TI DSP, MCU, Xilinx Zynq FPGA 기반의 프로그래밍 전문가 과정

<공학 수학>

2018.05.17 - 56 일차

강사 - Innova Lee(이상훈)

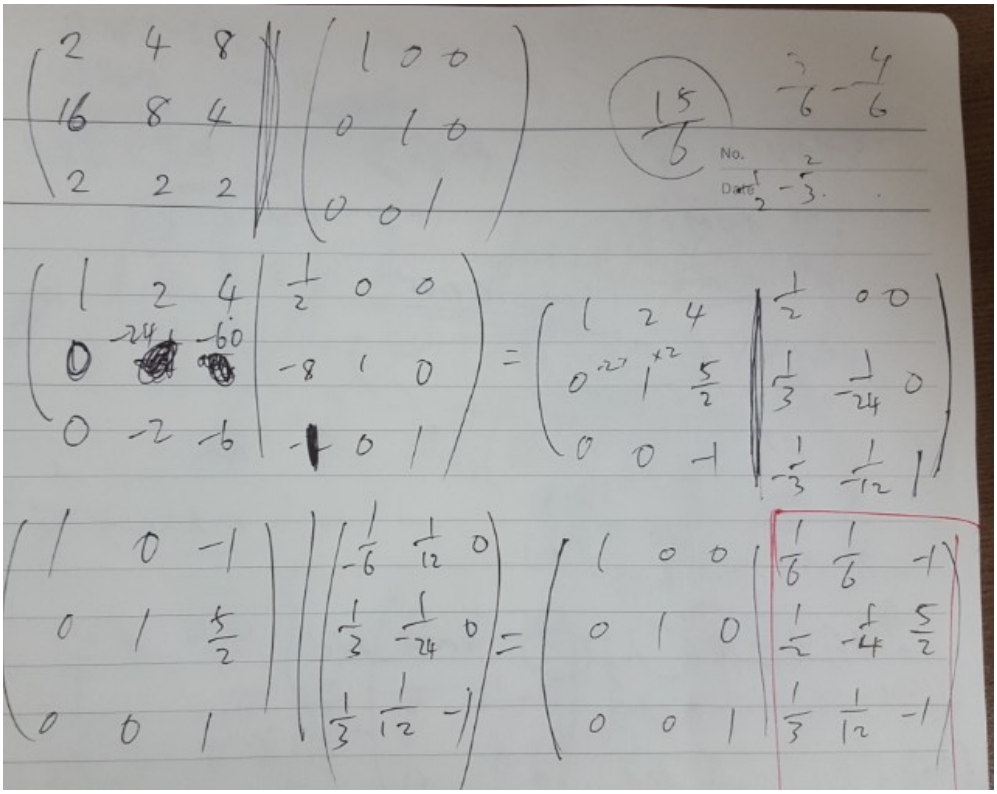
gcccompil3r@gmail.com

학생 - 안상재

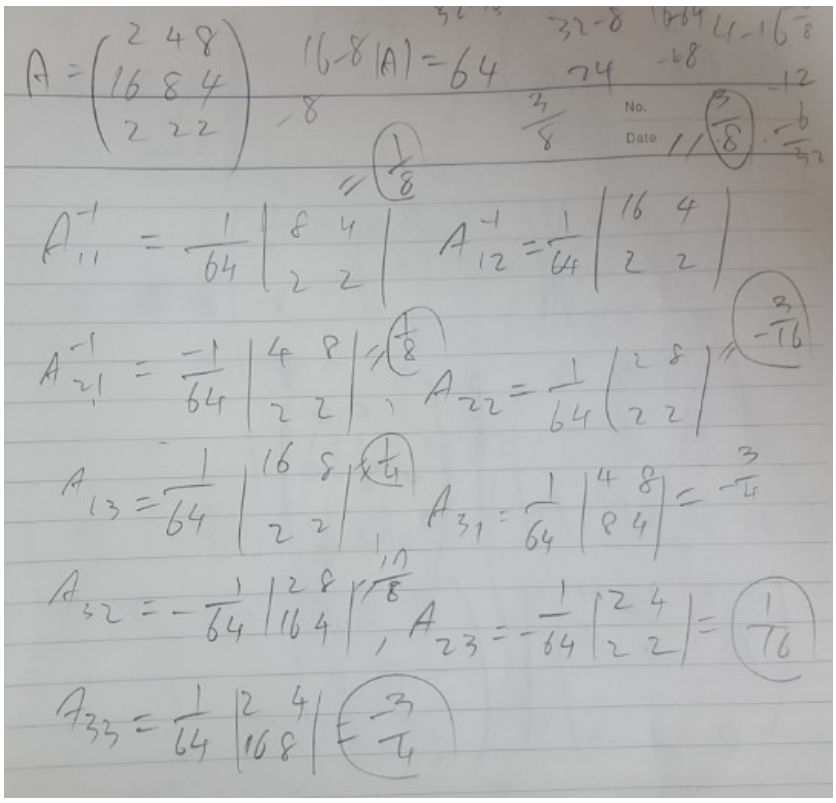
sangjae2015@naver.com

