C 言語と行列

桂田 祐史

平成 25 年 11 月 6 日

1 はじめに

微分方程式の数値シミュレーションのプログラムには、大きな行列 (や二重添字を持つ数列[1]) が現れます。Fortran では、行列を表すのに、2 次元配列を用いるのが普通です。しかし C 言語には整合配列の機能がないので、2 次元配列を用いると、ポータビリティーのある関数を作るのが難しくなります。そこで、次のような工夫をします。

1 次元配列の方法 1 次元配列、あるいはポインターを用いて領域を確保し、二重添字から自前で 1 次元的な番地を計算して、アクセスする

ポインター配列の方法 ポインター配列、あるいはポインターのポインターを用いて確保した領域を 2 次元配列的な記法でアクセスする

2 1 次元配列の方法

例えば $m \times n$ 型の行列 A を表現して、計算に用いるために

領域の確保

```
double a[m * n];
```

のように 1 次元配列 `a[]` を宣言するか、あるいは (こちらの方がずっと C らしいが)

[1] 例えば、長方形領域を格子に切って、格子点上での関数値を扱う場合など。

```
double *a;
```

とポインターを宣言しておいて、

```
if ((a = (double *)malloc(sizeof(double) * m * n)) == NULL) {
    エラーの場合の処理
}
```

のように動的に記憶領域を確保する。

成分へのアクセス　(i,j) 成分 $(0 \leq i \leq m-1, 0 \leq j \leq n-1)$ をアクセスする時は面倒だが

```
a[n * i + j]
```

のように書くか、どこかで

```
#define A(i,j) a[n*(i)+(j)]
```

のようなマクロを定義しておいて

```
A(i,j)
```

と書く。

3　ポインター配列の方法

例えば $m \times n$ 型の行列 A を表現して、計算に用いるために

領域の確保 (準備その 1)　(ここは 1 次元配列の方法と同様。)

```
double *abody;
```

とポインターを宣言しておいて、

```
if ((abody = (double *)malloc(sizeof(double) * m * n)) == NULL) {
    エラーの場合の処理
}
```

のように動的に記憶領域を確保する。

ポインター配列の設定 (準備その 2)

```
double **a;
....
if ((a == (double *)(malloc(sizeof(double *) * m))) == NULL) {
    エラーの場合の処理
}
```

のようにポインターへのポインターを宣言して、必要な領域を確保した後で、行列の各行の先頭のアドレスを

```
for (i = 0; i < m; i++)
    a[i] = abody + i * n;
```

とセットする。ここまでは大変だが、

成分のアクセス (i, j) 成分 $(0 \leq i \leq m-1, 0 \leq j \leq n-1)$ をアクセスする時は単に

```
a[i][j]
```

と書けばよい。

4 二つの方法の優劣

いずれが優れているか、決定打はないという感じである。

- 1 次元配列の方法は、二重添字から番地を計算するのがとにかく面倒である (マクロで一応は処理できるが、行列が増えると個数分のマクロを定義せねばならず、ちょっとイヤミ)。
- 1 次元配列の方法は、二重添字から番地を計算するのに、乗算が必要で、CPU によっては時間がかかる。
- ポインター配列の方法は、余分なメモリーを必要とする。
- ポインター配列の方法は、成分へのアクセスに、余分なメモリー・アクセスを必要とする。システムによっては時間がかかる。(少し工夫したコーディングをすれば無駄は減少するが、面倒である。)

- ポインター配列の方法では、行列の行の交換が、次のように実際の成分を移動せずに実現できるので、非常に高速。

```
double *ap;
...
ap = a[i]; a[i] = a[j]; a[j] = ap;
```

5 サンプル・プログラム

ポインター配列の方法で、行列を確保する関数 new_matrix() を以下に示す。

```
/*
 * test4.c
 */

#include <stdio.h>
#include <math.h>

typedef double *vector;
typedef vector *matrix;

/* m行n列の行列を作る */
matrix new_matrix(m, n)
int m, n;
{
    int i;
    matrix a;
    vector abody;

    if ((a = (matrix)malloc(m * sizeof(vector))) == NULL)
        return NULL;
    if ((abody = (vector)malloc(m * n * sizeof(double))) == NULL)
        return NULL;
    for (i = 0; i < m; i++)
        a[i] = abody + n * i;
    return a;
}

void free_matrix(a)
matrix a;
```

```
{
    free(a[0]);
    free(a);
}

int
main()
{
    int m,n,i,j;
    matrix a;
    printf("m,n: "); scanf("%d %d", &m, &n);
    a = new_matrix(m,n);
    if (a == NULL) {
        fprintf(stderr, "new_matrix() に失敗\n");
        exit(1);
    }
    kuku(a,m,n);
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++)
            printf("a[%d][%d]=%g ", i, j, a[i][j]);
        printf("\n");
    }
    free_matrix(a);
    return 0;
}

kuku(a,m,n)
matrix a;
int m,n;
{
    int i,j;
    for (i = 0; i < m; i++)
        for (j=0; j < n; j++)
            a[i][j] = (i + 1) * (j + 1);
}

#ifdef  __TURBOC__
double dummy() { return sin(0.0); }
#endif
```

6　C++ の場合

C++ でポインター配列に相当することをするには

```
/*
 * cpptest3.C --- ひとまずの完成
 */

#include <iostream.h>
#include <math.h>

typedef double scalar;
typedef scalar *vector;
typedef vector *matrix;

/* m 行 n 列の行列を作る */
matrix new_matrix(int m, int n)
{
    vector abody;
    matrix a;

    a = new vector [m];
    if (a == NULL) return NULL;
    abody = new scalar [m * n];
    if (abody == NULL) {
        delete a;
        return NULL;
    }
    for (int i = 0; i < m; i++) {
        a[i] = abody;
    abody += n;
    }
    return a;
}

free_matrix(matrix a)
{
    delete a[0];
    delete a;
}

int
main()
{
```

```
    int m,n,i,j;
    matrix a;
    cout << "m,n: "; cin >> m >> n;
    a = new_matrix(m,n);
    if (a == NULL) {
        cerr << "new_matrix() に失敗\n";
        exit(1);
    }
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            a[i][j] = (i+1)*(j+1);
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++)
            cout << "a[" << i << "][" << j << "]=" << a[i][j] << " ";
        cout << "\n";
    }
    return 0;
}
```