数値解析レポートNo.1

Ec4-39 湯嶋 皓騎

実施環境

- CPU: Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz
- OS: Arch Linux (WSL2 5.15.123.1-microsoft-standard-WSL2)
- コンパイラ: GNU Compiler Collection (GCC) 13.2.1 20230801
- Cライブラリ: GNU C Library (GNU libc) stable release version 2.38.

1. 行列の任意の2行を交換するプログラム

1.1. 実装

行列の任意の2行を交換する関数`swap_matrix_row`を示す.

```
int swap_matrix_row(double **matrix, int row, int i, int j) {
    double *tmp;

    if (row <= i || row <= j) {
        fprintf(stderr,
                "Error: row index is out of range. (row=%d, i=%d, j=%d)\n",
row,
                i, j);
        return -1;
    }
    if (i == j) return 0;  // nothing to do

    tmp = matrix[i];
    matrix[i] = matrix[j];
    matrix[j] = tmp;

    return 0;
}
```

1.2. 動作確認

この関数を用いて課題4のA行列の1行目と3行目を交換する.

行列の行交換を行うプログラムの主要部を示す.

```
#include <stdio.h>

#include "NAbasic.h"

#define MATRIX_FILENAME "k4-input1.csv"

int swap_matrix_row(double **matrix, int row, int i, int j);

int main(void) {
    FILE *matrix_fin;
    double **matrix;
    int row, col;
```

```
    // csvRead 関数で MATRIX_FILENAME からA行列を読み取る

    showMatrix((const double **)matrix, row, col);
    putchar('\n');
    swap_matrix_row(matrix, row, 0, 2);
    showMatrix((const double **)matrix, row, col);
    freeMatrix(matrix);

    return EXIT_SUCCESS;
}
```

関数 `swap_matrix_row` の引数で交換する行番号を指定するが，0-indexedとなっていることに気を付けること．

次にこのプログラムを実行した結果を示す．

```
2.0000  -1.0000 5.0000
-4.0000 2.0000  1.0000
8.0000  2.0000  -1.0000

8.0000  2.0000  -1.0000
-4.0000 2.0000  1.0000
2.0000  -1.0000 5.0000
```

正しく行の交換が行えていることが確認できる．

1.3. 工夫した点

`matrix` は二次元配列で実装されているおり，1行でひとつの配列になっているから，行を交換するときに要素を直接操作するのではなく i 番目の配列と j 番目の配列を交換する方法で実装した．これによって $O(1)$ で行の交換が可能となった．

2. 行列積を利用した内積計算プログラム

ベクトルの内積を1行n列の行列とn行1列の行列の行列積と見て計算することにする．これは以下のように書き表わせる．

$$\mathbf{a} \cdot \mathbf{b} = AB^T$$

ただしA, Bはn次元ベクトル$\mathbf{a}, \mathbf{b}$を1行n列の行列にしたものであり，B^Tは行列Bを転置したものである．

2.1. 実装

`matrix1` と `matrix2` の行列積を計算する関数`calcurate_matrix_product`と
行列積を利用して内積を計算する関数`calcurate_inner_product_by_matrix_product`の実装を示す．

```
double **calcurate_matrix_product(const double **matrix1, int row1, int
col1,
```

```
                                    const double **matrix2, int row2, int
col2) {
    double **result;
    int i, j, k;

    if (col1 != row2) return NULL;  // invalid matrix size
    result = allocMatrix(row1, col2);
    for (i = 0; i < row1; i++) {
        for (j = 0; j < col2; j++) {
            double tmp = 0.0;
            for (k = 0; k < col1; k++) {
                tmp += matrix1[i][k] * matrix2[k][j];
            }
            result[i][j] = tmp;
        }
    }
    return result;
}

double calcurate_inner_product_by_matrix_product(const double *vector1,
                                                 int vector1_length,
                                                 const double *vector2,
                                                 int vector2_length) {
    double **tmatrix, **result_matrix;
    double result;

    tmatrix = tpMatrix((const double **)&vector2, 1, vector2_length);
    result_matrix =
        calcurate_matrix_product((const double **)&vector1, 1,
vector1_length,
                                 (const double **)tmatrix, vector2_length,
1);
    if (result_matrix == NULL) {
        fprintf(stderr, "could not calcurate inner product\n");
        return 0.0;
    }
    // matrix size is 1x1 because row1 x col2 = 1 x 1
    result = result_matrix[0][0];
    freeMatrix(tmatrix);
    return result;
}
```

行列積 AB は A の列数と B の行数が一致するときに定義され，AB の大きさは A の行数と B の列数になる．
定義通りに行列積を計算すると`calcurate_matrix_product`のように実装できる．

`calcurate_inner_product_by_matrix_product`は内積を計算するのでベクトルを2つ引数に取る．
`NABasic`において
ベクトルは`double`型の配列で，行列は`double`型のポインタ配列で定義されている．そのため，ベクトルのアドレスを
`double **`型にキャストすることでベクトルを1行n列の行列として扱うことができる．したがって，
`vector1`と`vector2`の内積は