1. 가우스 조르단 소거법

A | I => I | A^-1 (I 는 단위 행렬)
A 를 숫자 조작을 통해서 I 로 만들면 I 는 A^-1 가 됨.

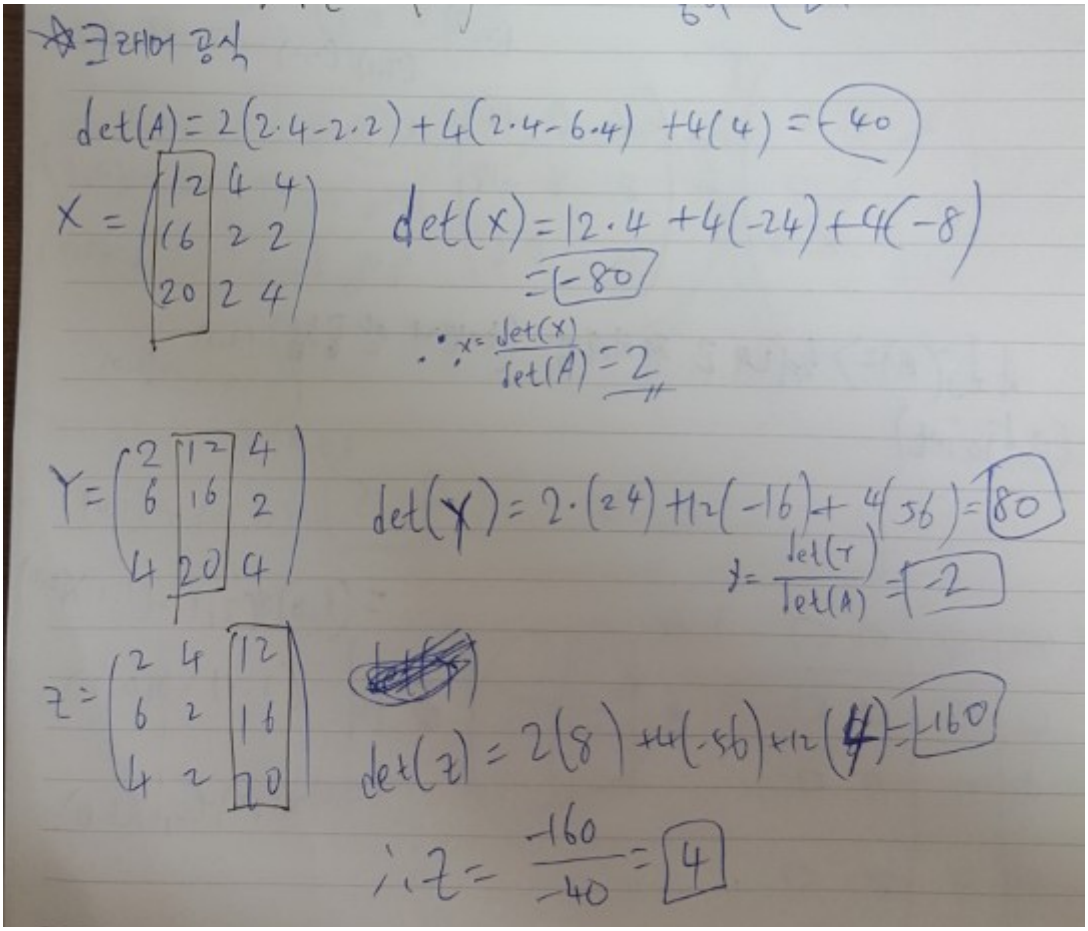


2. 역행렬 구하는 정석적인 방법 (A^-1 = adj(A) / det(A))



3. 크래머 공식

- 연립방정식의 해를 행렬의 판별식을 이용해서 쉽게 구함.



