積の規則の確認

01ca0125 鈴木 藍
2001 年 5 月 13 日

目 次

概 要

行列の和、差、実数倍、積、転置行列 を求める関数を C 言語で作成し、行列の積の規則を確認する。実際に行列のデータを用意し規則をプログラムで実行し、確認した。

レポートの目的

行列の演算を行う関数を久しぶりの C 言語で実装する。また、行列の演算を 配列で行う練習をする。

1 行列の規則

1.1 積の規則

作成した関数で以下の規則を確認する。

- $(AB)C = A(BC)$
- $AE = A \quad EA = A$ となるような E の存在
- $A(B + C) = AB + BC$
- $(A + B)C = AC + BC$

2 作成した関数

2.1 作成の環境

- OS : VineLinux 2.1.5
- コンパイラ : gcc

2.2 データの扱い

行列を表す構造体 Matrix を定義する。メンバとして、行列のデータを格納する 2 次元配列、行の数と列の数を行列の情報として持つ。

2.3 関数名とその内容

2.3.1 matplus

行列の和を求める関数。引数に、演算される行列 A, 演算する行列 B, 結果を格納する行列 ans をとり、演算結果 ans を返す。

```
Matrix *matplus(Matrix *A, Matrix *B, Matrix *ans)
{
   int i, j;

    for (i=0; i<A->rowsize; i++)
      for (j=0; j<A->columnsize; j++)
        ans->row[i][j] = A->row[i][j] + B->row[i][j];

    ans->rowsize    = A->rowsize;
    ans->columnsize = A->columnsize;
    return ans;
}
```

2.3.2 mattake

行列の差を求める関数。引数に、演算される行列 A, 演算する行列 B, 結果を格納する行列 ans をとり、演算結果 ans を返す。

```
Matrix *mattake(Matrix *A, Matrix *B, Matrix *ans)
{
   int i, j;

   for (i=0; i<A->rowsize; i++)
     for (j=0; j<A->columnsize; j++)
       ans->row[i][j] = A->row[i][j] - B->row[i][j];

   ans->rowsize    = A->rowsize;
   ans->columnsize = A->columnsize;
   return ans;
}
```

2.3.3 matmul

行列の実数倍を求める。引数に演算される行列 mat, 結果を格納する行列 ans, 演算する実数 k をとり、演算結果 ans を返す。

```
Matrix *matmul(Matrix *mat, Matrix *ans, int k)
{
   int i, j;

   for (i=0; i<mat->rowsize; i++)
     for (j=0; j<mat->columnsize; j++)
       ans->row[i][j] = mat->row[i][j] * 5;

   ans->rowsize    = mat->rowsize;
   ans->columnsize = mat->columnsize;
   return ans;
}
```

2.3.4 product

行列の積を求める関数。引数に演算される行列 A, 演算する行列 B, 結果を格納する行列 ans をとり、演算結果 ans を返す。

```
Matrix *product(Matrix *A, Matrix *B, Matrix *ans)
{
   Matrix  buffer;
   Matrix *buf;
   int     i, j;
   buf = &buffer;

   if (A->columnsize != B->rowsize)
     return NIL;

   buf = oppmove(B, buf);

   for (i=0; i<A->rowsize; i++)
     for (j=0; j<B->columnsize; j++)
       ans->row[i][j] = naiseki(A->row[i], buf->row[j], A->columnsize);

   ans->rowsize    = A->rowsize;
   ans->columnsize = B->columnsize;
   return ans;
}
```

2.3.5 oppmove

転置行列を求める関数。転置する行列 mat, 結果を格納する行列 ans を引数にとり演算結果 ans を返す。

```
Matrix *oppmove(Matrix *mat, Matrix *ans)
{
  int i, j, k, l;

  for (i=0, l=0; i<mat->rowsize; i++, l++)
    for (j=0, k=0; j<mat->columnsize; j++, k++)
      ans->row[k][l] = mat->row[i][j];

  ans->rowsize    = mat->columnsize;
  ans->columnsize = mat->rowsize;
  return ans;
}
```

