{ij}) \in GL(N;\mathbf{R})$, 単位上三角行列 $U = (u_{ij}) \in GL(N;\mathbf{R})$, 対角行列 $D = \mathrm{diag}\,(d_1, d_2, \cdots, d_N) \in GL(N;\mathbf{R})$ が一意的に存在して

$$A = LDU$$

となる。これから

$$LDU = A = A^T = (LDU)^T = U^T D^T L^T = U^T D L^T \quad (U^T \text{ は単位下三角行列}, L^T \text{ は単位上三角行列})$$

となるので、LDU 分解の一意性から

$$U = L^T.$$

ゆえに $A = LDL^T$ である。Sylvester の慣性律から D の対角成分 d_i はすべて正であることが分かる ($L^{-1}A(L^{-1})^T = D$ となることに注意)。それゆえ

$$D^{1/2} = \mathrm{diag}\,\left(\sqrt{d_1}, \sqrt{d_2}, \cdots, \sqrt{d_N}\right)$$

は D の平方根になる (すなわち $(D^{1/2})^2 = D$)。

$$\widehat{L} = LD^{1/2}$$

とおくと、

$$\widehat{L}\widehat{L}^T = (LD^{1/2})(LD^{1/2})^T = LD^{1/2}(D^{1/2})^T L^T = LD^{1/2}D^{1/2}L^T = LDL^T = A.$$

2.3 Cholesky 分解の一意性

$A = LL^T$ を満たす L は一意に定まらない。それは前小節の議論で D の平方根が一意でないことから明らかである。しかし不定性はそれ (符号の問題) だけである。L の対角成分を正であるものに限れば一意になる。

2.4 Cholesky 分解の計算法

2.2 で Cholesky 分解の一つの存在証明を与えた。その手続きによって Cholesky 分解が計算できるのは明らかであるが、ここではもっと直接的に計算手順を導出する。

$$A = LL^T$$

を成分を用いて表わすと、

$$a_{ij} = \sum_{k=1}^{N} \ell_{ik}\ell_{jk} \quad (i, j = 1, 2, \cdots, N) \tag{3}$$

となる。$L = (\ell_{ij})$ は下三角行列であるから、

$$i < j \quad \Longrightarrow \quad \ell_{ij} = 0$$

が成り立つ。ゆえに (3) は

$$a_{ij} = \sum_{k=1}^{\min(i,j)} \ell_{ik}\ell_{jk} \quad (i, j = 1, 2, \cdots, N)$$

と書ける。また A の対称性から $i \geq j$ についてだけ考えれば十分である。つまり (3) は次と同値である。

$$a_{ij} = \sum_{k=1}^{\min(i,j)} \ell_{ik}\ell_{jk} \quad (1 \leq j \leq i \leq N).$$

これを i の小さい順に書き下すと

$$\begin{aligned}
&a_{11} = \ell_{11}^2, \\
&a_{21} = \ell_{21}\ell_{11}, \quad a_{22} = \ell_{21}^2 + \ell_{22}^2, \\
&a_{31} = \ell_{31}\ell_{11}, \quad a_{32} = \ell_{31}\ell_{21} + \ell_{32}\ell_{22}, \quad a_{33} = \ell_{31}^2 + \ell_{32}^2 + \ell_{33}^2, \\
&a_{i1} = \ell_{i1}\ell_{11}, \quad a_{i2} = \ell_{i1}\ell_{21} + \ell_{i2}\ell_{22}, \quad \cdots, \quad a_{ii} = \ell_{i1}^2 + \ell_{i2}^2 + \cdots + \ell_{ii}^2, \\
&\vdots \\
&a_{N1} = \ell_{N1}\ell_{11}, \quad a_{N2} = \ell_{N1}\ell_{21} + \ell_{N2}\ell_{22}, \quad \cdots, \quad a_{NN} = \ell_{N1}^2 + \ell_{N2}^2 + \cdots + \ell_{NN}^2.
\end{aligned}$$

これは次のように順番に解ける。

$$
\begin{aligned}
&\ell_{11} = \pm\sqrt{a_{11}},\\
&\ell_{21} = a_{21}/\ell_{11}, \quad \ell_{22} = \pm\sqrt{a_{22} - \ell_{21}^2},\\
&\ell_{31} = a_{31}/\ell_{11}, \quad \ell_{32} = \left(a_{32} - \ell_{31}\ell_{21}\right)/\ell_{22}, \quad \ell_{33} = \pm\sqrt{a_{33} - \ell_{31}^2 - \ell_{32}^2},\\
&\vdots\\
&\ell_{i1} = a_{i1}/\ell_{11}, \quad \ell_{i2} = \left(a_{i2} - \ell_{i1}\ell_{21}\right)/a_{i2},\\
&\quad \ell_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\ell_{jk}\right)/\ell_{jj} \quad (j = 1, 2, \cdots, i-1),\\
&\quad \ell_{ii} = \pm\sqrt{a_{ii} - \ell_{i1}^2 - \ell_{i2}^2 - \cdots - \ell_{i,i-1}^2},\\
&\vdots\\
&\ell_{N1} = a_{N1}/\ell_{11}, \quad \ell_{N2} = \left(a_{N2} - \ell_{N1}\ell_{21}\right)/a_{N2},\\
&\quad \ell_{Nj} = \left(a_{Nj} - \sum_{k=1}^{j-1} \ell_{Nk}\ell_{jk}\right)/\ell_{jj} \quad (j = 1, 2, \cdots, N-1),\\
&\quad \ell_{NN} = \pm\sqrt{a_{NN} - \ell_{N1}^2 - \ell_{N2}^2 - \cdots - \ell_{N,N-1}^2}\\
&\qquad\qquad \text{(複号任意)}.
\end{aligned}
$$

この計算では $A = (a_{ij})$ の対角線の下側 ($i \geq j$ の部分) のみを使って、$L = (\ell_{ij})$ の対角線の下側 ($i \geq j$ の部分) のみを計算している (上側は 0 だから計算の必要はない)。

Cholesky 分解の疑似コード (ijk 版)

```
for (i = 1; i <= N; i++) {
   /* L[i][j] (i>j) の計算 */
   for (j = 1; j < i; j++) {
     s = a[i][j];
     for (k = 1; k < j; k++)
        s -= L[i][k] * L[j][k];
     L[i][j] = s / L[j][j];
   }
   /* L[i][i] の計算 */
   s = a[i][i];
   for (k = 1; k < i; k++)
     s -= sqr(L[i][k]);
   L[i][i] = sqrt(s);
}
```

このアルゴリズムは $\ell_{11}, \ell_{21}, \ell_{22}, \ell_{31}, \cdots, \ell_{33}, \cdots, \ell_{N1}, \ell_{N2}, \cdots, \ell_{NN}$ の順番 (行ごと) に計算していくが、ループを jik の順に回して、$\ell_{11}, \ell_{21}, \cdots, \ell_{N1}, \ell_{22}, \ell_{23}, \cdots, \ell_{N2}, \cdots, \ell_{N-1,N-1}, \ell_{N,N-1}, \ell_{NN}$ の順番 (列ごと) に計算していくことも出来る (実はその方が普通であるらしい)。

Cholesky 分解の疑似コード (jik 版)

```
  for (j = 1; j <= n; j++) {
    /* L[j][j] の計算 */
    s = a[j][j];
    for (k = 1; k < j; k++)
      s -= sqr(L[j][k]);
    if (s < 0) {
      fprintf(stderr, "s < 0\n");
      exit(0);
    }
    L[j][j] = sqrt(s);
    /* L[i][j] (i>j) の計算 */
    for (i = j + 1; i <= n; i++) {
      s = a[i][j];
      for (k = 1; k < j; k++)
        s -= L[i][k] * L[j][k];
      L[i][j] = s / L[j][j];
    }
  }
}
```

なお、文献によっては (例えば森 [5], Trefethen and Bau III [9], Higham [1] — おっと、参照した本のほとんどすべてがそうなっている)、$A = LL^T$ でなくて、$A = R^T R$ の形で記述している (つまり $R = L^T$ であり、対角線の上側ですべてを記述する) 場合もある。その場合の計算は単に上の記述で i と j を交換するだけのことであるが、一応書いておくと

Cholesky 分解 $A = R^T R$ の計算

$R^T R = A$ を成分で表わすと、$i \geq j$ なる (i, j) について

$$r_{1i} r_{1j} + r_{2i} r_{2j} + \cdots + r_{ii} r_{ij} = a_{ij}$$

これから

$$r_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} r_{kj}^2}, \quad r_{ij} = \left(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj} \right) / r_{ii} \quad (i > j).$$

(A の成分 a_{ij} についても、やはり対角線の上側だけしか使っていないことに注意。)