<행렬의 연산 소스코드>
- 덧셈
- 뺄셈
- 곱셈
- 크래머 공식
- 전치
- 역행렬

1. 헤더 파일
- 헤더 파일에 사용할 연산 함수와 필요한 구조체를 정의함.

```
#ifndef __MATRIX_H
#define __MATRIX_H

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

// my_function 구조체에 각 연산에 대한 함수 포인터를 멤버로 선언함
struct my_function{
	void (* print) (float (*)[3]);
	void (* add) (float (*)[3], float (*)[3], float (*)[3]);
	void (* sub) (float (*)[3], float (*)[3], float (*)[3]);
	void (* multiply) (float (*)[3], float (*)[3], float (*)[3]);
	void (* crammer) (float (*)[3], float *, float *);
	void (* mat_inverse) (float (* mat)[3], float (* res)[3]);
};

typedef struct my_function function;		// function 이라는 구조체 자료형 선언

void mat_add(float (* mat1)[3], float (* mat2)[3], float (* res)[3])	// 행렬의 덧셈
{
	int i,j;

	for(i=0;i<3;i++)
	{
		for(j=0;j<3;j++)
		{
			res[i][j] = mat1[i][j] + mat2[i][j];
		}
	}
}

void mat_sub(float (* mat1)[3], float (* mat2)[3], float (* res)[3])	// 행렬의 뺄셈
{
	int i,j;

	for(i=0;i<3;i++)
	{
```

```
		for(j=0;j<3;j++)
		{
			res[i][j] = mat1[i][j] - mat2[i][j];
		}
	}
}

void mat_print(float (* res)[3])		// 행렬 출력
{
	int i,j;

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

void mat_multiply(float (* mat1)[3], float (* mat2)[3], float (* res)[3])	// 행렬의 곱셈
{
	int i,j;

	res[0][0] = mat1[0][0]*mat2[0][0] + mat1[0][1]*mat2[1][0] + mat1[0][2]*mat2[2][0];
	res[0][1] = mat1[0][0]*mat2[0][1] + mat1[0][1]*mat2[1][1] + mat1[0][2]*mat2[2][1];
	res[0][2] = mat1[0][0]*mat2[0][2] + mat1[0][1]*mat2[1][2] + mat1[0][2]*mat2[2][2];

	res[1][0] = mat1[1][0]*mat2[0][0] + mat1[1][1]*mat2[1][0] + mat1[1][2]*mat2[2][0];
	res[1][1] = mat1[1][0]*mat2[0][1] + mat1[1][1]*mat2[1][1] + mat1[1][2]*mat2[2][1];
	res[1][2] = mat1[1][0]*mat2[0][2] + mat1[1][1]*mat2[1][2] + mat1[1][2]*mat2[2][2];

	res[2][0] = mat1[2][0]*mat2[0][0] + mat1[2][1]*mat2[1][0] + mat1[2][2]*mat2[2][0];
	res[2][1] = mat1[2][0]*mat2[0][1] + mat1[2][1]*mat2[1][1] + mat1[2][2]*mat2[2][1];
	res[2][2] = mat1[2][0]*mat2[0][2] + mat1[2][1]*mat2[1][2] + mat1[2][2]*mat2[2][2];
}

float mat_det(float (* mat)[3]) // 판별식(determination) 구하기
{
	return mat[0][0]*(mat[1][1]*mat[2][2] - mat[1][2]*mat[2][1])
		+mat[0][1]*(mat[1][2]*mat[2][0] - mat[1][0]*mat[2][2])
		+mat[0][2]*(mat[1][0]*mat[2][1] - mat[1][1]*mat[2][0]);
}

void mat_adj(float (* mat)[3], float (* res)[3])	// 행렬의 adj 구하기
{
```

```
	res[0][0] = mat[1][1]*mat[2][2] - mat[1][2]*mat[2][1];
	res[0][1] = mat[1][2]*mat[2][0] - mat[1][0]*mat[2][2];
	res[0][2] = mat[1][0]*mat[2][1] - mat[1][1]*mat[2][0];

	res[1][0] = mat[0][2]*mat[2][1] - mat[0][1]*mat[2][2];
	res[1][1] = mat[0][0]*mat[2][2] - mat[0][2]*mat[2][0];
	res[1][2] = mat[0][1]*mat[2][0] - mat[0][0]*mat[2][1];

	res[2][0] = mat[0][1]*mat[1][2] - mat[0][2]*mat[1][1];
	res[2][1] = mat[0][2]*mat[1][0] - mat[0][0]*mat[1][2];
	res[2][2] = mat[0][0]*mat[1][1] - mat[0][1]*mat[1][0];
}

void mat_crammer(float (* mat)[3], float *num, float *sol)    // 크래머 공식
{
	float x[3][3];
	float y[3][3];
	float z[3][3];
	float det_A, det_x, det_y, det_z;

	x[0][0] = num[0];
	x[1][0] = num[1];
	x[2][0] = num[2];
	x[0][1] = mat[0][1];
	x[0][2] = mat[0][2];
	x[1][1] = mat[1][1];
	x[1][2] = mat[1][2];
	x[2][1] = mat[2][1];
	x[2][2] = mat[2][2];

	y[0][1] = num[0];
	y[1][1] = num[1];
	y[2][1] = num[2];
	y[0][0] = mat[0][0];
	y[1][0] = mat[1][0];
	y[2][0] = mat[2][0];
	y[0][2] = mat[0][2];
	y[1][2] = mat[1][2];
	y[2][2] = mat[2][2];

	z[0][2] = num[0];
	z[1][2] = num[1];
	z[2][2] = num[2];
	z[0][0] = mat[0][0];
	z[0][1] = mat[0][1];
	z[1][0] = mat[1][0];
	z[1][1] = mat[1][1];
	z[2][0] = mat[2][0];
```

```
	z[2][1] = mat[2][1];

	det_A = mat_det(mat);
	det_x = mat_det(x);
	det_y = mat_det(y);
	det_z = mat_det(z);

	sol[0] = det_x / det_A;
	sol[1] = det_y / det_A;
	sol[2] = det_z / det_A;
}

void mat_transport(float (* mat)[3], float (* res)[3])	// 행렬의 전치
{
	res[0][0] = mat[0][0];
	res[1][0] = mat[0][1];
	res[0][1] = mat[1][0];
	res[1][1] = mat[1][1];
	res[0][2] = mat[2][0];
	res[2][0] = mat[0][2];
	res[1][2] = mat[2][1];
	res[2][1] = mat[1][2];
	res[2][2] = mat[2][2];
}

void mat_inverse(float (* mat)[3], float (* res)[3])	// 역행렬 구하기
{
	int i,j;
	float det_A = mat_det(mat);

	mat_adj(mat, res);

	for(i=0;i<3;i++)
		for(j=0;j<3;j++)
			res[i][j] /= det_A;
}

#endif
```

