TI DSP, MCU 및 Xilinx Zynq FPGA
프로그래밍 전문가 과정

2018-05-17 (56 회차)

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 - 정유경
ucong@naver.com

1. 역행렬 구하는 3 가지 방법

- 가우스 소거법

- 정석

- 크래머 법칙

2. 이론을 바탕으로 프로그램 구현하기

1. 행렬의 덧셈, 뺄셈, 곱셈

2. 역행렬 구하기(정석)

3. 연립방정식(가우스 소거법)

4. 역행렬구하기(가우스 소거법)

5. 크래머 공식

6. 행렬의 전치

7. 행렬의 판별식

*. 선생님 Tip

for 문 사용시 branch 가 들어가고 파이프라인이 깨진다. 단순한 연산에는 for 문을 사용하면 안된다. 다 풀어서 써라.

(1-1) mat3.h

```
#ifndef __MAT3_H__
#define __MAT3_H__

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

//float A[][3] or float (*A)[3]
void init_mat(float (*A)[3])
{
	int i, j;
	for(i = 0; i < 3; i++)
		for(j = 0; j < 3; j++)
			A[i][j] = i+j;
		//	A[i][j] = rand() % 4;
}
void print_mat(float (*R)[3])
{
	int i, j;
```

```
	for(i = 0; i < 3; i++)
	{
		for(j = 0; j < 3; j++)
			printf("%4.1f", R[i][j]);
		printf("\n");
	}
	printf("\n");
}

void add_mat(float (*A)[3], float (*B)[3], float (*R)[3])
{
	int i, j;

	for(i = 0; i < 3; i++)
		for(j = 0; j < 3; j++)
			R[i][j] = A[i][j] + B[i][j];
}

void sub_mat(float (*A)[3], float (*B)[3], float (*R)[3])
{
	int i, j;

	for(i = 0; i < 3; i++)
		for(j = 0; j < 3; j++)
			R[i][j] = A[i][j] - B[i][j];
}

void scale_mat(float scale_factor, float (*A)[3], float (*R)[3])
{
	int i, j;

	for(i = 0; i < 3; i++)
		for(j = 0; j < 3; j++)
			R[i][j] = scale_factor * A[i][j];
}


void mul_mat(float (*A)[3], float (*B)[3], float (*R)[3])
{
	/*간단한 연산은 되도록이면 for 문을 사용하지 않는다.
	분기할때마다 파이프라인이 깨져서 속도효율이 떨어진다.*/
	R[0][0] = A[0][0]*B[0][0]+A[0][1]*B[1][0]+A[0][2]*B[2][0];
	R[0][1] = A[0][0]*B[0][1]+A[0][1]*B[1][1]+A[0][2]*B[2][1];
```

```
	R[0][2] = A[0][0]*B[0][2]+A[0][1]*B[1][2]+A[0][2]*B[2][2];

	R[1][0] = A[1][0]*B[0][0]+A[1][1]*B[1][0]+A[1][2]*B[2][0];
	R[1][1] = A[1][0]*B[0][1]+A[1][1]*B[1][1]+A[1][2]*B[2][1];
	R[1][2] = A[1][0]*B[0][2]+A[1][1]*B[1][2]+A[1][2]*B[2][2];

	R[2][0] = A[2][0]*B[0][0]+A[2][1]*B[1][0]+A[2][2]*B[2][0];
	R[2][1] = A[2][0]*B[0][1]+A[2][1]*B[1][1]+A[2][2]*B[2][1];
	R[2][2] = A[2][0]*B[0][2]+A[2][1]*B[1][2]+A[2][2]*B[2][2];
}

float det_mat(float (*A)[3])
{
	return A[0][0] * (A[1][1] * A[2][2] - A[1][2] * A[2][1]) +
	     A[0][1] * (A[1][2] * A[2][0] - A[1][0] * A[2][2]) +
	     A[0][2] * (A[1][0] * A[2][1] - A[1][1] * A[2][0]);
}

void trans_mat(float (*A)[3], float (*R)[3])
{
	R[0][0] = A[0][0];
	R[1][1] = A[1][1];
	R[2][2] = A[2][2];

	R[0][1] = A[1][0];
	R[1][0] = A[0][1];

	R[0][2] = A[2][0];
	R[2][0] = A[0][2];

	R[2][1] = A[1][2];
	R[1][2] = A[2][1];
}

void adj_mat(float (*A)[3], float (*R)[3])
{
	R[0][0] = A[1][1] * A[2][2] - A[1][2] * A[2][1];
	R[0][1] = A[0][2] * A[2][1] - A[0][1] * A[2][2];
	R[0][2] = A[0][1] * A[1][2] - A[0][2] * A[1][1];

	R[1][0] = A[1][2] * A[2][0] - A[1][0] * A[2][2];
	R[1][1] = A[0][0] * A[2][2] - A[0][2] * A[2][0];
	R[1][2] = A[0][2] * A[1][0] - A[0][0] * A[1][2];
```

```
	R[2][0] = A[1][0] * A[2][1] - A[1][1] * A[2][0];
	R[2][1] = A[0][1] * A[2][0] - A[0][0] * A[2][1];
	R[2][2] = A[0][0] * A[1][1] - A[0][1] * A[1][0];
}

/*판별식으로 역행렬의 존재여부를 판단하고,
역행렬구해서 행렬 R 에 저장*/
bool inv_mat(float (*A)[3], float (*R)[3])
{
	float det;

	det = det_mat(A);

	if(det == 0.0)
		return false;

	adj_mat(A, R);
#ifdef __DEBUG__ // 사용법?
	printf("Adjoint Matrix\n");
	print_mat(R);
#endif
	scale_mat(1.0 / det, R, R);

	return true;