2.5 確認用数値実験

正の対角成分を持つ下三角行列 L を作って、$A = LL^T$ を計算し、A の Cholesky 分解を求めて、L と一致することを確認する。

2.5.1 ソースプログラム cholesky1.c

```
 1  /*
 2   * cholesky1.c
 3   */
 4
 5  #include <stdio.h>
 6  #include <math.h>
 7  #include <matrix.h>
 8  #include <limits.h> /* INT_MAX のため */
 9
10  /* [0,1) 範囲の一様疑似乱数 */
11  double drandom()
12  {
13    return random() / (double) INT_MAX;
14  }
15
16  /* 行列の積 */
17  void mul_mm(int n, matrix ab, matrix a, matrix b)
18  {
19    int i, j, k;
20    double s;
21    for (i = 1; i <= n; i++)
22      for (j = 1; j <= n; j++) {
23        s = 0;
24        for (k = 1; k <= n; k++)
25          s += a[i][k] * b[k][j];
26        ab[i][j] = s;
27      }
28  }
29
30  /* 行列の表示 */
31  void print_matrix(int n, matrix a)
32  {
33    int i, j;
34    for (i = 1; i <= n; i++) {
35      for (j = 1; j <= n; j++)
36        printf("%f ", a[i][j]);
37      printf("\n");
38    }
39  }
40
41  void clear_m(int n, matrix a)
42  {
43    int i, j;
44    for (i = 1; i <= n; i++)
45      for (j = 1; j <= n; j++)
46        a[i][j] = 0;
47  }
48
49  /* 自乗 */
50  double sqr(double x) { return x * x; }
51
52  /* Cholesky 分解 */
53  void cholesky(int n, matrix L, matrix a)
54  {
```

```
 55    int i, j, k;
 56    double s;
 57    for (i = 1; i <= n; i++) {
 58      for (j = 1; j < i; j++) {
 59        s = a[i][j];
 60        for (k = 1; k < j; k++)
 61          s -= L[i][k] * L[j][k];
 62        L[i][j] = s / L[j][j];
 63      }
 64      /* L[i][i] の計算 */
 65      s = a[i][i];
 66      for (k = 1; k < i; k++)
 67        s -= sqr(L[i][k]);
 68      if (s < 0) {
 69        fprintf(stderr, "s < 0\n");
 70        exit(0);
 71      }
 72      L[i][i] = sqrt(s);
 73    }
 74  }
 75
 76  int main()
 77  {
 78    int i, j, N;
 79    matrix a, L, Lt;
 80    printf("N="); scanf("%d", &N);
 81    a = new_matrix(N+1, N+1);
 82    L = new_matrix(N+1, N+1);
 83    Lt = new_matrix(N+1, N+1);
 84    for (i = 1; i <= N; i++) {
 85      for (j = 1; j <= i; j++)
 86        Lt[j][i] = L[i][j] = drandom();
 87      for (j = i + 1; j <= N; j++)
 88        Lt[j][i] = L[i][j] = 0.0;
 89    }
 90    mul_mm(N, a, L, Lt);
 91    printf("L=\n");
 92    print_matrix(N, L);
 93    printf("a=\n");
 94    print_matrix(N, a);
 95
 96    clear_m(N, L);
 97    cholesky(N, L, a);
 98    printf("L=\n");
 99    print_matrix(N, L);
100
101    return 0;
102  }
```

2.5.2 cholesky1 の実行結果

```
oyabun% ./cholesky1
N=4
L=
0.968071 0.000000 0.000000 0.000000
0.066731 0.478281 0.000000 0.000000
0.909534 0.351692 0.932534 0.000000
0.654436 0.021070 0.512205 0.202019
a=
0.937162 0.064600 0.880494 0.633540
0.064600 0.233206 0.228902 0.053748
0.880494 0.228902 1.820559 1.080290
0.633540 0.053748 1.080290 0.731896
L=
0.968071 0.000000 0.000000 0.000000
0.066731 0.478281 0.000000 0.000000
0.909534 0.351692 0.932534 0.000000
0.654436 0.021070 0.512205 0.202019
oyabun%
```

2.5.3 jik とループを回す Cholesky 分解のソースプログラム

変更部分のみ示す。

```
 1  /* Cholesky 分解 */
 2  void cholesky(int n, matrix L, matrix a)
 3  {
 4    int i, j, k;
 5    double s;
 6    for (j = 1; j <= n; j++) {
 7      /* L[j][j] の計算 */
 8      s = a[j][j];
 9      for (k = 1; k < j; k++)
10        s -= sqr(L[j][k]);
11      if (s < 0) {
12        fprintf(stderr, "s < 0\n");
13        exit(0);
14      }
15      L[j][j] = sqrt(s);
16      for (i = j + 1; i <= n; i++) {
17        s = a[i][j];
18        for (k = 1; k < j; k++)
19          s -= L[i][k] * L[j][k];
20        L[i][j] = s / L[j][j];
21      }
22    }
23  }
```

2.5.4 jik 版の実行結果

```
oyabun% ./cholesky2
N=4
L=
0.968071 0.000000 0.000000 0.000000
0.066731 0.478281 0.000000 0.000000
0.909534 0.351692 0.932534 0.000000
0.654436 0.021070 0.512205 0.202019
a=
0.937162 0.064600 0.880494 0.633540
0.064600 0.233206 0.228902 0.053748
0.880494 0.228902 1.820559 1.080290
0.633540 0.053748 1.080290 0.731896
L=
0.968071 0.000000 0.000000 0.000000
0.066731 0.478281 0.000000 0.000000
0.909534 0.351692 0.932534 0.000000
0.654436 0.021070 0.512205 0.202019
oyabun%
```

2.6 上三角部分を求める Cholesky 分解のソースプログラム

$U = L^T$ であるから、L を求めるプログラムの L[i][j] を u[j][i] に変える という要領で良い。ただし、A の成分についても対角線の下側でなくて上側を参照するようにしておくべきかもしれない (すべてを対角線の上側で記述するように統一する、という意味)。

```
 1  /* Cholesky 分解 A=U^T U (A の対角線の上半分のみ参照) */
 2  void cholesky(int n, matrix U, matrix a)
 3  {
 4    int j, i, k;
 5    double s;
 6    for (i = 1; i <= n; i++) {
 7      /* U[i][i] の計算 */
 8      s = a[i][i];
 9      for (k = 1; k < i; k++)
10        s -= sqr(U[k][i]);
11      if (s < 0) {
12        fprintf(stderr, "s < 0\n");
13        exit(0);
14      }
15      U[i][i] = sqrt(s);
16      for (j = i + 1; j <= n; j++) {
17        s = a[i][j];
18        for (k = 1; k < i; k++)
19          s -= U[k][j] * U[k][i];
20        U[i][j] = s / U[i][i];
21      }
22    }
23  }
```

2.7 Cholesky 分解の存在証明 (2)

この説明は Trefethen and Bau III [9] による。

(整理中) A の $(1,1)$ 成分が 1 であるときの Gauss の消去法の第 1 ステップは

$$A = \begin{pmatrix} 1 & w^T \\ w & K \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ w & I \end{pmatrix} \begin{pmatrix} 1 & w^T \\ 0 & K - ww^T \end{pmatrix}$$

となる。対称性を保つために

$$\begin{pmatrix} 1 & w^T \\ 0 & K - ww^T \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & K - ww^T \end{pmatrix} \begin{pmatrix} 1 & w^T \\ 0 & I \end{pmatrix}$$

と分解してまとめると、

$$A = \begin{pmatrix} 1 & 0 \\ w & I \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & K - ww^T \end{pmatrix} \begin{pmatrix} 1 & w^T \\ 0 & I \end{pmatrix}.$$

$$A = \left(\begin{array}{c|c} a_{11} & w^T \\ \hline w & B \end{array}\right)$$

のようにブロックわけすると、$\alpha = \sqrt{a_{11}}$ として、

$$A = \begin{pmatrix} \alpha & 0 \\ w/\alpha & I \end{pmatrix} \begin{pmatrix} 1 & 0^T \\ 0 & K - ww^T/a_{11} \end{pmatrix} \begin{pmatrix} \alpha & 0 \\ w/\alpha & I \end{pmatrix} = R_1^T A_1 R_1$$

と分解できる。これを帰納的に

$$A_1 = R_2^T A_2 R_2, \quad A_2 = R_3^T A_3 R_3, \quad \cdots, A_{N-1} = R_N^T A_N R_N$$

と続けると、$A_N = I$ (N 次単位行列) となるので、

$$A = R_1^T R_2^T \cdots R_N^T R_N \cdots R_2 R_1.$$

そこで $R = R_N \cdots R_2 R_1$ とおくと、

$$A = R^T R$$

となる。

Cholesky 分解のアルゴリズム

$R = A$
for $k = 1$ to m
  for j=k+1 to m
    $R_{j,j:m} = R_{j,j:m} - R_{k,j:m}\overline{R}_{kj}/R_{kk}$
$R_{k,k:m} = R_{k,k:m}/\sqrt{R_{kk}}$

2.8 計算量

乗算の回数は $N^3/6+O(N^2)$, 四則演算の回数は $N^3/3+O(N^2)$ であり、通常の Gauss の消去法の半分であると言える。

平方根を計算する回数は N 回である。これを嫌う人もいるようだが、$O(N)$ であるし、平方根の計算時間は例えば除算の計算時間と比べて、それほど長いわけでないので、大したことはないであろう。それよりは、通常の LU 分解では、L または U の一方の対角成分はすべて 1 であるのに、Cholesky 分解ではそうでないので、分解後に連立 1 次方程式を解く場合に、除算の回数が N 回多くなることの方が嫌みであろう ($O(N^2)$ に対する N は積もれば無視できない)。これが気になる場合は修正 Cholesky 分解を採用するのが良い。

2.9 安定性

この小節の説明は Trefethen and Bau III [9] による (アルゴリズムの定義などもその本によった)。Cholesky 分解のアルゴリズムについては、2.7 で説明したものを採用したとする。

Cholesky 分解では、通常の Gauss の消去法にあった難点のすべてが消えてしまう。Cholesky 分解は ピボッティングなし で常に後退安定である (以下の定理 2.2 を見よ)。