以上のモジュールは、Matrix というモジュールライブラリにまとめ使えるようにした。

2.4 ヘッダファイル

ヘッダファイル matrix.h の内容は以下の通りである。

```
#define MATRIX_SIZE  60
#define  T    1
#define  NIL  0

typedef struct mat Matrix;
struct mat {
  int  rowsize, columnsize;
  int row[MATRIX_SIZE][MATRIX_SIZE];
};

int naiseki(int *VA, int *VB, int size);
int car(int V[]);
int *cdr(int V[]);
Matrix *matmul(Matrix *mat, Matrix *ans, int k);
Matrix *matplus(Matrix *A, Matrix *B, Matrix *ans);
Matrix *mattake(Matrix *A, Matrix *B, Matrix *ans);
Matrix *oppmove(Matrix *mat, Matrix *ans);
Matrix *product(Matrix *A, Matrix *B, Matrix *ans);
```

3 積の規則の確認

main 関数内でデータを格納された状態の行列を必要な数だけ 記述し、それを用いて確認した。

3.1 $(AB)C = A(BC)$

ファイル trans.c に main 関数を記述した。用意したデータは以下の通りである。

```
Matrix  matrixA = {4, 4, {{ 1, 2, 3, 4},
                          { 5, 6, 7, 8},
                          { 9,10,11,12},
                          {13,14,15,16}}};

Matrix  matrixB = {4, 4, {{ 1, 2, 3, 4},
                          { 5, 6, 7, 8},
                          { 9,10,11,12},
                          {13,14,15,16}}};

Matrix  matrixC = {4, 4, {{ 1, 2, 3, 4},
                          { 5, 6, 7, 8},
                          { 9,10,11,12},
                          {13,14,15,16}}};
```

また、一時的に演算結果を格納する行列 buf を使った。演算結果は $(AB)C$ を ans1 に、$A(BC)$ を ans2 に格納し、表示する。

3.1.1 $(AB)C = A(BC)$ 演算

$(AB)C$

```
buf  = product( A, B, buf);
ans1 = product(buf, C, ans1);
```

$A(BC)$

```
buf  = product(  B, C, buf);
ans2 = product(buf, A, ans2);
```

表示部は trans.c の 44 行目以降にある。

3.1.2 $(AB)C = A(BC)$ の演算結果

```
[spiral: ~/simulation/matrix/src]$ make abc
gcc -Wall trans.c Matrix && \
  a.out
  (AB)C =
    3140  3560  3980  4400
    7268  8232  9196 10160
   11396 12904 14412 15920
   15524 17576 19628 21680
  A(BC) =
    3140  3560  3980  4400
    7268  8232  9196 10160
   11396 12904 14412 15920
   15524 17576 19628 21680
```

3.2 $AE = A \quad EA = A$

ファイル root.c に main 関数を記述した。用意したデータは以下の通りである。

```
Matrix  matrixA = {4, 4, {{1, 2, 3, 4},
                          {5, 6, 7, 8},
                          {9,10,11,12},
                          {13,14,15,16} }};
/* 単位行列 */
Matrix  matrixE = {4, 4, {{1, 0, 0, 0},
                          {0, 1, 0, 0},
                          {0, 0, 1, 0},
                          {0, 0, 0, 1} }};
```

一時的に演算結果を格納する行列 buf を使った。演算結果は AE を ans1 に、EA を ans2 に格納し、表示する。

3.2.1 $AE = A \quad EA = A$ の演算

AE

```
ans1 = product(A, E, ans1);
```

EA

```
ans2 = product(E, A, ans2);
```

表示部は root.c の 36 行目以降にある。

3.2.2 $AE = A$　$EA = A$ の演算結果

```
[spiral: ~/simulation/matrix/src]$ make ae
gcc -Wall root.c Matrix && \
  a.out
  A*E =
       1     2     3     4
       5     6     7     8
       9    10    11    12
      13    14    15    16
  E*A =
       1     2     3     4
       5     6     7     8
       9    10    11    12
      13    14    15    16
```

