置換の回数を求める算法

01ca0125 鈴木 藍
2002 年 5 月 14 日

目 次

概 要

行列式で使う 符号を表す関数 $sign$ を C 言語を使って実装した。また、実際にテストデータを 2 つ使って動作を確認した。

レポートの目的

欠席した授業内容に関する調査を行い、行列式の理解を深める。また、置換と互換の関係について調べた。

1 置換と互換について

欠席中に行われた講義の内容を調査した。

1.1 置換

ある集合 M_1 から、ある集合 M_2 の 1 対 1 の写像を置換という。また、M_1 の集合の要素の数を n とする。M_1 の置換をすべて集めた集合を $S(n)$ と書くとすると、この $S(n)$ は順列と同一視する事が出来るので、$S(n)$ の要素の個数は $n!$ 個になる。

1.2 行列式

例として、3 次の正方行列 A に対して、その行列式 $detA$, または $|A|$ を

$$detA = |A| = \sum_{\sigma \in S(3)} sign\sigma a_{1\sigma(1)} a_{2\sigma(2)} a_{3\sigma(3)}$$

で定義する。これは、行列の第 1 行から 第 3 行まで各行から一つずつ成分を取り出して掛け合わせている。ただし、これらの成分を取り出す時に、同じ列から取り出さないようにするという制約がある。これは、1 対 1 の写像で表すことが出来る。
そして、このような 3 個の要素の取り出し方は全部で 3! 通りあるので、和は 3! 個の和となる。また、それぞれの 3 個の要素の積には $sign\sigma$ となる係数がかけてあり、これは σ の性質により、$+1$ か -1 の値をとる。 $+1$ か -1 のどちらの値を取るかは、σ が偶置換であるか奇置換であるかによって決まる。

1.3 置換の集合 $S(n)$

例として、$1, 2, 3$ のすべての置換を 外延的に示すと

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$$

の 6 つになる。このうち、$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$ は、実際には置き換えがされていないが、置換の一つと考え、これを恒等置換という。

1.4 互換

置換のうちで、$1, 2, 3$ の ふたつの要素のみが互いに対応しあい、他の要素が不変であるものを互換と言う。互換は、互いに対応しあう要素、i j のとき、それを括弧で括って (i, j) とあらわす。どのような置換も、互換の合成されたもの (積) として表すことが出来る。特に, 偶数個の互換の合成で表される置換を偶置換、奇数個の互換の合成で表される置換を奇置換という。

1.5 符号 $sign\sigma$

ある置換 σ に対し、その符号 $sign\sigma$ を

$$sign\sigma = \begin{cases} 1 & \sigma\text{が偶置換} \\ -1 & \sigma\text{が奇置換} \end{cases}$$

と定義する。一般的に $sign(\tau\sigma) = (sign\tau)(sign\sigma)$ が成り立つ。

2 $sign(\sigma)$ の計算

2.1 課題の内容

ある置換 σ を与えた時、 $sign(\sigma)$ を計算する方法、またプログラムを作る。もととなるものは恒等置換とする。また、プログラムのテストは 5 文字からなる 置換で実行する。

2.2 方針

置換を、2 次元配列で表現し、配列の各要素の添字を j とする。恒等置換とは、2 次元配列 1 行目と 2 行目の すべての要素が同じ並びで格納されている状態とする。1 行目の j 番目の要素 と 2 行目の j 番目の要素を比べ、異なる要素であるなら、1 行目の j 番目と同じ要素を 2 行目から捜し出す。見付かったら その要素と入れ換え、これを配列の長さ分繰り返す。入れ換えが行われた回数を数え、偶数回入れ換えを行ったら 1 を、奇数回入れ換えを行ったら -1 を返し、終了する。アルゴリズムの時間に習った 線形探索を使って求める。