因子 R は決して大きくならない。2 ノルムで考えると

$$\|R\|_2 = \|R^T\|_2 = \|A\|^{1/2} \quad \text{(証明は特異値分解)}$$

であるから、p ノルム $(1 \le p \le \infty)$ で考えても、せいぜい $\sqrt{N}$ だけ変わるだけである。

$$\frac{1}{\sqrt{N}}\|A\|^{1/2} \le \|R\| = \|R^T\| \le \sqrt{N}\|A\|^{1/2}.$$

(Cf. 通常の LU 分解では、partial pivoting をしても ($\|L\|_\infty = 1$ とした場合) $\|U\|_\infty = 2^{N-1}\|A\|_\infty$ となることがあり得た。)

定理 2.2 (Cholesky 分解の後退安定性) A は N 次正値対称行列で、浮動小数点演算の基本公理 (A), (B) を満たすコンピューターで Cholesky 分解すると、計算機イプシロン ε_M が十分小さいとき、計算は必ず完了し、その結果を $\widetilde{R}$ とすると、

$$(4) \qquad \exists \delta A \in M(N;\mathbf{R}) \quad \text{s.t.} \quad \widetilde{R}^T\widetilde{R} = A + \delta A, \quad \frac{\|\delta A\|}{\|A\|} = O(\varepsilon_M) \quad (\varepsilon_M \downarrow 0).$$

注意 2.3 この定理は $O(\cdot)$ を使って書いてあって、精度保証には使えない。Higham [1] の p.197 の Theorem 10.3 には

$$|\delta A| \le \gamma_{N+1}|\widetilde{R}^T|\,|\widetilde{R}|$$

という評価が載っている。ここで γ_n は数値線形代数で標準的に使われる記号で、一般には

$$\gamma_n = \frac{nu}{1-nu} \quad (u \text{ は the unit round off})$$

で定義される量である。IEEE 754 の場合には $u = \varepsilon_M$ としてよい。■

もちろん、上の定理は $\widetilde{R}$ が、真の因子 R に近いことを意味しない。これについて言えるのはせいぜい

$$\frac{\|\widetilde{R}-R\|}{\|R\|}=O(\kappa(A)\varepsilon_M)$$

という評価ぐらいである。

2.10 $Ax=b$ を解く

分解が得られてしまえば、連立 1 次方程式が簡単に解けるのは、通常の LU 分解と同じである。念のため一応説明しておく。

A が正値対称行列であるとき、方程式 $Ax=b$ を解くには、まず A を $A=LL^T$ と Cholesky 分解して、 方程式を

$$LL^Tx=b$$

と書き直す。まず「前進消去」で y についての方程式

$$Ly=b$$

を解いてから、後退代入で

$$L^Tx=y$$

を x について解けば良い。計算量は $O(N^2)$ である。

この過程は (もちろん) 後退安定である。

定理 2.4 (Cholesky 分解による連立 1 次方程式の求解の後退安定性) 正値対称行列を係数とする連立 1 次方程式 $Ax=b$ を、浮動小数点演算の基本公理 (A), (B) を満たすコンピューターを用いて、Cholesky 分解で解くアルゴリズムは後退安定である。すなわち計算された解 $\widetilde{x}$ は

$$\exists\Delta A\in M(N;\mathbf{R})\quad\text{s.t.}\quad(A+\Delta A)\widetilde{x}=b,\quad\frac{\|\Delta A\|}{\|A\|}=O(\varepsilon_M)$$

を満たす。

注意 2.5 Higham [1] によると以下の評価が成り立つ。まず p.198, Theorem 10.4 には、絶対値行列の記法を用いた ($O(\cdot)$ でない) 具体的な評価

$$|\Delta A|\le\gamma_{3N+1}|\widetilde{R}^T|\,|\widetilde{R}|$$

が載っている。さらには

$$\|\Delta A\|_2\le\gamma_{3N+1}N(1-N\gamma_{N+1})^{-1}\|A\|_2$$

というノルム評価もできる。■

(この結果からすぐに連立 1 次方程式の近似解の誤差評価が出る。それについて書いておくべきだ…)

3　修正 Cholesky 分解

(これは時間が間に合わないので別のノートから取ってきたものです。あまりまとまっていません。)

定理 3.1 (修正 Cholesky 分解) N 次対称行列 $A = (a_{ij})$ の主座小行列式 δ_k ($k = 1, 2, \cdots, N$) がすべて 0 でないならば、一意的に

$$A = LDL^T \quad (L \text{ は単位下三角行列, } D \text{ は対角行列})$$

と分解できる。

証明　行列を

$$L = \begin{pmatrix} 1 & & & & \\ \ell_{21} & 1 & & \mathbf{0} & \\ \ell_{31} & \ell_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ \ell_{N1} & \ell_{N2} & \cdots & \ell_{N,N-1} & 1 \end{pmatrix}, \quad D = \begin{pmatrix} d_1 & & & \mathbf{0} \\ & d_2 & & \\ & & \ddots & \\ \mathbf{0} & & & d_N \end{pmatrix}$$

とおくと、LDL^T の第 (i, j) 成分は

$$(\ell_{i,1}\ \ell_{i,2}\ \cdots\ \ell_{i,i-1}\ 1\ 0\ \cdots\ 0) \begin{pmatrix} d_1\ell_{j1} \\ d_2\ell_{j2} \\ \vdots \\ d_{j-1}\ell_{j,j-1} \\ d_j \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \sum_{k=1}^{\min\{i,j\}} d_k\ell_{ik}\ell_{jk} \quad (\text{ただし } \ell_{ii} = 1).$$

これが A に等しいとするのだが、対称性から $i \le j$ についてだけ条件を書けばよい。$\ell_{ii} = 1$ に注意すると

$$a_{ij} = \sum_{k=1}^{i} d_k\ell_{ik}\ell_{jk} = \begin{cases} d_i + \displaystyle\sum_{k=1}^{i-1} d_k\ell_{ik}^2 & (j = i) \\ \displaystyle\sum_{k=1}^{i-1} d_k\ell_{ik}\ell_{jk} + d_i\ell_{ji} & (j = i+1, i+2, \cdots, N) \end{cases}$$

を得る。これから $i = 1, 2, \cdots, N$ の順に

$$\begin{aligned} d_i &= a_{ii} - \sum_{k=1}^{i-1} d_k\ell_{ik}^2, \\ \ell_{ji} &= \frac{1}{d_i}\left(a_{ij} - \sum_{k=1}^{i-1} d_k\ell_{ik}\ell_{jk}\right) \quad (j = i+1, \cdots, N) \end{aligned}$$

で $\{d_i\}, \{\ell_{ji}\}_{i\le j}$ が求まる。実際 d_i は Gauss の消去法を i 段まで進めた時の枢軸 $d_{ii}^{(i)}$ に等しいので $\neq 0$ であるから。 ■

最初のうち

$$
\begin{aligned}
d_1 &= a_{11} = \delta_1 \neq 0, \quad \ell_{j1} = a_{j1}/d_1 \quad (j = 2, \cdots, N), \\
d_2 &= a_{22} - d_1\ell_{21}^2 = \delta_2/\delta_1 \neq 0, \quad \ell_{j2} = (a_{2j} - d_1\ell_{21}\ell_{j1})/d_2 \quad (j = 3, \cdots, N), \\
d_3 &= a_{33} - d_1\ell_{31}^2 - d_2\ell_{32}^2 = \delta_3/\delta_2 \neq 0, \quad \ell_{j3} = (a_{3j} - d_1\ell_{31}\ell_{j1} - d_2\ell_{32}\ell_{j2})/d_3 \quad (j = 4, \cdots, N),
\end{aligned}
$$

これを A の修正 **Cholesky** 分解とよぶ。

対称性を利用してうまく計算すると、通常の LU 分解の約半分の計算量で分解できる (森 [6] などを見よ)。

4 方程式を解く

(念のため) Cholesky 分解は LU 分解の一種であるから、係数行列 A の Cholesky 分解 $A = L^TL$ を知っていれば、三角行列を係数とする連立 1 次方程式

$$Ly = b,$$

$$L^Tx = y$$

を順に解くだけで $Ax = b$ の解が求まる。

```
 1  /* 三角行列を係数とする連立一次方程式 */
 2  /* Ux=b を解く */
 3  void solve_uxb(int n, matrix u, vector b)
 4  {
 5    int i, j;
 6    for (i = n; i >= 1; i--) {
 7      for (j = i + 1; j <= n; j++)
 8        b[i] -= u[i][j] * b[j];
 9      b[i] /= u[i][i];
10    }
11  }
12
13  /* L^T x=b を解く … solve_uxb() の u[i][j] を L[j][i] に */
14  void solve_ltxb(int n, matrix L, vector b)
15  {
16    int i, j;
17    for (i = n; i >= 1; i--) {
18      for (j = i + 1; j <= n; j++)
19        b[i] -= L[j][i] * b[j];
20      b[i] /= L[i][i];
21    }
22  }
23
24  /* Lx=b を解く */
25  void solve_lxb(int n, matrix L, vector b)
26  {
27    int i, j;
28    for (i = 1; i <= n; i++) {
29      for (j = 1; j < i; j++)
30        b[i] -= L[i][j] * b[j];
31      b[i] /= L[i][i];
32    }
33  }
34
35  /* U^T x=b を解く */
36  void solve_utxb(int n, matrix U, vector b)
37  {
38    int i, j;
39    for (i = 1; i <= n; i++) {
40      for (j = 1; j < i; j++)
41        b[i] -= U[j][i] * b[j];
42      b[i] /= U[i][i];
43    }
44  }
```

```
oyabun% ./cholesky3
N=5
L=
0.968071 0.000000 0.000000 0.000000 0.000000
0.066731 0.478281 0.000000 0.000000 0.000000
0.909534 0.351692 0.932534 0.000000 0.000000
0.654436 0.021070 0.512205 0.202019 0.000000
0.939977 0.204082 0.378829 0.793114 0.288201
a=
0.937162 0.064600 0.880494 0.633540 0.909965
0.064600 0.233206 0.228902 0.053748 0.160334
0.880494 0.228902 1.820559 1.080290 1.279986
0.633540 0.053748 1.080290 0.731896 0.973717
0.909965 0.160334 1.279986 0.973717 1.780807
L=
0.968071 0.000000 0.000000 0.000000 0.000000
0.066731 0.478281 0.000000 0.000000 0.000000
0.909534 0.351692 0.932534 0.000000 0.000000
0.654436 0.021070 0.512205 0.202019 0.000000
0.939977 0.204082 0.378829 0.793114 0.288201
x=
1.000000 2.000000 3.000000 4.000000 5.000000
oyabun%
```