3.3 $A(B + C) = AB + AC$

ファイル divide1.c に main 関数を記述した。用意したデータは以下の通りである。

```
  Matrix  matrixA = {4, 4, {{ 1, 2, 3, 4},
                            { 5, 6, 7, 8},
                            { 9,10,11,12},
                            {13,14,15,16}}};

  Matrix  matrixB = {4, 4, {{ 1, 2, 3, 4},
                            { 5, 6, 7, 8},
                            { 9,10,11,12},
                            {13,14,15,16}}};

  Matrix  matrixC = {4, 4, {{ 1, 2, 3, 4},
                            { 5, 6, 7, 8},
                            { 9,10,11,12},
                            {13,14,15,16}}};
```

一時的に演算結果を格納する行列 buf1, buf2, buf3 を使った。演算結果は $A(B + C)$ を ans1 に、$AB + AC$ を ans2 に格納し、表示する。

3.3.1 $A(B+C) = AB + AC$ の演算

$A(B+C)$

```
buf1 = matplus( B, C, buf1);
ans1 = product(A, buf1, ans1);
```

$AB + AC$

```
buf2 = product(A, B, buf2);
buf3 = product(A, C, buf3);

ans2 = matplus(buf2, buf3, ans2);
```

表示部は divide1.c の 48 行目以降にある。

3.3.2 $A(B+C) = AB + AC$ の演算結果

```
[spiral: ~/simulation/matrix/src]$ make ab+c
gcc -Wall divide1.c Matrix && \
  a.out
A(B+C) =
   180   200   220   240
   404   456   508   560
   628   712   796   880
   852   968  1084  1200
AB+AC =
   180   200   220   240
   404   456   508   560
   628   712   796   880
   852   968  1084  1200
```

3.4 $(A+B)C = AC+BC$

ファイル divide2.c に main 関数を記述した。用意したデータは以下の通りである。

```
Matrix  matrixA = {4, 4, {{ 1, 2, 3, 4},
                          { 5, 6, 7, 8},
                          { 9,10,11,12},
                          {13,14,15,16}}};

Matrix  matrixB = {4, 4, {{ 1, 2, 3, 4},
                          { 5, 6, 7, 8},
                          { 9,10,11,12},
                          {13,14,15,16}}};

Matrix  matrixC = {4, 4, {{ 1, 2, 3, 4},
                          { 5, 6, 7, 8},
                          { 9,10,11,12},
                          {13,14,15,16}}};
```

一時的に演算結果を格納する行列 buf1, buf2, buf3 を使った。演算結果は $(A+B)C$ を ans1 に、$AB+AC$ を ans2 に格納し、表示する。

3.4.1 $(A+B)C = AC+BC$ の演算

$(A+B)C$

```
buf1 = matplus( A, B, buf1);
ans1 = product(buf1, C, ans1);
```

$AC+BC$

```
buf2 = product(A, C, buf2);
buf3 = product(B, C, buf3);

ans2 = matplus(buf2, buf3, ans2);
```

表示部は divide1.c の 48 行目以降にある。

3.4.2 $(A+B)C = AC + BC$ の演算結果

```
[spiral: ~/simulation/matrix/src]$ make a+bc
gcc -Wall divide2.c Matrix && \
  a.out
(A+B)C =
   180   200   220   240
   404   456   508   560
   628   712   796   880
   852   968  1084  1200
AC+BC =
   180   200   220   240
   404   456   508   560
   628   712   796   880
   852   968  1084  1200
```

まとめ・結論

行列の規則は、1 年次の基礎数学でならい、計算をしたがプログラムでの演算結果も、期待通りのものになった。手で計算するよりも計算機で計算した方が、行列の規則を実感しやすいと感じた。行列を 2 次元配列で表現したが、転置などの演算を行う場合、操作する添字に規則性があった。

感想

C 言語を久く書かなかったので非常に手間取り、 プログラムが書けない事を実感した。もう、Lisp はプロトタイピングで使っていこうと思った。その他の言語も Smalltalk 以外はあまり触らないようにしようと思った。

参考文献

[1] 線形台数
http://www4.justnet.ne.jp/ masema/content.html
2001/ 5/11 参照

[2] 入門 情報処理数学
野ノ山 隆幸
2000 年 4 月 20 日 第 9 版