TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 - 최대성
c3d4s19@naver.com

2018.05.17. – 055 일차 수업

< matrix_op.h 헤더파일 >

```
#ifndef __MATRIX_OP_H__
#define __MATRIX_OP_H__

#include <stdio.h>

typedef struct _vector{
   double x;
   double y;
   double z;
}vect;

typedef struct _matrix{
   vect a;
   vect b;
   vect c;
}mat;
/*
 *  matrix = { ax, ay, az,
 *             bx, by, bz,
 *             cx, cy, cz }
 */

void init_to_zero_mat(mat* A);
void init_to_I_mat(mat* A);
double get_mat_elem(mat* A ,int row, int col);
void print_vect(vect* v);
void print_matrix(mat* A);
void scale_mat_col(double scale, mat* A, int
col);
void chg_elem(double* a, double* b);
void chg_mat_col(mat* A, int col1, int col2);
void scale_matrix(double scale, mat* A, mat*
res);
void add_matrix(mat* A, mat* B, mat* res);
void sub_matrix(mat* A, mat* B, mat* res);
void product_mat_by_vect(mat* A, vect* B,
vect* res);
void product_mat_by_mat(mat* A, mat* B, mat*
res);
double determinant_matrix(mat* A);
double minor_determinant(mat* A, int row, int
col);
void cofactors_matrix(mat* A, mat* res);
void transpose_matrix(mat* A, mat* res);
void adjoint_matrix(mat* A, mat* res);
void inverse_matrix_general(mat* A, mat* res);
void sub_scaled_col(mat* A, mat* B, int col1, int
col2, int row);
void inverse_matrix_gaussian_elim(mat A, mat*
res);
void cramers_rule(mat* A, vect* B, vect* res);

#endif
```