A 帯行列用 Cholesky 分解

A.1 C 版

```
 1  /*
 2   * band_cholesky.c --- 正定値対称帯行列 A の Cholesky 分解
 3   */
 4
 5  #include <stdio.h>
 6  #include <limits.h>
 7  #include <math.h>
 8  #include <matrix.h>
 9
10  double drandom() { return random() / (double) INT_MAX; }
11  double sqr(double x) { return x * x; }
12  double max(double a, double b) { return (a > b) ? a : b; }
13  double min(double a, double b) { return (a < b) ? a : b; }
14
15  /* 節約版 Cholesky 分解 A=U^T U (A, U は対角線の上半分のみ与える)
16   *
17   * 半バンド幅 m の n 次正定値対称帯行列 A=(a_{ij}) (1 ≦ i ≦ n, 1 ≦ j ≦ n) を
18   *   ASB=(asb_{ij}) (1 ≦ i ≦ n, 1 ≦ j ≦ m+1)
19   * に詰めこむ
20   *
21   *   asb[i][j-i+1] = a[i][j]  (1 ≦ i ≦ n, i ≦ j ≦ min(i+m,n))
22   *
23   * 結果の因子 U は ASB に上書きする。
24   *
```

```
25   */
26  void band_cholesky(int n, int m, matrix ASB)
27  {
28    int j, i, k;
29    double s;
30    for (i = 1; i <= n; i++) {
31      /* U[i][i] の計算 */
32      s = ASB[i][1];
33      for (k = max(1,i-m); k < i; k++)
34        s -= sqr(ASB[k][i-k+1]);
35      if (s < 0) {
36        fprintf(stderr, "s < 0\n");
37        exit(0);
38      }
39      ASB[i][1] = sqrt(s);
40      /* U[i][j] (i < j) の計算 */
41      for (j = i + 1; j <= min(i + m, n); j++) {
42        s = ASB[i][j-i+1];
43        for (k = max(1,j-m); k < i; k++)
44          s -= ASB[k][j-k+1] * ASB[k][i-k+1];
45        ASB[i][j-i+1] = s / ASB[i][1];
46      }
47    }
48  }
49
50  /* 三角行列を係数とする連立一次方程式 */
51  /* U^T x=b を解く */
52  void solve_band_utxb(int n, int m, matrix UB, vector b)
53  {
54    int i, j;
55    for (i = 1; i <= n; i++) {
56      for (j = max(i-m,1); j < i; j++)
57        b[i] -= UB[j][i-j+1] * b[j];
58      b[i] /= UB[i][1];
59    }
60  }
61
62  /* Ux=b を解く */
63  void solve_band_uxb(int n, int m, matrix UB, vector b)
64  {
65    int i, j;
66    for (i = n; i >= 1; i--) {
67      for (j = i + 1; j <= min(i+m,n); j++)
68        b[i] -= UB[i][j-i+1] * b[j];
69      b[i] /= UB[i][1];
70    }
71  }
72
73  /* パックしてある帯行列を表示する */
74  void print_symmetric_band_matrix1(int n, int m, matrix asb)
75  {
76    int i, j;
77    for (i = 1; i <= n; i++) {
78      for (j = 1; j <= n; j++)
79         if (abs(j - i) > m)
80           printf("%f ", 0.0);
```

```
81          else {
82            if (j >= i)
83              printf("%f ", asb[i][j-i+1]);
84            else /* 要するに i と j を入れ替える */
85              printf("%f ", asb[j][i-j+1]);
86          }
87        printf("\n");
88      }
89    }
90
91    /* 詰め込んだ上三角帯行列を表示する */
92    void print_upper_triangular_band_matrix1(int n, int m, matrix ub)
93    {
94      int i, j;
95      for (i = 1; i <= n; i++) {
96        for (j = 1; j <= n; j++)
97          if (abs(j - i) > m)
98            printf("%f ", 0.0);
99          else {
100           if (j >= i)
101             printf("%f ", ub[i][j-i+1]);
102           else
103             printf("%f ", 0.0);
104         }
105       printf("\n");
106     }
107   }
108
109   void print_vector1(int n, vector x)
110   {
111     int i;
112     for (i = 1; i <= n; i++)
113       printf("%f ", x[i]);
114     printf("\n");
115   }
116
117   int main()
118   {
119     int n, m, i, j, k;
120     double s;
121     matrix ASB, UB;
122     vector x, b;
123
124     /* 未知数の個数, 半バンド幅 */
125     n = 8; m = 3;
126
127     /* ASB, UB は n × (m+1) 型の行列に詰め込める */
128     ASB = new_matrix(n + 1, (m+1) + 1);
129     UB = new_matrix(n + 1, (m+1) + 1);
130     x = new_vector(n + 1);
131     b = new_vector(n + 1);
132
133     /* 上三角行列 U */
134     for (i = 1; i <= n; i++)
135       for (j = i; j <= min(i+m, n); j++)
136         UB[i][j-i+1] = drandom();
```

```
137    printf("U=\n");
138    print_upper_triangular_band_matrix1(n, m, UB);
139
140    /* A=U^T U を計算する。対称なので対角線の上側だけ計算する。 */
141    for (i = 1; i <= n; i++)
142      for (j = i; j <= min(i+m,n); j++) { /* 対角線の上だけ */
143        s = 0;
144        /* U[k][i]*U[k][j]
145           max(i-m,1) ≦ k ≦ i かつ max(j-m,1) ≦ k ≦ j の場合のみ ≠ 0
146           もともと i ≦ j だから max(j-m,1) ≦ k ≦ i ということ
147         */
148        for (k = max(j-m,1); k <= i; k++)
149          s += UB[k][i-k+1] * UB[k][j-k+1];
150        ASB[i][j-i+1] = s;
151      }
152    printf("A:=U^T U\n");
153    print_symmetric_band_matrix1(n, m, ASB);
154
155    /* 簡単な x */
156    for (i = 1; i <= n; i++)
157      x[i] = i;
158
159    /* b=A x */
160    for (i = 1; i <= n; i++) {
161      s = ASB[i][1] * x[i];
162      for (j = i + 1; j <= min(i+m,n); j++)
163        s += ASB[i][j-i+1] * x[j];
164      for (j = max(i-m,1); j < i; j++)
165        s += ASB[j][i-j+1] * x[j];
166      b[i] = s;
167    }
168
169    /* Cholesky 分解 A=U^T U */
170    band_cholesky(n, m, ASB);
171    printf("Cholesky 分解の結果 U=\n");
172    print_upper_triangular_band_matrix1(n, m, ASB);
173
174    /* U^T U x = b を解く */
175    solve_band_utxb(n, m, UB, b);
176    solve_band_uxb(n, m, UB, b);
177    printf("x=\n");
178    print_vector1(n, b);
179
180    return 0;
181  }
```

```
oyabun% ./band_cholesky
U=
0.968071 0.066731 0.478281 0.909534 0.000000 0.000000 0.000000 0.000000
0.000000 0.351692 0.932534 0.654436 0.021070 0.000000 0.000000 0.000000
0.000000 0.000000 0.512205 0.202019 0.939977 0.204082 0.000000 0.000000
0.000000 0.000000 0.000000 0.378829 0.793114 0.288201 0.267157 0.000000
0.000000 0.000000 0.000000 0.000000 0.356823 0.128374 0.703298 0.737414
0.000000 0.000000 0.000000 0.000000 0.000000 0.802521 0.915231 0.511382
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.762057 0.456381
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.471062
A:=U^T U
0.937162 0.064600 0.463010 0.880494 0.000000 0.000000 0.000000 0.000000
0.064600 0.128141 0.359881 0.290854 0.007410 0.000000 0.000000 0.000000
0.463010 0.359881 1.360726 1.148772 0.501110 0.104532 0.000000 0.000000
0.880494 0.290854 1.148772 1.439862 0.504137 0.150407 0.101207 0.000000
0.000000 0.007410 0.501110 0.504137 1.640353 0.466215 0.462838 0.263126
0.000000 0.000000 0.104532 0.150407 0.466215 0.785229 0.901772 0.505059
0.000000 0.000000 0.000000 0.101207 0.462838 0.901772 1.984379 1.334442
0.000000 0.000000 0.000000 0.000000 0.263126 0.505059 1.334442 1.235474
Cholesky 分解の結果 U=
0.968071 0.066731 0.478281 0.909534 0.000000 0.000000 0.000000 0.000000
0.000000 0.351692 0.932534 0.654436 0.021070 0.000000 0.000000 0.000000
0.000000 0.000000 0.512205 0.202019 0.939977 0.204082 0.000000 0.000000
0.000000 0.000000 0.000000 0.378829 0.793114 0.288201 0.267157 0.000000
0.000000 0.000000 0.000000 0.000000 0.356823 0.128374 0.703298 0.737414
0.000000 0.000000 0.000000 0.000000 0.000000 0.802521 0.915231 0.511382
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.762057 0.456381
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.471062
x=
1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000
oyabun%
```