2. c 파일

- 연산 함수를 main() 안에서 호출해서 연산 결과를 출력함

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

function func = {mat_print, mat_add, mat_sub, mat_multiply, mat_crammer, mat_inverse};

float mat1[3][3] = {{2,4,4},{6,2,2},{4,2,4}};
float mat2[3][3] = {{1,2,3},{1,2,3},{1,2,3}};
float res[3][3] = {{0,0,0},{0,0,0},{0,0,0}};
float num[3] = {12,16,20};
float sol[3] = {0};
float A[3][3] = {{2,4,8},{16,8,4},{2,2,2}};


int main(void)
{

	func.add(mat1, mat2, res);	// 행렬의 덧셈
	printf("행렬의 덧셈\n");
	func.print(res);

	func.sub(mat1, mat2, res);	// 행렬의 뺄셈
	printf("행렬의 뺄셈\n");
	func.print(res);

	func.multiply(mat1, mat2, res);	// 행렬의 곱셈
	printf("행렬의 곱셈\n");
	func.print(res);

	func.crammer(mat1, num, sol);	// 크래머 공식
	printf("크래머 공식\n");
	printf("%f %f %f\n\n", sol[0], sol[1], sol[2]); /* 크래머 공식에 의한 해는 sol[0], sol[1], sol[2] */
	mat_transport(mat1, res);	// 행렬의 전치
	printf("행렬의 전치\n");
	func.print(res);

	mat_inverse(A, res);	// 역행렬 구하기
	printf("역행렬 구하기\n");
	func.print(res);

	return 0;
}
```