C 言語のための matrix ライブラリィ

桂田 祐史

1998 年 6 月 2 日

1 matrix ライブラリィ

1.1 C 言語で数値計算するユーザーの憂鬱

最近では、伝統的に Fortran を用いてきたような数値計算プログラミングにも、C 言語が使われるようになってきたが、いくつか問題がある。

1. 倍精度浮動小数点型 (double) 以外には、色々問題がある。
   - 単精度小数点型 (float) データの演算の際に、勝手に倍精度小数点型データに変換されることがある。
   - 単精度小数点型 (float) を引数とするライブラリィ関数がそろっていないことが多い。
   - 4 倍精度浮動小数点型 (long float?) が使える処理系が少ない。
2. C 言語には Fortran の整合配列 (conformant array) 機能がない。
3. 信頼できる数値計算ライブラリィがあまり揃っていない。

問題点 1 の意味は分かりやすい。これに対しては、ANSI C 規格である程度まで解決はできたし、倍精度計算だけで我慢することで、それほど大きな不都合が起こるわけでもない。

問題点 3 には、C 言語を使って数値計算する歴史が浅いことも原因の一つではあるが、実は問題点 2 が大きな原因になっていると言える。そのため、単に時間が経過するだけでは問題が解決しない可能性が高い。

そこで、問題点 2 について解説しよう。整合配列とは、不定長の寸法を持つ配列を副プログラムとの間でやり取りするための仕掛けである。例えば

```
subroutine sub(a, m, n)
int m, n
real a(m,n)
...
end
```

のように、2 次元配列の「寸法」 m, n を変数で受け取る副プログラムが書けて、

```
real a(5,3), b(10,6)
```

のように宣言した 2 次元配列 a, b を

```
call sub(a, 5, 3)
...
call sub(a, 10, 6)
```

のようにサブルーチンに sub 渡すことができる。

数値計算には、行列や格子点上の関数値など、二重添字を持った数列データが現れることが多い。Fortran をプログラミング言語に選択したユーザーは、伝統的に 2 次元配列を用いることでそれらのデータを表現して、うまくやってきた。その際、整合配列という仕組みは、一般性のあるサブルーチン・ライブラリィを作るためになくてはならないものであった。

C 言語に整合配列の機能がないのは、なかなか困ったことである。どうやって解決しているかと言うと、実は

根本的な解決策はない

というのが本当のところであろう。

1.2 matrix ライブラリのねらい

matrix ライブラリィは、ポインターを用いて、二重添字を持った数列データを扱う関数を集めたライブラリィである。これを用いると、

```
void sub(a, m, n)
  int m, n;
  matrix a;
{
...
}
```

のようなプログラムが書ける。つまり整合配列がないことを埋め合わせることができる。

1.3 matrix ライブラリの提供するもの

データ型 1. `scalar` 実数を表すためのデータ型 (実体は double)

2. `vector` scalar を成分とするベクトルを表すためのデータ型
3. `matrix` scalar を成分とする行列を表すためのデータ型

関数 **vector newvector(int n)** n 次元の `vector` を動的に確保する `vector x;` のように宣言しておいて

```
x = newvector(n);
if (x == NULL) エラーの処理;
```

とするのが典型的な使い方である。

void freevector(vector v) `newvector()` で確保した `vector` を解放する。

matrix newmatrix(int m, int n) m × n 型の `matrix` を動的に確保する `matrix a;` のように宣言しておいて

```
a = newmatrix(m, n);
if (a == NULL) エラーの処理;
```

とするのが典型的な使い方である。

void freematrix(matrix m) `newmatrix()` で確保した `matrix` を解放する。

double dotprod(int n, vector a, vector b) 二つの `vector` a, b の内積を計算する

1.4 matrix ライブラリィを使用したプログラムの例

```
#include <stdio.h>
#include <matrix.h>

main()
{
    int i, j, n;
    vector u, v, bj;
    matrix a, b, c;

    printf("n="); scanf("%d", &n);

    /* n 次元のベクトル u, v を準備 */
    if ((u = new_vector(n)) == NULL) {
        fprintf(stderr, "cannot allocate u\n"); exit(1);
    }
    if ((v = new_vector(n)) == NULL) {
        fprintf(stderr, "cannot allocate v\n"); exit(1);
    }
    /* u, v の全成分を 1 にする */
    for (i = 0; i < n; i++) u[i] = v[i] = 1.0;
    /* u, v の内積を計算して表示する */
    printf("dotprod(n, u, v)=%g\n", dotprod(n, u, v));

    /* n 次正方行列 a, b, c を準備 */
    if ((a = new_matrix(n, n)) == NULL) {
        fprintf(stderr, "cannot allocate a\n"); exit(1);
    }
    if ((b = new_matrix(n, n)) == NULL) {
        fprintf(stderr, "cannot allocate b\n"); exit(1);
    }
    if ((c = new_matrix(n, n)) == NULL) {
        fprintf(stderr, "cannot allocate c\n"); exit(1);
    }
    /* n 次元のベクトル bj を準備 */
    if ((bj = new_vector(n)) == NULL) {
        fprintf(stderr, "cannot allocate v\n"); exit(1);
    }
    /* a, b の全成分を 1 にする */
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            a[i][j] = b[i][j] = 1.0;
    for (j = 0; j < n; j++) {
        /* 行列 b の第 j 列ベクトルを bj にコピーする */
        for (i = 0; i < n; i++) bj[i] = b[i][j];
        /* c の (i,j) 成分を a の第 i 行と b の第 j 列の内積とする
         * i.e. 行列 c を行列 a, b の積とする */
        for (i = 0; i < n; i++) c[i][j] = dotprod(n, a[i], bj);
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
            printf("%g ", c[i][j]);
        printf("\n");
    }
    return 0;
}
```