A.2 C++ & Profil 版

Profil を使うバージョンも作ってみた。

```
 1  /*
 2   * band_cholesky_C++.C --- 正定値対称帯行列 A の Cholesky 分解
 3   */
 4
 5  #include <iostream.h>
 6  #include <iomanip.h>
 7  #include <limits.h>
 8  #include <math.h>
 9  #include <Matrix.h>
10  #include <Vector.h>
11
12  double drandom(void) { return random() / (double) INT_MAX; }
13  double sqr(double x) { return x * x; }
14  // double max(double a, double b) { return (a > b) ? a : b; }
15  // double min(double a, double b) { return (a < b) ? a : b; }
16
17  /* 節約版 Cholesky 分解 A=U^T U (A, U は対角線の上半分のみ与える)
```

```
18   *
19   * 半バンド幅 m の n 次正定値対称帯行列 A=(a_{ij}) (1 ≦ i ≦ n, 1 ≦ j ≦ n) を
20   *   ASB=(asb_{ij}) (1 ≦ i ≦ n, 1 ≦ j ≦ m+1)
21   * に詰めこむ
22   *
23   *   asb[i][j-i+1] = a[i][j]  (1 ≦ i ≦ n, i ≦ j ≦ min(i+m,n))
24   *
25   * 結果の因子 U は ASB に上書きする。
26   *
27   */
28  void band_cholesky(int n, int m, MATRIX &ASB)
29  {
30    double s;
31    for (int i = 1; i <= n; i++) {
32      /* U[i][i] の計算 */
33      s = ASB(i,1);
34      for (int k = max(1,i-m); k < i; k++)
35        s -= sqr(ASB(k,i-k+1));
36      if (s < 0) {
37        cerr << "s < 0" << endl;
38        exit(0);
39      }
40      ASB(i,1) = sqrt(s);
41      /* U[i][j] (i < j) の計算 */
42      for (int j = i + 1; j <= min(i + m, n); j++) {
43        s = ASB(i,j-i+1);
44        for (int k = max(1,j-m); k < i; k++)
45          s -= ASB(k,j-k+1) * ASB(k,i-k+1);
46        ASB(i,j-i+1) = s / ASB(i,1);
47      }
48    }
49  }
50
51  /* 三角行列を係数とする連立一次方程式 */
52  /* U^T x=b を解く */
53  void solve_band_utxb(int n, int m, const MATRIX &UB, VECTOR &b)
54  {
55    for (int i = 1; i <= n; i++) {
56      for (int j = max(i-m,1); j < i; j++)
57        b(i) -= UB(j,i-j+1) * b(j);
58      b(i) /= UB(i,1);
59    }
60  }
61
62  /* Ux=b を解く */
63  void solve_band_uxb(int n, int m, const MATRIX &UB, VECTOR &b)
64  {
65    for (int i = n; i >= 1; i--) {
66      for (int j = i + 1; j <= min(i+m,n); j++)
67        b(i) -= UB(i,j-i+1) * b(j);
68      b(i) /= UB(i,1);
69    }
70  }
71
72  /* パックしてある帯行列を表示する */
73  void print_symmetric_band_matrix1(int n, int m, const MATRIX &asb)
```

```
 74 {
 75   for (int i = 1; i <= n; i++) {
 76     for (int j = 1; j <= n; j++)
 77        if (abs(j - i) > m)
 78          cout << 0.0 << " ";
 79        else {
 80          if (j >= i)
 81    cout << asb(i,j-i+1) << " ";
 82          else /* 要するに i と j を入れ替える */
 83            cout << asb(j,i-j+1) << " ";
 84        }
 85     cout << endl;
 86   }
 87 }
 88
 89 /* 詰め込んだ上三角帯行列を表示する */
 90 void print_upper_triangular_band_matrix1(int n, int m, const MATRIX &ub)
 91 {
 92   for (int i = 1; i <= n; i++) {
 93     for (int j = 1; j <= n; j++)
 94        if (abs(j - i) > m)
 95          cout << 0.0 << " ";
 96        else {
 97          if (j >= i)
 98            cout << ub(i,j-i+1) << " ";
 99          else
100            cout << 0.0 << " ";
101        }
102     cout << endl;
103   }
104 }
105
106 void print_vector1(int n, const VECTOR &x)
107 {
108   for (int i = 1; i <= n; i++)
109     cout << x(i) << " ";
110   cout << endl;
111 }
112
113 int main()
114 {
115   int n, m, i, j, k;
116   double s;
117
118   cout << setprecision(4);
119   cout << setiosflags(ios::fixed);
120   /* 未知数の個数, 半バンド幅 */
121   n = 8; m = 3;
122
123   /* ASB, UB は n × (m+1) 型の行列に詰め込める */
124   MATRIX ASB(n,m+1), UB(n,m+1);
125   VECTOR x(n), b(n);
126
127   /* 上三角行列 U */
128   for (i = 1; i <= n; i++)
129     for (j = i; j <= min(i+m, n); j++)
```

```
130         UB(i,j-i+1) = drandom();
131     cout << "U=" << endl;
132     print_upper_triangular_band_matrix1(n, m, UB);
133
134     /* A=U^T U を計算する。対称なので対角線の上側だけ計算する。 */
135     for (i = 1; i <= n; i++)
136       for (j = i; j <= min(i+m,n); j++) { /* 対角線の上だけ */
137         s = 0;
138         /* U[k](i)*U[k](j)
139            max(i-m,1) ≦ k ≦ i かつ max(j-m,1) ≦ k ≦ j の場合のみ ≠ 0
140            もともと i ≦ j だから max(j-m,1) ≦ k ≦ i ということ
141          */
142         for (k = max(j-m,1); k <= i; k++)
143           s += UB(k,i-k+1) * UB(k,j-k+1);
144         ASB(i,j-i+1) = s;
145       }
146     cout << "A:=U^T U" << endl;
147     print_symmetric_band_matrix1(n, m, ASB);
148
149     /* 簡単な x */
150     for (i = 1; i <= n; i++)
151       x(i) = i;
152
153     /* b=A x */
154     for (i = 1; i <= n; i++) {
155       s = ASB(i,1) * x(i);
156       for (j = i + 1; j <= min(i+m,n); j++)
157         s += ASB(i,j-i+1) * x(j);
158       for (j = max(i-m,1); j < i; j++)
159         s += ASB(j,i-j+1) * x(j);
160       b(i) = s;
161     }
162
163     /* Cholesky 分解 A=U^T U */
164     band_cholesky(n, m, ASB);
165     cout << "Cholesky 分解の結果 U=" << endl;
166     print_upper_triangular_band_matrix1(n, m, ASB);
167
168     /* U^T U x = b を解く */
169     solve_band_utxb(n, m, UB, b);
170     solve_band_uxb(n, m, UB, b);
171     cout << "x=" << endl;
172     print_vector1(n, b);
173
174     return 0;
175   }
```

```
oyabun% g++ -W -I/usr/local/include -o band_cholesky_C++ band_cholesky_C++.C -lProfil -lmatrix
oyabun% ./band_cholesky_C++
U=
0.9681 0.0667 0.4783 0.9095 0.0000 0.0000 0.0000 0.0000
0.0000 0.3517 0.9325 0.6544 0.0211 0.0000 0.0000 0.0000
0.0000 0.0000 0.5122 0.2020 0.9400 0.2041 0.0000 0.0000
0.0000 0.0000 0.0000 0.3788 0.7931 0.2882 0.2672 0.0000
0.0000 0.0000 0.0000 0.0000 0.3568 0.1284 0.7033 0.7374
0.0000 0.0000 0.0000 0.0000 0.0000 0.8025 0.9152 0.5114
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.7621 0.4564
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.4711
A:=U^T U
0.9372 0.0646 0.4630 0.8805 0.0000 0.0000 0.0000 0.0000
0.0646 0.1281 0.3599 0.2909 0.0074 0.0000 0.0000 0.0000
0.4630 0.3599 1.3607 1.1488 0.5011 0.1045 0.0000 0.0000
0.8805 0.2909 1.1488 1.4399 0.5041 0.1504 0.1012 0.0000
0.0000 0.0074 0.5011 0.5041 1.6404 0.4662 0.4628 0.2631
0.0000 0.0000 0.1045 0.1504 0.4662 0.7852 0.9018 0.5051
0.0000 0.0000 0.0000 0.1012 0.4628 0.9018 1.9844 1.3344
0.0000 0.0000 0.0000 0.0000 0.2631 0.5051 1.3344 1.2355
Cholesky 分解の結果 U=
0.9681 0.0667 0.4783 0.9095 0.0000 0.0000 0.0000 0.0000
0.0000 0.3517 0.9325 0.6544 0.0211 0.0000 0.0000 0.0000
0.0000 0.0000 0.5122 0.2020 0.9400 0.2041 0.0000 0.0000
0.0000 0.0000 0.0000 0.3788 0.7931 0.2882 0.2672 0.0000
0.0000 0.0000 0.0000 0.0000 0.3568 0.1284 0.7033 0.7374
0.0000 0.0000 0.0000 0.0000 0.0000 0.8025 0.9152 0.5114
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.7621 0.4564
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.4711
x=
1.0000 2.0000 3.0000 4.0000 5.0000 6.0000 7.0000 8.0000
oyabun%
```

