エルミート行列の固有値分解アルゴリズム

Kosugitti の研究ノート

2008/02/06

1　原理

線形代数では、$\boldsymbol{Q}\bar{\boldsymbol{Q}} = \boldsymbol{I}$ すなわち、$\boldsymbol{Q}^{-1} = \bar{\boldsymbol{Q}}$ という性質を持つ行列 $\boldsymbol{Q}$ をユニタリ行列 (unitary matrix) という。また、右上三角行列というのは、非ゼロ要素が体格の右上半分だけにある行列、すなわち

$$\begin{pmatrix} * & * & * & \cdots & * & * \\ 0 & * & * & \cdots & * & * \\ 0 & 0 & * & \cdots & * & * \\ 0 & 0 & 0 & \cdots & * & * \\ & & \cdots & & & \\ 0 & 0 & 0 & \ldots & * & * \\ 0 & 0 & 0 & \ldots & 0 & * \end{pmatrix} \tag{1}$$

という形の行列である (*には任意の数値が入る)。

エルミート行列の固有値分解は、**QR** 法と呼ばれる算術アルゴリズムによって達成される。**QR 法 (QR algorithm)** というのは、正則な行列 $\boldsymbol{A}$ が与えられたとき、

1. $\boldsymbol{A}$ をユニタリ行列 $\boldsymbol{Q}$ と上三角行列 $\boldsymbol{R}$ の積に分解する。すなわち、$\boldsymbol{A}^{(t)} = \boldsymbol{QR}$
2. その結果を逆順に掛ける。$\boldsymbol{A}^{(t+1)} = \boldsymbol{RQ}$

という 2 つの操作を反復して、固有値を求める。これらの計算は、固有値を変化させずに行列の形を変形するアルゴリズムであり、このような反復を繰り返すと、非対角要素の絶対値は次第に小さくなり、t が十分に大きくなれば $\boldsymbol{A}^{(t)}$ は対角行列に収束し、その対角項が固有値となる。

$\boldsymbol{A}$ を $\boldsymbol{Q}$ と $\boldsymbol{R}$ との積に分解する方法はいくつかあるが、ここではグラム・シュミット (Gram-Schmidt) の直交化法を紹介する。

この方法では、行列 $\boldsymbol{A}$ の第一列 $\boldsymbol{a}_1$、第二列 $\boldsymbol{a}_2$、$\cdots$、第 n 列 $\boldsymbol{a}_n$ を元に、行列 $\boldsymbol{Q}$ の第一列 $\boldsymbol{q}_1$、第二列 $\boldsymbol{q}_2$、$\cdots$、第 n 列 $\boldsymbol{q}_n$ を作る。このとき $\boldsymbol{Q}$ がユニタリ行列になるようにするためには、

- $\boldsymbol{q}_i$ と $\boldsymbol{q}_j$ が互いに直交する。すなわち、$(\boldsymbol{q}_i, \boldsymbol{q}_j) = 0.$ ただし $i \neq j$
- 各 $\boldsymbol{q}_i$ の長さは 1 である。すなわち、$(\boldsymbol{q}_i, \boldsymbol{q}_i) = 1$

の条件を満たしていればよい。今回は複素行列を分解するので、内積 $(\boldsymbol{x}, \boldsymbol{y})$ は

$$(\boldsymbol{x}, \boldsymbol{y}) = x_1\bar{y}_1 + x_2\bar{y}_2 + \cdots + x_n\bar{y}_n$$

であることに注意が必要である。

そこで、

$$
\begin{array}{rcl}
\boldsymbol{b}_1 & = & \boldsymbol{a}_1 \\
\boldsymbol{b}_2 & = & (\boldsymbol{a}_2\text{から } \boldsymbol{a}_1\text{に平行な成分を抜き取ったもの}) \\
\boldsymbol{b}_3 & = & (\boldsymbol{a}_3\text{から } \boldsymbol{a}_1, \boldsymbol{a}_2\text{に平行な成分を抜き取ったもの}) \\
 & \cdots & \\
\boldsymbol{b}_n & = & (\boldsymbol{a}_n\text{から } \boldsymbol{a}_1, \cdots \boldsymbol{a}_{n-1}\text{に平行な成分を抜き取ったもの})
\end{array}
\tag{2}
$$

という要領で、$\boldsymbol{b}_1, \boldsymbol{b}_2 \cdots \boldsymbol{b}_n$ を作り、各 $\boldsymbol{b}_i$ をその長さ $|\boldsymbol{b}_i|$ で割って $\boldsymbol{q}_i$ にするとよい。