}
/*크래머 공식을 사용하기 위해 detX, detY, detZ 를 구한다.*/
void molding_mat(float (*A)[3], float *ans, int idx, float (*R)[3])
{
	int i, j;

	for(i = 0; i < 3; i++)
	{
		for(j = 0; j < 3; j++)
		{
			if(j == idx)
				continue;
			R[i][j] = A[i][j];
		}
		// idx 열의 성분을 ans[i]와 대체한다.
		R[i][idx] = ans[i];
	}
}
```

```
void crammer_formula(float (*A)[3], float *ans, float *xyz)
{
	float detA, detX, detY, detZ;
	float R[3][3] = {};

	detA = det_mat(A);

	molding_mat(A, ans, 0, R);
	detX = det_mat(R);

	molding_mat(A, ans, 1, R);
	detY = det_mat(R);

	molding_mat(A, ans, 2, R);
	detZ = det_mat(R);

	xyz[0] = detX / detA; // x
	xyz[1] = detY / detA; // y
	xyz[2] = detZ / detA; // z
}

void print_vec3(float *vec)
{
	int i;

	for(i = 0; i < 3; i++)
		printf("%4.1f", vec[i]);

	printf("\n");
}

void create_3x4_mat(float (*A)[3], float *ans, float (*R)[4])
{
	int i, j;

	for(i = 0; i < 3; i++)
	{
		for(j = 0; j < 3; j++)
			R[i][j] = A[i][j];

		R[i][3] = ans[i];
	}
}
```

```
void print_3x4_mat(float (*R)[4])
{
	int i, j;

	for(i = 0; i < 3; i++)
	{
	for(j = 0; j < 4; j++)
			printf("%4.1f", R[i][j]);
	printf("\n");
	}
	printf("\n");
}

// 연립방정식을 가우시안으로 풀기위한 과정
void adjust_3x4_mat(float (*A)[4], int idx, float (*R)[4])
{
	int i, j;
	float div_factor;
	// idx: 어떤 열의 숫자(A[idx][idx])를 0 으로 만들것인지 지정
	for(i = idx + 1; i < 3; i++)
	{
		div_factor = -A[i][idx] / A[idx][idx];
		printf("div_factor = %f\n", div_factor);
		/*각 열에 대하여 */
		for(j = 0; j < 4; j++)
			R[i][j] = A[idx][j] * div_factor + A[i][j];
	}
}

void finalize(float (*R)[4], float *xyz)
{
	xyz[2] = R[2][3] / R[2][2];
	xyz[1] = (R[1][3] - R[1][2] * xyz[2]) / R[1][1];
	xyz[0] = (R[0][3] - R[0][2] * xyz[2] - R[0][1] * xyz[1]) / R[0][0];
}

void gauss_elimination(float (*A)[3], float *ans, float *xyz)
{
	float R[3][4] = {};

	create_3x4_mat(A, ans, R);

	adjust_3x4_mat(R, 0, R);
```

```
        adjust_3x4_mat(R, 1, R);

        finalize(R, xyz);
}

void create_3x6_mat(float (*A)[3], float (*R)[6])
{
        int i, j;

        for(i = 0; i < 3; i++)
                for(j = 0; j < 3; j++)
                {
                        R[i][j] = A[i][j];

                        if(i == j)
                                R[i][j + 3] = 1;
                        else
                                R[i][j + 3] = 0;
                }
}

void print_3x6_mat(float (*R)[6])
{
        int i, j;

        for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 6; j++)
            printf("%10.4f", R[i][j]);
        printf("\n");
    }
    printf("\n");
}

void adjust_3x6_mat(float (*A)[6], int idx, float (*R)[6])
{
    int i, j;
    float div_factor, scale;

        scale = A[idx][idx];

    for(i = idx + 1; i < 3; i++)
    {
        div_factor = -A[i][idx] / A[idx][idx];
```

```
        printf("div_factor = %f\n", div_factor);


                if(div_factor == 0.0)
                        continue;
        for(j = 0; j < 6; j++)
           R[i][j] = A[idx][j] * div_factor + A[i][j];
    }
        for(j = 0; j < 6; j++)
                R[idx][j] = A[idx][j] / scale;
}


void gauss_elim_mat(float (*A)[3], float (*R)[3])
{
        float mid[3][6] = {};


        create_3x6_mat(A, mid);


        adjust_3x6_mat(mid, 0, mid);


        adjust_3x6_mat(mid, 1, mid);
}
#endif
```