2.3 作業環境

- OS : VineLinux 2.1.5
- 使用言語 : C 言語
- 処理系 : GNU C compiler

2.4 計算方法

$sign(\sigma)$ を求める関数の流れ図は以下の通りである。

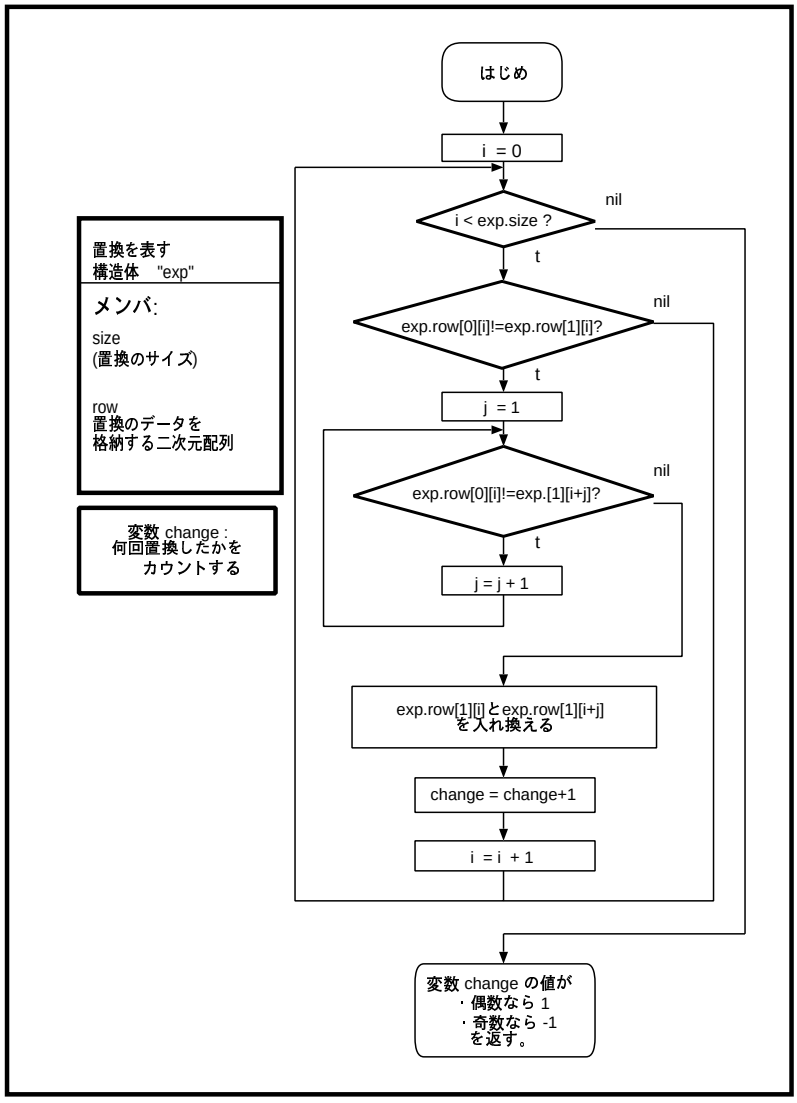

図 1: $sign(\sigma)$ を求める流れ図

3 関数の実装

3.1 ヘッダファイル

ヘッダファイルに置換を表す構造体を定義した。

```
#define  PERMUTATION_SIZE  64
#define  ROW                2
#define  UPPER              0
#define  LOWER              1
#define  EVEN               1
#define  ODD               -1
typedef struct expresson Permutation;
struct expresson {
   int  expressonSize;
   int  row[ROW][PERMUTATION_SIZE];
};
int           sign(Permutation *exp);
Permutation  *swap(Permutation *exp, int from, int to);
Permutation  *init(Permutation *exp, int size);
```

3.2 *sign* 関数

sign 関数の実装は以下の通りである。置換を格納した構造体を受け取り、入れ換えが偶数回行われれば 1 を、奇数回行われれば -1 を返す。また、配列の要素の入れ換えは swap 関数が行う。

```
int sign(Permutation *exp)
{
   int i, j, change;
   change = 0;
   for (i=0; i<exp->expressonSize; i++) {
     if (exp->row[UPPER][i]
         != exp->row[LOWER][i]) {
      for (j=1; exp->row[UPPER][i]
           != exp->row[LOWER][i+j]; j++)
        if(j>exp->expressonSize) exit(NULL);
      exp     = swap(exp, i, (i+j));
      change += 1;
      } /* for 'j' */
   }/* for 'i' */
   return  (change%2) ? EVEN : ODD;
}
```

3.3 その他の関数

それぞれの関数には、関数の動作を確認するために 置換の要素の値を表示する記述がある。これは、ヘッダファイルに定義されている

```
#define TRACE
```