6701 号室の環境で、コンパイルするには -lmatrix というリンカー・オプションをつければよい[1]。

```
cc -o testmat testmat.c -lmatrix -lm
```

vector は、実際には double へのポインターなので、double 型の配列 double [数] や、double 型へのポインター*double と互換性がある。

A ソース

```
/*
 * matrix.c
 */

#include "matrix.h"

matrix new_matrix(nrow, ncol)
int nrow, ncol;
{
    int i;
    vector ap;
    matrix a;

    if ((a = (matrix)malloc(nrow * sizeof(vector))) == NULL)
        return NULL;
    if ((ap = (vector)malloc(nrow * ncol * sizeof(scalar))) == NULL) {
        free(a);
        return NULL;
    }
    for (i = 0; i < nrow; i++)
        a[i] = ap + (i * ncol);
    return a;
}

void free_matrix(a)
matrix a;
{
    free(a[0]);
    free(a);
}
```

[1]桂田研ノートパソコンでは -L/usr/local/lib -lmatrix で OK.

```
/*
 * vector.c
 */

#include "matrix.h"

vector new_vector(n)
int n;
{
    return (vector)malloc(sizeof(scalar) * n);
}

void free_vector(v)
vector v;
{
    free(v);
}
```

```
/*
 * dotprod.c
 */

#include "matrix.h"

double dotprod(n, u, v)
int n;
vector u, v;
{
    int i, n4;
    double s;

    s = 0;
    n4 = n & 3; /* n % 4 */
    for (i = 0; i < n4; i++)
        s += u[i] * v[i];
    for (i = n4; i < n; i += 4)
        s += u[i] * v[i] + u[i + 1] * v[i + 1]
            + u[i + 2] * v[i + 2] + u[i + 3] * v[i + 3];
    return s;
}
```

```
/*
 * matrix.h
 */

#ifndef MATUTIL
#define MATUTIL

#include <stdio.h>
#include <stdlib.h>

#ifndef scalar
#define scalar double
#endif

typedef int    *ivector;
typedef float  *fvector;
typedef double *dvector;
typedef scalar *vector;

typedef ivector *imatrix;
typedef fvector *fmatrix;
typedef dvector *dmatrix;
typedef vector *matrix;

ivector new_ivector();
void free_ivector();

fvector new_fvector();
void free_fvector();

dvector new_dvector();
void free_dvector();

vector new_vector();
void free_vector();

imatrix new_imatrix();
void free_imatrix();

fmatrix new_fmatrix();
void free_fmatrix();

dmatrix new_dmatrix();
void free_dmatrix();

matrix new_matrix();
void free_matrix();

double dotprod();
#endif
```

```
#
# Makefile for matrix.a
#

SRCS    =       matrix.c vector.c dotprod.c
OBJS    =       matrix.o vector.o dotprod.o
LIB     =       matrix.a
PROGS   =       testmat test-glsc+ heat2d-e
CC      =       cc
CFLAGS  =       -O4
LIBDIR  =       /usr/local/lib
INCDIR  =       /usr/local/include
JUNK    =       Meta

default: $(LIB)
all: $(LIB) $(PROGS)

matrix.o: matrix.c
vector.o: vector.c
dotprod.o: dotprod.c
matrix.a: $(OBJS)
        ar cr $@ $(OBJS) && ranlib $@
clean:
        rm -f $(LIB) $(OBJS) $(PROGS) $(JUNK)
test: testmat
        testmat
testmat: testmat.c matrix.a
        $(CC) -o $@ $(CFLAGS) testmat.c matrix.a
test-glsc+: test-glsc+.c
        $(CC) -o $@ $(CFLAGS) test-glsc+.c -lglscd -L/usr/lib/X11 -lX11 -lm
heat2d-e: heat2d-e.c $(LIB)
        $(CC) -o $@ $(CFLAGS) heat2d-e.c $(LIB) -lglscd -L/usr/lib/X11 -lX11 -lm
install: matrix.a
        -mv -f $(LIBDIR)/libmatrix.a $(LIBDIR)/libmatrix.a.$$
        -mv -f $(INCDIR)/matrix.h $(INCDIR)/matrix.h.$$
        cp -p matrix.a $(LIBDIR)/libmatrix.a
        ranlib $(LIBDIR)/libmatrix.a
        cp -p matrix.h $(INCDIR)
```