B　浮動小数点演算の基本公理

Trefethen and Bau III [9] に従い、ある理想的な (指数部の範囲の制限をつけないという意味) 浮動小数点システムにおける浮動小数点数の全体 $\mathbf{F}$ について、以下の二つの公理が成り立つことを要請する。

基本公理 (A)

$\mathbf{F}$ への丸め写像 $\mathrm{fl}\colon \mathbf{R} \to \mathbf{F}$ は次を満たす。

$$\forall x \in \mathbf{R} \quad \exists \varepsilon \in \mathbf{R} \quad \text{s.t.} \quad |\varepsilon| \le \varepsilon_M, \quad \mathrm{fl}(x) = x(1+\varepsilon).$$

基本公理 (B)

$\mathbf{F}$ における四則演算 $\oplus$, $\ominus$, $\otimes$, $\oslash$ は次を満たす。

$$\forall * \in \{+, -, \times, \div\}, \quad \forall x, \forall y \in \mathbf{F} \quad \exists \varepsilon \in \mathbf{R} \quad \text{s.t.} \quad |\varepsilon| \le \varepsilon_M, \quad x \mathbin{\circledast} y = (x * y)(1+\varepsilon).$$

C　内積、2次形式の話

C.1　$\mathbf{R}^N$ の標準内積

$\mathbf{R}^N$ の標準的な内積を $(\cdot,\cdot)_{\mathbf{R}^N}$ または誤解のない限り単に $(\cdot,\cdot)$ で表わす。すなわち $x=(x_i)\in\mathbf{R}^N$, $y=(y_i)\in\mathbf{R}^N$ に対して

$$(x,y)_{\mathbf{R}^N}=(x,y)=\sum_{i=1}^{N}x_iy_i.$$

これは y の転置ベクトル y^T と x の積であるとみなすこともできる:

$$(x,y)_{\mathbf{R}^N}=y^Tx.$$

C.2　$\mathbf{R}^N$ 上の双一次形式

$a\colon \mathbf{R}^N\times\mathbf{R}^N\to\mathbf{R}$ が双一次形式であるとは、次の条件が成り立つことである。
各変数について線型、すなわち

1. $\forall x\in\mathbf{R}^N$, $\forall y\in\mathbf{R}^N$, $\forall z\in\mathbf{R}^N$ に対して $a(x+y,z)=a(x,z)+a(y,z)$, $a(x,y+z)=a(x,z)+a(y,z)$.
2. $\forall x\in\mathbf{R}^N$, $\forall y\in\mathbf{R}^N$, $\forall\lambda\in\mathbf{R}$ に対して $a(\lambda x,y)=\lambda a(x,y)$, $a(x,\lambda y)=\lambda a(x,y)$.

N 次正方行列 $A=(a_{ij})\in M(N;\mathbf{R})$ に対して、

$$a(x,y)\overset{\text{def.}}{=}(Ax,y)_{\mathbf{R}^N}$$

とおくと、これは $\mathbf{R}^N$ 上の双一次形式となる。実は $\mathbf{R}^N$ 上の双一次形式はこういう形のもので尽くされる。実際、双一次形式 a に対して、$a_{ij}=a(e_j,e_i)$ とおくと、

$$a\left(\sum_{j=1}^{N}x_je_j,\sum_{i=1}^{N}y_ie_i\right)=\sum_{j=1}^{N}\sum_{i=1}^{N}x_jy_ia(e_j,e_i)=\sum_{i=1}^{N}\left(\sum_{j=1}^{N}a_{ij}x_j\right)y_i=(Ax,y)_{\mathbf{R}^N}.$$

ただし $A=(a_{ij})$ とおいた。

C.3　対称双一次形式

$\mathbf{R}^N$ 上の双一次形式が

$$\forall x\in\mathbf{R}^N\quad\forall y\in\mathbf{R}^N\quad a(x,y)=a(y,x)$$

を満たすとき、a は $\mathbf{R}^N$ 上の対称双一次形式であるという。
$a(x,y)=(Ax,y)_{\mathbf{R}^N}$ とするとき、

$$a(x,y)=(Ax,y)_{\mathbf{R}^N}=y^TAx=(A^Ty)^Tx=(x,A^Ty)_{\mathbf{R}^N}=(A^Ty,x)_{\mathbf{R}^N},$$

$$a(y,x) = (Ay,x)_{\mathbf{R}^N}$$

であるから、

$$a \text{ が対称} \quad \Leftrightarrow \quad \forall x \in \mathbf{R}^N \quad \forall y \in \mathbf{R}^N \quad (A^T y, x)_{\mathbf{R}^N} = (Ay,x)_{\mathbf{R}^N} \quad \Leftrightarrow \quad A = A^T.$$

ゆえに $\mathbf{R}^N$ 上の対称双一次形式は、実対称行列 A を用いて (Ax,y) の形に表わされるもので尽くされることが分かる。

C.4　内積

$\mathbf{R}^N$ 上の内積とは、$\mathbf{R}^N$ 上の双一次形式 $a(\cdot,\cdot)$ で、正値性と呼ばれる次の性質を持つもののことを言う。

$$\forall x \in \mathbf{R}^N \quad a(x,x) \geq 0 \quad \text{かつ} \quad \left(\forall x \in \mathbf{R}^N \quad a(x,x) = 0 \quad \Leftrightarrow \quad x = 0\right).$$

$a(\cdot,\cdot)$ が $\mathbf{R}^N$ 上の正値対称双一次形式であるための必要十分条件は、a が適当な正値対称行列 A を用いて

$$a(x,y) = (Ax,y)_{\mathbf{R}^N}$$

と表わされることである。

C.5　2 次形式

$\boldsymbol{a}\colon \mathbf{R}^N \to \mathbf{R}$ が R^N 上の 2 次形式であるとは、2 次同次多項式関数であることを言う。これは適当な $A = (a_{ij})$ を用いて

$$\boldsymbol{a}(x) = \sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij} x_i x_j,$$

すなわち

$$\boldsymbol{a}(x) = (Ax,x)_{\mathbf{R}^N}$$

と書けるということである。

$$(Ax,x)_{\mathbf{R}^N} = (x, A^T x)_{\mathbf{R}^N} = (A^T x, x)_{\mathbf{R}^N}$$

であるから、A の代りに A^T を用いても同じであるし、さらに

$$(Ax,x)_{\mathbf{R}^N} = \frac{1}{2}((Ax,x)_{\mathbf{R}^N} + (A^T x,x)_{\mathbf{R}^N}) = \left(\frac{1}{2}(A + A^T)x, x\right)_{\mathbf{R}^N}$$

であるから、A の対称部分 $(A + A^T)/2$ を用いても同じである（「2 次形式 $x \mapsto (Ax,x)$ の値には、実は A の対称部分しか効いてこない」）。そこで通常は係数行列としては、実対称行列に限ることにする。こうすると、2 次形式全体と実対称行列全体が一対一対応するし、

$$\nabla \frac{1}{2}(Ax,x) = \frac{1}{2}(A + A^T)x = Ax$$

のような簡単な公式が成り立つ。

最初に述べた定義から、$\mathbf{R}^N$ 上の 2 次形式とは、$\mathbf{R}^N$ 上の対称双一次形式の変数 (x, y) に (x, x) を代入したものであるとみなせることが分かるが、対称双一次形式について成り立つ

$$a(x,y) = \frac{1}{2}\left(a(x+y, x+y) - a(x,x) - a(y,y)\right) = \frac{1}{2}\left(\boldsymbol{a}(x+y) - \boldsymbol{a}(x) - \boldsymbol{a}(y)\right)$$

という公式を見ると、行列表現をしなくても 2 次形式から直接双一次形式の値が計算できることが分かる。

C.6 変数変換

実対称行列 $A = (a_{ij}) \in M(N; \mathbf{R})$ 係数行列に持つ 2 次形式

$$f(x) = (Ax, x) = x^T A x$$

に対して、$P \in GL(N; \mathbf{R})$, $x = Py$ という変数変換を行うと

$$f(x) = f(Py) = (Py)^T A(Py) = y^T P^T A P y = (P^T A P y, y) = (A' y, y).$$

つまり $A' := P^T A P$ を係数行列とする 2 次形式に変換される (A' は対称行列であることに注意)。A' が対角行列 $\mathrm{diag}\,(\lambda_1, \cdots, \lambda_N)$ に等しくなったとすると、

$$f(x) = \sum_{i=1}^{N} \lambda_i y_i^2.$$

この形にしたときに、λ_i の符号がどうなるかは、実は A だけで決まってしまう。

定理 C.1 (Sylvester の慣性律 (Sylvester's law of inertia)) A を n 次実対称行列とする。P が $D := P^T A P$ が対角行列であるような n 次正則行列であるとするとき、D の対角成分のうちで負であるものの個数と正であるものの個数は、A だけで定まる。