`vector1`を1行`vector1_length`列の行列にキャストしたものと`vector2`を転置したものの行列積で求められる．

2.2. 動作確認

課題2(予備)(2)の問題で動作確認を行った．

予備演習課題の式を示す．

$$\begin{pmatrix} 1 \\ 5 \\ 8 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 7 \\ -2 \end{pmatrix} = 1 \cdot 3 + 5 \cdot 7 + 8 \cdot (-2) = 22$$

動作確認を行う上で，2つのベクトルをそれぞれCSVファイルに格納した．

動作確認を行う際の定数定義と`main`関数を示す．
ただし，日本語のコメントでコードの非本質的な部分を省略している．

```
#define VECTOR1_FILENAME "k2-input1.csv"
#define VECTOR2_FILENAME "k2-input2.csv"

int main(void) {
    FILE *vector1_fin, *vector2_fin;
    double **matrix1, **matrix2, *vector1, *vector2, inner_product;
    int row1, col1, row2, col2;

    // VECTOR1_FILENAME, VECTOR2_FILENAME からデータを読み取って matrix1,
matrix2 に格納する

    // show warning if either row or column is not 1
    // 行列のサイズが不正なら警告やエラーを出す

    // convert from matrix (anything vector) to column vector
    vector1 = matrix2colVector((const double **)matrix1, row1, col1);
    vector2 = matrix2colVector((const double **)matrix2, row2, col2);
    // ファイルディスクリプタとmatrix1, matrix2 を解放する

    inner_product = calcurate_inner_product_by_matrix_product(
        (const double *)vector1, row1 * col1, (const double *)vector2,
        row2 * col2);
    printf("%f\n", inner_product);

    // vector1, vector2 を解放して return
}
```

このプログラムを実行した結果，`22.000000`と出力され，正しく動作していることが確認できた．

2.3. 工夫した点

`matrix2colVector` 関数を用いて行列を列ベクトルに変換することで，もし入力が
列ベクトル以外だったとしても処理が継続できるようにした．

また，キャストを適切に用いることで値のコピーをなるべく減らしながら処理を記述することができた．

3. 内積計算プログラムを利用したベクトルのなす角度の計算プログラム

ベクトル$\mathbf{a}, \mathbf{b}$がなす角度をθとしたとき，

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos\theta$$

だから，

$$\theta = \arccos\left(\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|} \right) \ [\mathrm{rad}]$$

である．

3.1. 実装

ベクトルのなす角度を計算する関数`calcurate_vector_angle_by_inner_product`の実装を示す．ただし，関数`acos`と`sqrt`，及び定数`M_PI`はヘッダファイル`math.h`で定義されているものである．

```
#include <math.h>

double calcurate_vector_angle_by_inner_product(const double *vector1,
                                               int vector1_length,
                                               const double *vector2,
                                               int vector2_length) {
    double inner_product, vector1_square, vector2_square, angle_rad;

    inner_product = calcurate_inner_product_by_matrix_product(
        vector1, vector1_length, vector2, vector2_length);
    vector1_square = calcurate_inner_product_by_matrix_product(
        vector1, vector1_length, vector1, vector1_length);
    vector2_square = calcurate_inner_product_by_matrix_product(
        vector2, vector2_length, vector2, vector2_length);

    angle_rad = acos(inner_product / sqrt(vector1_square *
vector2_square));
    return angle_rad * 180.0 / M_PI;
}
```

ベクトルの大きさの2乗は自分自身と内積を取ることで計算できるので，新たに
ノルムを計算する関数を定義する必要はない．

関数`acos`の結果は弧度法で表わされるので，$\frac{180}{\pi}$ を乗じることで度数法に変換している．

3.2. 動作確認

課題2(予備)(2)の問題で動作確認を行った．
上に示したθの式で角度を計算すると，$\theta = 72.87161773\ [^\circ]$ と求まった．

次に関数`calcurate_vector_angle_by_inner_product`を用いて計算するプログラムの`main`関数などを示す．こちらも日本語のコメントでコードの非本質的な部分を省略している．

```
int main(void) {
    FILE *vector1_fin, *vector2_fin;
    double **matrix1, **matrix2, *vector1, *vector2, angle;
    int row1, col1, row2, col2;

    // 問2と同様に vector1, vector2 にデータを読み込む

    angle = calcurate_vector_angle_by_inner_product(
        (const double *)vector1, row1 * col1, (const double *)vector2,
        row2 * col2);
    printf("%f\n", angle);

    // vector1, vector2 を解放して return
}
```

このプログラムを実行した結果，`72.871618`と出力され，正しく動作していることが確認できた．

3.3. 工夫した点

ベクトルのノルムを計算する際に，各ベクトルごとに平方根を取らずに最後に一度だけ`sqrt`関数を実行することで近似誤差を減らすことができた．また，関数内では自動変数のみを使用しているためメモリリークの発生が起こらないようにしている．

4. 感想

問1は演習課題4で既に作成済みだったので素早く進めることができた．全体を通してC言語のポインタの使い方をよく思い出しながら作る必要があったが，大きな問題なく取り組むことができた．

5. 要望

特にありません．