をコメントアウトすることで表示しないようにすることが出来る。

3.3.1 *init* 関数

置換を表す配列に、実際に値を代入する。

3.3.2 *swap* 関数

配列の要素の入れ換えを行う。

4 実行結果

4.1 main 関数の概要

main 関数では、以下のように *sign* 関数を呼び出している。sibma は 置換を表す構造体、answer は、*sign* 関数の返却値を格納する int 型の変数である。

```
answer = sign(sigma);
```

また、返却値を視覚的に確認するために 返却値に応じたメッセージを表示するようにした。

4.2 テストデータ

テストデータは、授業内容にあったものと 5 行からなる置換を用意した。また、値の格納は、関数 *init* に記述されている。

$$testA = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 4 & 2 & 3 & 5 \end{pmatrix}$$

A 置換は、偶置換である。

$$testB = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 1 & 5 \end{pmatrix}$$

4.3 A の実行結果

```
[spiral: ~/simulation/permutation/src]$ make
gcc -Wall -o Permutation -c sign.c
gcc -Wall main.c Permutation && \
  a.out
initing for calculation:
permutation 1th atom is 1
permutation 2th atom is 4
permutation 3th atom is 2
permutation 4th atom is 3
permutation 5th atom is 5
and permutations size is 5

calculation is starting...
>
change!
permutation 2th atom to 3th.
now, permutation atoms are
1th is 1
2th is 2
3th is 4
4th is 3
5th is 5
change!
permutation 3th atom to 4th.
now, permutation atoms are
1th is 1
2th is 2
3th is 3
4th is 4
5th is 5
permutation changes ODD  times to constant permutation.
```

テストデータ A の実行結果は、偶置換である。

4.4 B の実行結果

```
[spiral: ~/simulation/permutation/src]$ make
gcc -Wall -o Permutation -c sign.c
gcc -Wall main.c Permutation && \
  a.out
initing for calculation:
permutation 1th atom is 3
permutation 2th atom is 4
permutation 3th atom is 2
permutation 4th atom is 1
permutation 5th atom is 5
and permutations size is 5

calculation is starting...
>
change!
permutation 1th atom to 4th.
now, permutation atoms are
1th is 1
2th is 4
3th is 2
4th is 3
5th is 5
change!
permutation 2th atom to 3th.
now, permutation atoms are
1th is 1
2th is 2
3th is 4
4th is 3
5th is 5
change!
permutation 3th atom to 4th.
now, permutation atoms are
1th is 1
2th is 2
3th is 3
4th is 4
5th is 5
permutation changes EVEN times to constant permutation.
```

テストデータ B の実行結果は、奇置換である。

4.4.1 テストデータ B の確認

実際に互換で確認した。互換は (i, j) で表す。
1 回目の入れ換え

$$(1,4)\begin{pmatrix}1&2&3&4&5\\3&4&2&1&5\end{pmatrix} = \begin{pmatrix}1&2&3&4&5\\1&4&2&3&5\end{pmatrix}$$

2 回目の入れ換え

$$(2,3)\begin{pmatrix}1&2&3&4&5\\1&4&2&3&5\end{pmatrix} = \begin{pmatrix}1&2&3&4&5\\1&2&4&3&5\end{pmatrix}$$

3 回目の入れ換え

$$(3,4)\begin{pmatrix}1&2&3&4&5\\1&2&4&3&5\end{pmatrix} = \begin{pmatrix}1&2&3&4&5\\1&2&3&4&5\end{pmatrix}$$

よって、B は 奇数個の置換で表される。積の形で表すと

$$\begin{pmatrix}1&2&3&4&5\\3&4&2&1&5\end{pmatrix} = (1,4)(2,3)(3,4)\begin{pmatrix}1&2&3&4&5\\1&2&3&4&5\end{pmatrix}$$

となる。

このプログラムは、以下の URL からダウンロードが出来る。
http://www.soulhack.net/TeX/permutation/

まとめ・結論

行列式の性質は 1 年次に習ったものと、それ以外の物があった。$sign$ はその名の通り符号であり、行列式の計算をおこなったときになぜ符号が入れ替わり $+$ と $-$ で変わっているのかがわかった。

感想

まだ、これらの行列に関するプログラムをいつ使うのか検討が付かないが、早く何かに活用できればと思う。

参考文献

[1] 線形台数 : 行列式の定義
http://www.fbc.keio.ac.jp/ hkomiya/education/senkei_daisu_2001-0.2.pdf
2002 年 5 月 13 日参照