D Sylvester の慣性律

(この節は下書きです。この文書に入れるよりは、固有値問題の文書に入れる方が良いかもしれない。もしかしたら移します。 2013/8/26 10:52:43)

D.1 本題に入る前に

自分自身が最初のうち景色が見えなかったので、色々書いてみる。

Sylvester の慣性律とは、2 次形式を平方完成したときの係数の符号が、平方完成のやり方 (それはたくさんある) によらずに決まる、という事実を定式化したものである。

平方完成のことを「2 次形式の対角化」ともいう。それは議論を行列の言葉で書くことが出来て、与えられた対称行列 (2 次形式の係数行列) A に対して、適当な正則行列 P を見つけて、P^TAP を対角行列にする操作に相当しているからである。

この変換 $A \mapsto P^TAP$ は、相似変換 $A \mapsto P^{-1}AP$ と似ているが、一応は別物であることに注意する (前者は正則な線形変数変換によるもので、後者は基底の取り替えによるものである)。ややこしいのは、対称行列の固有値問題では、直交行列 ($P^T = P^{-1}$ を満たす行列) で対角化するため、両者が重なってしまうことである (個人的にはかなり混乱したところ)。

何に使われるかについても、あらかじめ、少しは知っておいた方が良いかもしれない。

大学 1,2 年次に遭遇する適用例としては、多変数関数の極値問題を微分法で扱う際に、2 階までの Taylor 展開をしたとき、2 次の項が 2 次形式になっている、というのがある (何にでもかいてあるが、例えば桂田 [4])。f が n 変数関数で、内点 a で極値を取るには、$f'(a) = 0$ が必要で、そのとき

$$f(a+h) = f(a) + h \text{ の 2 次形式} + O(\|h\|^2) \quad (h \to 0).$$

そこで 2 次形式の符号に興味が出て来るわけである。どういう場合があって、どうやれば判定できるか。他にも Morse の Lemma などで、関数の「様子」を理解するために使われる。

数値線形代数においては、固有値を求めるための 2 分法というアルゴリズムが、Sylvester の慣性律で理解出来る、という応用もある。

D.2 見掛けが少し異なる 3 つの定理

以下に現れる行列は (時々「実」を書き漏らすけれど) すべて実行列とする。

実対称行列 A に対して、A の固有値のうちで正であるものの個数、負であるものの個数、0 であるものの個数をそれぞれ $\pi(A), \nu(A), \zeta(A)$ と書くことにする。

A は n 次正方行列として、

$$\pi(A), \nu(A), \zeta(A) \geq 0, \quad \pi(A) + \nu(A) + \zeta(A) = n$$

である。

まず実対称行列の実直交行列による対角化に関する常識的事項 (線形代数で習うはず) を復習する (これがなくても Sylvester の慣性律の証明は出来るわけだが、使った方が私には見通しが良いので…)。

A が n 次実対称行列ならば、実直交行列 U ($U^{-1} = U^T$) と実数 $\lambda_1, \cdots, \lambda_n$ が存在して、

$$U^TAU = \operatorname{diag}(\lambda_1, \cdots, \lambda_n).$$

ゆえに $\forall x \in \mathbf{R}^n$ に対して $y := U^{-1}x$ とおくと、$x = Uy$ で、

$$(Ax, x) = (AUy, Uy) = (U^TAUy, y) = \sum_{j=1}^{n} \lambda_j y_j^2.$$

こうして 2 次形式 (Ax, x) を平方和 — 定数 $\times$ (何か)^2 の和 — に変形できる。平方完成と呼んでも良いだろうが、これを 2 次形式の対角化という。標語的に

と言っても良いであろう (筋の通った主張にするには色々言葉を足さなければいけないが、迷子にならないために手短に言い切っておく)。

ところで、上の対角化の議論の要点は

(i) U^TAU が対角行列 $\mathrm{diag}\,(\lambda_1,\cdots,\lambda_n)$ である

(ii) U は正則

の二つである、と言えるだろう。

(Ax,x) が平方和の形に書けるためには、U が実直交行列であることは必要ない (だから以下では U でなく、P と書いたりする)。

U が実直交行列でなければ、λ_j は A の固有値とは限らないが、λ_j $(j=1,\cdots,n)$ のうちの正であるものの個数、負であるものの個数は、A で定まる。それを定式化したものが、次の Sylvester の慣性律と呼ばれる定理である。

定理 D.1 (Sylvester の慣性律, (Sylvester's law of inertia)) 任意の実対称行列 A に対して、正則行列 P があって、$P^TAP=\mathrm{diag}\,(\lambda_1,\cdots,\lambda_n)$ となるとき、

$$\begin{aligned} n_p &:= \sharp\{j\in\{1,\cdots,n\};\lambda_j>0\},\\ n_n &:= \sharp\{j\in\{1,\cdots,n\};\lambda_j<0\},\\ n_z &:= \sharp\{j\in\{1,\cdots,n\};\lambda_j=0\} \end{aligned}$$

($\sharp$ は集合の要素数を表す)

は P に依らず、A だけで定まる。(実対称行列が正則行列で対角化されるとき、対角成分のうち正であるものの個数、負であるものの個数、0 であるものの個数は、正則行列によらず、A だけで定まる。)

この定理は、少し後で紹介する「良くテキストに載っている証明」からすると自然な主張であるが、やや正体が分かりづらいのでは？と思う。次の定理とセットにして覚えることを勧める。

定理 D.2 (老婆心版 Sylvester の慣性律) 任意の実対称行列 A に対して、正則行列 P があって、$P^TAP=\mathrm{diag}\,(\lambda_1,\cdots,\lambda_n)$ となるとき、

$$\begin{aligned} n_p &:= \sharp\{j\in\{1,\cdots,n\};\lambda_j>0\},\\ n_n &:= \sharp\{j\in\{1,\cdots,n\};\lambda_j<0\},\\ n_z &:= \sharp\{j\in\{1,\cdots,n\};\lambda_j=0\} \end{aligned}$$

とおくと、

$$n_p=\pi(A),\quad n_n=\nu(A),\quad n_z=\zeta(A).$$

証明　定理 D.1 を認めた上での証明を与える。P として $U^TAU = \mathrm{diag}[\lambda_1, \cdots, \lambda_n]$ となるような実直交行列 U を取れば、$P^TAP = U^TAU = U^{-1}AU$ は A の相似変換であり、$\lambda_1, \cdots, \lambda_n$ は A の固有値である。ゆえに n_p, n_n, n_z はそれぞれ $\pi(A), \nu(A), \zeta(A)$ に等しい。 ■

杉原・室田 [8] には、次の形の Sylvester の慣性律が載っている。

定理 D.3 (杉原・室田版 Sylvester の慣性律) A を実対称行列とするとき，任意の正則行列 P に対して、$B := P^TAP$ とおくと、

$$\pi(A) = \pi(B), \quad \nu(A) = \nu(B), \quad \zeta(A) = \zeta(B)$$

が成り立つ。

結論部分が「〜に依らず、〜で定まる」でなくて、具体的な等式である点は、定理 D.2 と同じである。対角化でなくて、変換した行列 B の固有値の話にしてしまうのも、少なくとも彼らの目的 (固有値の数値計算アルゴリズムの議論をする) にとっては使いやすいようである。

ここでは、定理 D.2 と定理 D.3 が同等であること (一方を認めれば他方がすぐに導かれること) を見てみよう。

まず、定理 D.3 を認めよう。n 次実対称行列 A, 正則行列 P に対して、$\exists\lambda_1, \cdots, \lambda_n$ s.t. $P^TAP = \mathrm{diag}[\lambda_1, \cdots, \lambda_n]$ となったとすると、$B := P^TAP$ の固有値は (対角成分である) $\lambda_1, \cdots, \lambda_n$ であるから、

$$n_p = \pi(B) = \pi(A), \quad n_n = \nu(B) = \nu(A), \quad n_z = \zeta(B) = \zeta(A).$$

逆に定理 D.2 を認めよう。実対称行列 A, 正則行列 P に対して、$B := P^TAP$ とおく。B は実対称行列であるから、適当な実直交行列 U で対角化できる: $\exists\lambda_1, \cdots, \lambda_n \in \mathbf{R}$ s.t. $U^TBU = \mathrm{diag}[\lambda_1, \cdots, \lambda_n]$. このとき

$$\begin{aligned}&\mathrm{diag}\,(\lambda_1, \cdots, \lambda_n) = U^TBU = U^TP^TAPU = (PU)^TA(PU) = P'^TAP',\\ &P' := PU.\end{aligned}$$

P と U が正則であるから、P' は正則である。定理 D.2 から、(B の固有値である) $\lambda_1, \cdots, \lambda_n$ のうちで正であるものの個数、負であるものの個数、0 であるものの個数は、それぞれ $\pi(A)$, $\nu(A)$, $\zeta(A)$ に等しい。ゆえに $\pi(B) = \pi(A)$, $\nu(B) = \nu(A)$, $\zeta(B) = \zeta(A)$. ■

