TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 - 이상훈
gcccompil3r@gmail.com

학생 – 이우석
colre99@naver.com

[5/17(목) - 56일차]

[공업수학]

※ 벡터

벡터는 크기와 방향을 가진다
(방향과 크기가 있으면 벡터 , 방향이 없고 크기만 있다면 스칼라)

벡터의 연산방법은 x 축은 x 축 끼리 , y 축은 y 축 끼리, z 축은 z 축 끼리 연산.

벡터의 표기법은 (0,0) 위치에서 벡터의 포인트 지점으로, 같은 성분끼리 더한다.
ex) (Ax,Bx) (Ay,By) => (Ax+Ay, Bx+By) = (1,1) (2,1) => (3,2) x 는 x 끼리, y 는 y 끼리 더한다.

벡터의 곱셈 4 가지
1. 스칼라 곱셈
2. 내적
3. 외적
4. 텐서 연산 (주로 우주선 및 항공기 만들때 쓰임)

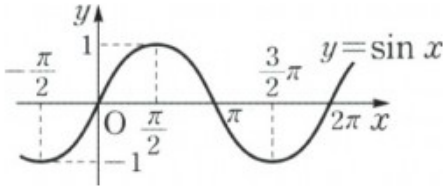
스케일링
ex) 2 배 스케일링 시 1x1 사이즈가 2 로 바꾸면 2x2 사이즈. 즉 1x1 보다 넓이가 4 배 증가

기저 시스템 (단위벡터를 포함하고 있다)
기저라고 부르기에 한가지 조건이 있다. → 축이 있음(각각의 단위벡터)
단위벡터는 크기가 1 인 벡터.

i hat(x 축), j hat(y 축), k hat(z 축)
=> sin x , cos x, tan x

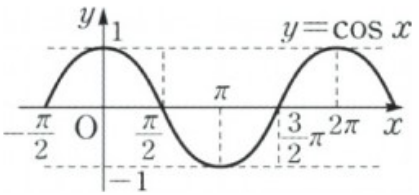
※ 삼각함수의 그래프

y=sin x (위(+) 아래(-) 파동 그래프) - 주기적분은 언제나 0.



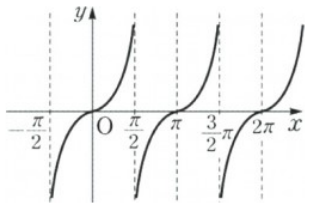
정의역: 실수 전체의 집합
치역 : {y | - 1 ≦ y ≦ 1}
주기 : 2π
대칭성 : 원점에 대하여 대칭

y=cos x 위 아래로 볼록 그래프



정의역: 실수 전체의 집합
치역 : {y | - 1 ≦ y ≦ 1}
주기 : 2π
대칭성 : y 에 대하여 대칭축

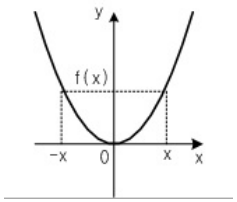
tan x = 떨어져있는 물결



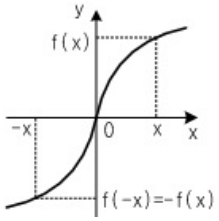
정의역: $x \neq \frac{\pi}{2} + n\pi$ (n 은 정수)인 실수 전체의 집합
치역 : 실수 전체의 집합
주기 : π
대칭성 : 원점에 대하여 대칭축

※ 우함수 & 기함수

우함수
: y =f(x)가 우함수 이면 모든 정의역의 원소 x 에 대하여 f(-x) = f(x)
=> y 축에 대하여 대칭은 그래프



기함수
: y =f(x)가 기함수 이면 모든 정의역의 원소 x 에 대하여 f(-x) = -f(x)
=> 원점에 대하여 대칭은 그래프



내적의 기하하적 의미는 결과가 스칼라(= 방향이 없다)

차량이나 드론을 만들때, 그램슈미트 정규 직교화 할때 내적 표기 필요.
double 이나 float 의 단점은 오차가 있다. 오차가 중첩된다면 연산이 달라짐

내적을 계산하는 새로운 패러다임 계산법
(= Orthogonal Projection)

Orthogonal Projection 을 이해해야 그램 슈미트를 이용하여 코딩할 수 있다.

-Gram-Schmidt Orthogonalization-
ex) 꼬인 좌표를 원상태로 되돌아오게 해주는 계산법
가장 연산하기 쉬운거를 기저로 잡고 계산하면 편리.

방향 코사인

오메가의 방향을 오메가의 크기로..
계산을 편하게 하기위해 내적으로 계산.