より具体的には、

1. $\boldsymbol{b}_1 = \boldsymbol{a}_1$
2. $r_{11} = |\boldsymbol{b}_1|$
3. $\boldsymbol{q}_1 = 1/r_{11}\boldsymbol{b}_1$
4. $k = 2, 3, \cdots, n$ の順に
   (a) $r_{jk} = (\boldsymbol{a}_k, \boldsymbol{q}_j)(j = 1, 2, \cdots, k-1)$
   (b) $\boldsymbol{b}_k = \boldsymbol{a}_k - r_{1,k}\boldsymbol{q}_1 - r_{2,k}\boldsymbol{q}_2 - \cdots - r_{k-1,k}\boldsymbol{q}_{k-1}$
   (c) $r_{kk} = |\boldsymbol{b}_k|$
   (d) $\boldsymbol{q}_k = 1/r_{kk}\boldsymbol{b}_k$

とし、こうして作られた $\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_n$ を列ベクトルとする行列が $\boldsymbol{Q}$ に、r_{ij} を要素とする右上三角行列が $\boldsymbol{R}$ になる。

参考:戸川隼人著 (1992)『UNIX ワークステーションによる科学技術計算ハンドブック』新装版 サイエンス社.

2 プログラムについて

QR 法のプログラムを以下に示す。ここで、配列 H[N][N] は $N \times N$ のエルミート行列であり、固有値が複素ベクトル $Eigenval[N]$ に入る。なお、このプログラムはフリーの C コンパイラである、GNU C Compiler[*1] に適するように書かれたものである。他のコンパイラを用いる場合、複素数型の宣言や演算について変更が必要かもしれない。

例えば、このコンパイラでは複素数型は

```
__complex__ int A;
__complex__ double B;
```

といった形で宣言する。複素数 $a + bi$ の実数部分 (a) は、

```
"__real__ A"
```

複素数部分 (b) は、

```
"__imag__ A"
```

[*1] gcc にはいくつかの種類があるが、筆者は Mingw32 を用いている。

とすれば参照できる。また、共役複素数 $(a - bi)$ は

"~A"(チルダ A)

で表現する。

2.1 プログラム

```
void GS_QR(__complex__ double *pt,__complex__ double *pt2,int N, int *pt3){
    __complex__ double H[N][N];
    __complex__ double B[N];
    __complex__ double Q[N][N];
    __complex__ double R[N][N];
    __complex__ double r,temp;
    __complex__ double Eigenval[N];
    double eps = 1e-100;
    int i,j,k,l;
    int FLG = 0;
    int count = 1;

    while( FLG == 0 ){

        count++;
        if( count == 100000 ){ FLG = 1; }

        //変数の初期化
        for( i = 0 ; i < N ; i++ ){
            for( j = 0 ; j < N ; j++ ){
                Q[i][j] = 0.0;
                R[i][j] = 0.0;
            }
            B[i] = 0.0;
        }


        /*Gram-Schmidt の正規直交化法*/
        for( k = 0 ; k < N ; k++){
            for( i = 0 ; i < N ; i++){
                B[i] = H[i][k];
            }
            for( j = 0 ; j < k ; j++){
                r = 0.0;
                for( i = 0 ; i < N ; i++ ){
                    r = r + ( B[i] * ~Q[i][j]);
                }
                for( i = 0 ; i < N ; i++ ){
                    B[i] = B[i] - ( r * Q[i][j] );
                }
                R[j][k] = r;
            }
            r = 0.0;
            for( i = 0 ; i < N ; i++ ){
                r = r + ( B[i] * ~B[i] );
            }
            r = sqrt(r);
            for( i = 0 ; i < N ; i++ ){
                Q[i][k] = B[i] / r;
            }
            R[k][k] = r;
        }
```

```
        //新 H = RQ
        for( i = 0 ; i < N ; i++ ){
            for( j = 0 ; j < N ; j++ ){
                r = 0.0;
                for( k = 0 ; k < N ; k++ ){
                    r = r + ( R[i][k] * Q[k][j] );
                }
                H[i][j] = r;
            }
        }

        //フラグ
        r = 0.0;
        for( i = 0 ; i < N ; i++ ){
            for( j = 0 ; j < i ; j++ ){
                r = r + H[i][j];
            }
        }

        if( fabs(__real__ r) < eps && fabs(__imag__ r) < eps ){
            for( k = 0 ; k < N ; k++ ){
                Eigenval[k] = H[k][k];
            }
            //サイズの順に並べ替え
            for( i = 0 ; i < N ; i++ ){
                for( j = i ; j < N ; j++ ){
                    if( __real__ Eigenval[i] < __real__ Eigenval[j] ){
                        temp = Eigenval[i];
                        Eigenval[i] = Eigenval[j];
                        Eigenval[j] = temp;
                    }
                }
            }
            FLG = 1;
        }
    }

}
```