D.3　Sylvester の慣性律の証明

それでは定理 D.1 を証明しよう。次に掲げるのは、線形代数のテキストで良く見掛けるものである。

定理 D.1 の証明　線型形式の独立性 (行列のランクというべき？) の議論をしている。

$x = Sy$, $x = Rz$ という正則変数変換で

$$\begin{aligned}(Ax, x) &= \alpha_1y_1^2 + \cdots + \alpha_ry_r^2 - \alpha_{r+1}y_{r+1}^2 - \cdots - \alpha_py_p^2 \quad (\alpha_j > 0)\\ &= \beta_1z_1^2 + \cdots + \beta_sz_s^2 - \beta_{s+1}z_{s+1}^2 - \cdots - \beta_qz_q^2 \quad (\beta_j > 0)\end{aligned}$$

と対角化できたとする。一般性を失うことなく $p \geq q$ として良い。目標は $p = q, r = s$ を証明することである。

各 y_j, z_k は変数 x についての線形形式と見なせる。y_j 同士、z_k 同士は 1 次独立である。

もしも $r < s$ ならば、$y_1, \cdots, y_p$ は $y_1, \cdots, y_r, z_{s+1}, \cdots, z_q$ の 1 次結合では表せない。(もし出来たとすると、独立な p 個の 1 次形式 $y_1, \cdots, y_p$ が、それより少ない $r + (q - s)$ 個の 1 次形式で表せることになって矛盾する。$r + (q - s) = (r - s) + q < q \leq p$ に注意)。例えば y_k, $r + 1 \leq k \leq p$ が $y_1, \cdots, y_r, z_{s+1}, \cdots, z_q$ で表せなかったとすると、

$$y_1 = \cdots = y_r = z_{s+1} = \cdots = z_q = 0, \quad y_k = 1$$

は $x_1, \cdots, x_n$ に関する連立 1 次方程式として解を持つ。しかし、

$$\begin{aligned}(Ax, x) &= \alpha_1 \cdot 0^2 + \cdots + \alpha_r \cdot 0^2 - \alpha_{r+1} y_{r+1}^2 - \cdots - \alpha_k \cdot 1^2 - \cdots \alpha_p y_p^2 \leq -\alpha_k < 0, \\ (Ax, x) &= \beta_1 z_1^2 + \cdots + \beta_s z_s^2 - \beta_{s+1} \cdot 0^2 - \cdots - \beta_q \cdot 0^2 \geq 0\end{aligned}$$

となって矛盾する。

同様に $r > s$ ならば、$y_1, \cdots, y_r$ が $y_{r+1}, \cdots, y_p, z_1, \cdots, z_s$ で表されない。例えば y_k $(1 \leq k \leq r)$ が $y_{r+1}, \cdots, y_p, z_1, \cdots, z_s$ で表されないとすると、

$$y_{r+1} = \cdots = y_p = z_1 = \cdots = z_s = 0, \quad y_k = 1$$

は $x_1, \cdots, x_n$ に関する連立 1 次方程式として解を持つ。しかし、

$$\begin{aligned}(Ax, x) &= \alpha_1 \cdot y_1^2 + \cdots + \alpha_k \cdot 1^2 + \cdots \alpha_r \cdot y_r^2 - \alpha_{r+1} \cdot 0^2 - \cdots - \alpha_p \cdot 0^2 \geq \alpha_k > 0, \\ (Ax, x) &= \beta_1 \cdot 0^2 + \cdots + \beta_s \cdot 0^2 - \beta_{s+1} \cdot y_{s+1}^2 - \cdots - \beta_q \cdot \beta_q^2 \leq 0\end{aligned}$$

となって矛盾する。

ゆえに $r = s$. 必要ならば $(-Ax, x)$ を考えることによって、負の係数を持つ項の個数が等しいことも示せる。■

この証明はやや分り辛い (少なくとも私にとって)。そこでこれとは独立に定理 D.2 の証明を与える (これは自前でででっち上げたものなので、後でもう一度チェックすること)。

定理 D.2 の証明　A を n 次実対称行列とする。

$$\begin{aligned}\mathcal{P} &:= \{V; V \text{ は } \mathbf{R}^n \text{ の部分空間}, \forall x \in V \setminus \{0\} \quad (Ax, x) > 0\}, \\ \mathcal{N} &:= \{V; V \text{ は } \mathbf{R}^n \text{ の部分空間}, \forall x \in V \setminus \{0\} \quad (Ax, x) < 0\}, \\ \mathcal{Z} &:= \{V; V \text{ は } \mathbf{R}^n \text{ の部分空間}, \forall x \in V \setminus \{0\} \quad (Ax, x) = 0\}\end{aligned}$$

とおく。$\{0\} \in \mathcal{P}, \mathcal{N}, \mathcal{Z}$ であるから、$\mathcal{P}, \mathcal{N}, \mathcal{Z} \neq \emptyset$.

$$N_p := \max\{\dim V; V \in \mathcal{P}\}, \quad N_n := \max\{\dim V; V \in \mathcal{N}\}, \quad N_z := \max\{\dim V; V \in \mathcal{Z}\}$$

が定まる (念のため: $\dim\{0\} = 0$ である)。

主張: $N_p = \pi(A)$, $N_n = \nu(A)$, $N_z = \zeta(A)$.

A は実対称行列であるから、固有ベクトルからなる正規直交基底 $v_1, \cdots, v_n$ が存在する。$Av_j = \lambda_j v_j$ として、

$$V_p := \mathrm{Span}\{v_j; \lambda_j > 0\}, \quad V_n := \mathrm{Span}\{v_j; \lambda_j < 0\}, \quad V_z := \mathrm{Span}\{v_j; \lambda_j = 0\}$$

とおくと、$V_p \in \mathcal{P}$, $V_n \in \mathcal{N}$, $V_z \in \mathcal{Z}$. $\dim V_p = \pi(A)$, $\dim V_n = \nu(A)$, $\dim V_z = \zeta(A)$ であるから、N_p, N_n, N_z の最大性によって

$$(\sharp) \qquad N_p \geq \pi(A), \quad N_n \geq \nu(A), \quad N_z \geq \zeta(A).$$

さて、一般に

$$W_p \in \mathcal{P}, \quad W_n \in \mathcal{N}, \quad W_z \in \mathcal{Z} \quad \Rightarrow \quad W_p \cap W_n = W_N \cap W_z = W_z \cap W_p = \{0\}$$

が成り立つことは容易に証明できる。W_p, W_n, W_z として、特に

$$\dim W_p = N_p, \quad \dim W_n = N_n, \quad \dim W_z = N_z$$

を満たすものを取って、

$$W_p \oplus W_n \oplus W_z \subset \mathbf{R}^n$$

において、次元を調べる。$(\sharp)$ を用いると

$$n = \pi(A) + \nu(A) + \zeta(A) \leq N_p + N_n + N_z \leq n.$$

左辺と右辺が一致するので、不等式はすべて等式で

$$N_p = \pi(A), \quad N_n = \nu(A), \quad N_z = \zeta(A) \quad (\text{主張の証明終わり}).$$

さて、正則行列 P による変換 $x = Py$ で、

$$(Ax, x) = \sum_{j=1}^{n} \mu_j y_j^2$$

になったとする。簡単のために順番を修正して、最初の p 個は正，次の q 個は負、残りは 0, と出来る。つまり

$$(Ax, x) = \alpha_1 y_1^2 + \cdots + \alpha_p y_p^2 - \alpha_{p+1} y_{p+1}^2 - \cdots - \alpha_{p+q} y_{p+q}^2, \quad \alpha_j > 0 \quad (1 \leq j \leq p+q)$$

として良い。

(a) p 次元空間 $\{Py; y_{p+1} = \cdots = y_n = 0\}$ では $(Ax, x) > 0$ になるので、$p \leq \pi(A)$.

(b) q 次元空間 $\{Py; y_1 = \cdots = y_p = y_{p+q+1} = \cdots = y_n = 0\}$ では $(Ax, x) < 0$ になるので、$q \leq \nu(A)$.

(c) $n-(p+q)$ 次元空間 $\{Sy; y_1 = \cdots = y_{p+q} = 0\}$ では $(Ax, x) = 0$ になるので、$n-(p+q) \leq \zeta(A)$.

(c) から $p+q+\zeta(A) \geq n$ であるが、(a) から $\pi(A) \geq p$, (b) から $\nu(A) \geq q$ であるから、

$$n = \pi(A) + \nu(A) + \zeta(A) \geq p + q + \zeta(A) \geq n.$$

両端が等しいので、途中はすべて等式で $p = \pi(A)$, $q = \nu(A)$. ■

参考文献

[1] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002.

[2] 伊理正夫. 一般線形代数. 岩波書店, 2003. 伊理正夫, 線形代数 I, II, 岩波講座応用数学, 岩波書店 (1993, 1994) の単行本化.

[3] 伊理正夫. 線形代数汎論. 朝倉書店, 2009. 「一般線形代数」のリニューアル.

[4] 桂田祐史. 多変数の微分積分学 1 講義ノート (2013 年度版). http://www.math.meiji.ac.jp/~mk/lecture/tahensuu1-2013/tahensuu1-new-text.pdf, 2013.

[5] 森正武. 数値解析. 共立出版, 1973. 第 2 版が 2002 年に出版された。

[6] 森正武. 数値解析法. 朝倉現代物理学講座 7. 朝倉書店, 1984.

[7] 森正武, 杉原正顯, 室田一雄. 線形計算. 岩波講座 応用数学. 岩波書店, 1994.

[8] 杉原正顯, 室田一雄. 線形計算の数理. 岩波書店, 2009.

[9] Lloyd Nicholas Trefethen and David Bau III. *Numerical Linear Algebra*. SIAM, 1997.