< matrix_op.c C 파일 >

```
#include "matrix_op.h"

/*
 *  matrix = { ax, ay, az,
 *             bx, by, bz,
 *             cx, cy, cz }
 */

void init_to_zero_mat(mat* A){
   for(int i = 0; i < 9; i++)
      *((double*)A + i) = 0;
}

void init_to_I_mat(mat* A){
   init_to_zero_mat(A);
   for(int i = 0; i < 3; i++)
      *((double*)A + i*(1 + 3)) = 1;
}

double get_mat_elem(mat* A ,int row, int col){
   return *((double*)A + (row + 3 * col));
}

void print_vect(vect* v){
   for(int i = 0; i < 3; i++)
      printf("%lf     ",*((double*)v + i) );
}

void print_matrix(mat* A){
   double tmp;
   for(int i = 0; i < 3; i++){
      for(int j = 0; j < 3; j++){
         printf("%lf    ", get_mat_elem(A, j, i));
      }
      printf("\n");
   }
}

void scale_mat_col(double scale, mat* A, int
col){
   for(int i = 0; i < 3; i++)
      *((double*)A + 3*col + i) = scale *
get_mat_elem(A, i, col);
}

void chg_elem(double* a, double* b){
   double tmp = *a;
   *a = *b;
   *b = tmp;
}

void chg_mat_col(mat* A, int col1, int col2){
   for(int i = 0; i < 3; i++)
      chg_elem( ((double*)A + 3*col1 + i) ,
((double*)A + 3*col2 + i) );
}

void scale_matrix(double scale, mat* A, mat*
```

```
res){
    for(int i = 0; i < 9; i++)
        *((double*)res + i) = scale *
get_mat_elem(A, i, 0);
}

void add_matrix(mat* A, mat* B, mat* res){
    for(int i = 0; i < 9; i++)
        *((double*)res + i) = get_mat_elem(A, i, 0)
+ get_mat_elem(B, i, 0);
}

void sub_matrix(mat* A, mat* B, mat* res){
    for(int i = 0; i < 9; i++)
        *((double*)res + i) = get_mat_elem(A, i, 0)
- get_mat_elem(B, i, 0);
}

void product_mat_by_vect(mat* A, vect* B,
vect* res){
    for(int i = 0; i < 3; i++){
        *((double*)res + i) = 0;
        for(int j = 0; j < 3; j++)
            *((double*)res + i) += get_mat_elem(A,
j, i) * (*((double*)res + j));
    }
}

void product_mat_by_mat(mat* A, mat* B, mat*
res){
    for(int i = 0; i < 9; i++){
        *((double*)res + i) = 0;
        for(int j = 0; j < 3; j++)
            *((double*)res + i) += get_mat_elem(A,
j, i / 3) * get_mat_elem(B, i % 3, j);
    }
}

double determinant_matrix(mat* A){
    double res = 0;
    for(int row = 0; row < 3; row++){
        res += get_mat_elem(A, row, 0) *
minor_determinant(A, row, 0);
    }
    return res;
}

double minor_determinant(mat* A, int row, int
col){
    return
    (get_mat_elem(A, (row + 1) % 3, (col + 1) %
3) * get_mat_elem(A, (row + 2) % 3, (col + 2) %
3)
    -get_mat_elem(A, (row + 1) % 3, (col + 2) %
3) * get_mat_elem(A, (row + 2) % 3, (col + 1) %
3));
}

void cofactors_matrix(mat* A, mat* res){
    for(int col = 0; col < 3; col++){
        for(int row = 0; row < 3; row++){
            *((double*)res + (row + 3 * col)) =
minor_determinant(A, row, col);
        }
    }
}

void transpose_matrix(mat* A, mat* res){
    double tmp;
    for(int col = 0; col < 3; col++){
        for(int row = 0; row < 3; row++){
            *((double*)res + (row + 3*col)) =
*((double*)A + (3*row + col));
        }
    }
}

void adjoint_matrix(mat* A, mat* res){
    mat tmp;
    cofactors_matrix(A, &tmp);
    transpose_matrix(&tmp, res);
}

void inverse_matrix_general(mat* A, mat* res){
    double det = determinant_matrix(A);
    if(A == 0)
        printf("역행렬 없음!\n");
    else{
        mat tmp;
        adjoint_matrix(A, &tmp);
        scale_matrix(1/det, &tmp, res);
    }
}

//col1 을 scale 후 col2 와 빼기 ( A, B 배열 둘 다 적용됨 )
void sub_scaled_col(mat* A, mat* B, int col1, int
col2, int row){
    if(!get_mat_elem(A, row, col2))
        return;
    // 행렬(row, col1) 요소를 기준으로 행렬 (row, col2)
요소를 0 으로 만듬
    double scale = *((double*)A + row +
3*col2) / *((double*)A + row + 3*col1);
    for(int row = 0; row < 3; row++){
        *((double*)A + (row + 3*col2)) =
get_mat_elem(A, row, col2) - scale *
get_mat_elem(A, row, col1);
        *((double*)B + (row + 3*col2)) =
get_mat_elem(B, row, col2) - scale *
get_mat_elem(B, row, col1);
    }
}

void inverse_matrix_gaussian_elim(mat A, mat*
res){
    double scale;
    int col;

    if(!determinant_matrix(&A)){
        printf("역행렬 없음!\n");
        return;
    }
    init_to_I_mat(res);

    for(int i = 0; i < 3; i++){
        col = i;
        while( !get_mat_elem(&A, i, col) )
            chg_mat_col(&A, i, ++col % 3);
        scale = 1 / get_mat_elem(&A, i, i);
        scale_mat_col(scale, &A, i);
```

```
        scale_mat_col(scale, res, i);
        sub_scaled_col(&A, res, i, (i + 1) % 3, i);
        sub_scaled_col(&A, res, i, (i + 2) % 3, i);
    }
}

void cramers_rule(mat* A, vect* B, vect* res){
    double det = determinant_matrix(A);
    mat tmp;
    for(int i = 0; i < 3; i++){
        tmp = *A;
        for(int j = 0; j < 3; j++)
            *((double*)&tmp + (i + 3*j)) =
*((double*)B + j );
        *((double*)res + i) =
determinant_matrix(&tmp) / det;
    }
}

#define THIS_IS_FOR_TEST 1
#if THIS_IS_FOR_TEST

// main() for test functions
int main(){
    vect R = {2, 3, 1};

    mat A = {1, 1, 0,
             2, 9, 2,
             1, 0, 1};

    mat B = {0, 0, 1,
             1, 2, 1,
             0, 1, 1};

    mat C = {4, 5, 0,
             2, 1, 2,
             1, 5, 2};

    double tmp;
    mat res_mat;
    mat tmp_mat;
    vect res_vect;

    printf("sub matrix\n");
    sub_matrix(&A, &B, &res_mat);
    print_matrix(&res_mat);
    printf("\n");

    printf("product mat x vect\n");
    product_mat_by_vect(&A, &R, &res_vect);
    print_vect(&res_vect);
    printf("\n");printf("\n");

    printf("product mat x mat\n");
    product_mat_by_mat(&A, &B, &res_mat);
    print_matrix(&res_mat);
    printf("\n");

    printf("cofactor matrix\n");
    cofactors_matrix(&A, &res_mat);
    print_matrix(&res_mat);
    printf("\n");

    printf("adjoint matrix\n");
    adjoint_matrix(&A, &res_mat);
    print_matrix(&res_mat);
    printf("\n");

    printf("determinant matrix\n");
    printf("%lf \n", determinant_matrix(&A));
    printf("\n");

    printf("product mat x inv_mat\n");
    product_mat_by_mat(&A, &tmp_mat,
&res_mat);
    print_matrix(&res_mat);
    printf("\n");

    printf("cramers_rule\n");
    cramers_rule(&C, &R, &res_vect);
    print_vect(&res_vect);
    printf("\n");printf("\n");

    printf("sub_scaled_col ( before )\n");
    print_matrix(&A);
    printf("\n");
    print_matrix(&B);
    sub_scaled_col(&A, &B ,0 ,2, 0);
    printf("sub_scaled_col ( after )\n");
    print_matrix(&A);
    printf("\n");
    print_matrix(&B);
    printf("\n");

    init_to_zero_mat(&res_mat);
    print_matrix(&res_mat);
    printf("\n");

    init_to_I_mat(&res_mat);
    print_matrix(&res_mat);
    printf("\n");

    printf("inverse_matrix_gaussian_elim\n");
    inverse_matrix_gaussian_elim(A, &res_mat);
    print_matrix(&res_mat);
    printf("\n");printf("\n");

    printf("inverse matrix\n");
    inverse_matrix_general(&A, &res_mat);
    print_matrix(&res_mat);
    printf("\n");
}
#endif
```