(1-2) mat3.c

```
#include "mat3.h"
int main(void)
{
        bool inv_flag;


        float test[3][3] = {{2.0, 0.0, 4.0}, {0.0, 3.0, 9.0}, {0.0, 0.0,
1.0}};
        float stimul[3][3] = {{2.0, 4.0, 4.0}, {6.0, 2.0, 2.0}, {4.0, 2.0,
4.0}};
        float ans[3] = {12.0, 16.0, 20.0};
        float xyz[3] = {};


        float A[3][3] = {};
        float B[3][3] = {};
        float R[3][3] = {};


        srand(time(NULL));


        printf("Init A Matrix\n");
        init_mat(A);
        print_mat(A);
```

```
printf("Init B Matrix\n");
init_mat(B);
print_mat(B);

printf("A + B Matrix\n");
add_mat(A, B, R);
print_mat(R);

printf("A - B Matrix\n");
sub_mat(A, B, R);
print_mat(R);

printf("Matrix Scale(A)\n");
scale_mat(0.5, A, R);
print_mat(R);

printf("AB Matrix\n");
mul_mat(A, B, R);
print_mat(R);

printf("det(A) = %f\n", det_mat(A));
printf("det(B) = %f\n", det_mat(B));

printf("\nA^T(Transpose) Matrix\n");
trans_mat(A, R);
print_mat(R);

printf("B^T(Transpose) Matrix\n");
trans_mat(B, R);
print_mat(R);

printf("A Inverse Matrix\n");
inv_flag = inv_mat(A, R);
if(inv_flag)
        print_mat(R);
else
        printf("역행렬 없다!\n");

printf("test Inverse Matrix\n");
inv_flag = inv_mat(test, R);
if(inv_flag)
        print_mat(R);
else
        printf("역행렬 없다!\n");
```

```
	printf("크래머 공식 기반 연립 방정식 풀기!\n2x + 4y + 4z = 12\n6x +
2y + 2z = 16\n4x + 2y + 4z = 20\n");

	crammer_formula(stimul, ans, xyz);

	print_vec3(xyz);


	printf("가우스 소거법 기반 연립 방정식 풀기!(문제 위의 것과 동일함)\n");

	gauss_elimination(stimul, ans, xyz);

	print_vec3(xyz);


	printf("가우스 소거법으로 역행렬 구하기!\n");

	gauss_elim_mat(test, R);

	print_mat(R);


	return 0;

}
```