※ 그램 슈미트 정규 직교화 예제 1.

```
#include "vector_3d.h"
#include <stdio.h>

int main(void)
{
	vec3 A = {3, 2, 1};
	vec3 B = {1, 1, 1};
	vec3 X = {1, 0, 0};
	vec3 Y = {0, 1, 0};
	vec3 v[3] = {{0, 4, 0}, {2, 2, 1}, {1, 1, 1}};
	vec3 w[3] = {};
	vec3 R = {0, 0, 0,
			vec3_add, vec3_sub, vec3_scale,
			vec3_dot, vec3_cross, print_vec3,
			gramschmidt_normalization};

	R.add(A, B, &R);
	R.print(R);

	R.sub(A, B, &R);
	R.print(R);

	R.scale(3, R, &R);
```

```
	R.print(R);

	printf("A dot B = %f\n", R.dot(A, B));

	R.cross(X, Y, &R);
	R.print(R);

	R.gramschmidt(v, w, R);

	return 0;
}
```

위의 소스 헤더파일

```
#ifndef __VECTOR_3D_H__
#define __VECTOR_3D_H__

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

typedef struct vector3d vec3;

struct vector3d
{
	float x;
	float y;
	float z;
```

```
	void (* add)(vec3, vec3, vec3 *);
	void (* sub)(vec3, vec3, vec3 *);
	void (* scale)(float, vec3, vec3 *);
	float (* dot)(vec3, vec3);
	void (* cross)(vec3, vec3, vec3 *);
	void (* print)(vec3);

	void (* gramschmidt)(vec3 *, vec3 *, vec3);
};

void vec3_add(vec3 a, vec3 b, vec3 *r)
{
	r->x = a.x + b.x;
	r->y = a.y + b.y;
	r->z = a.z + b.z;
}

void vec3_sub(vec3 a, vec3 b, vec3 *r)
{
	r->x = a.x - b.x;
	r->y = a.y - b.y;
	r->z = a.z - b.z;
}

void vec3_scale(float factor, vec3 a, vec3 *r)
{
	r->x = a.x * factor;
	r->y = a.y * factor;
	r->z = a.z * factor;
}

float vec3_dot(vec3 a, vec3 b)
```

```
{
	return a.x * b.x + a.y * b.y + a.z * b.z;
}

void vec3_cross(vec3 a, vec3 b, vec3 *r)
{
	r->x = a.y * b.z - a.z * b.y;
	r->y = a.z * b.x - a.x * b.z;
	r->z = a.x * b.y - a.y * b.x;
}

void print_vec3(vec3 r)
{
	printf("x = %f, y = %f, z = %f\n", r.x, r.y, r.z);
}

float magnitude(vec3 v)
{
	return sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
}

void gramschmidt_normalization(vec3 *arr, vec3 *res, vec3 r)
{
	vec3 scale1 = {};
	float dot1, mag1;

	mag1 = magnitude(arr[0]);
	r.scale(1.0 / mag1, arr[0], &res[0]);
	r.print(res[0]);

	mag1 = magnitude(res[0]);
	dot1 = r.dot(arr[1], res[0]);
```

```
        r.scale(dot1 * (1.0 / pow(mag1, 2.0)), res[0], &scale1);
        r.sub(arr[1], scale1, &res[1]);
        r.print(res[1]);
}

#endif
```

※행렬

행렬은 숫자를 행(가로)와 열(세로) 로 배열한것.
(행의개수) x (열의 개수) 로 표기한 것을 '행렬의 크기'
행과 열의 개수가 같은 것을 정방행렬.

행렬의 종류에는
정방행렬, 전치행렬(transpose), 대칭행렬(symmetric) ↔부호반대(skew-symmetric),
upper triangular 행렬, lower triangular 행렬, 대각행렬(diagonal), 계수행렬, 확장행렬 등이 있다.

연립방정식

복잡한 연립방정식을 푸는데에는
가우스 조단 소거법 (Gaussian elimination)
크래머 공식(Cramer's Rule)
대치법(Substitution) 등 다양한 방법이 있다.

행렬에서 단위행렬을 I 라고 한다.

행렬안에 들어있는게 벡터 = 행렬은 벡터들의 집합

Determinant 는 행렬의 판별식
0 이 아니어야지 역형렬 ==> 이 계산은 주로 칼만필터에서 사용된다.(= 물리모델링을 위해서)

역행렬은 ad – bc
한번 끊어지면 – (마이너스), 두번 끊어지면 - - (마이너스 마이너스) 해서 + (플러스) 가 된다.

크래머 공식 (Cramer's Rule)

$$x_j = \begin{vmatrix} a_{11} & \cdots & a_{1j-1} & b_1 & a_{1j+1} & \cdots & a_{1n} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ a_{n+1} & \cdots & a_{n-1} & b_n & a_{n+1} & \cdots & a_n \end{vmatrix} \Big/ \begin{vmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_n \end{vmatrix}$$

※ 크래머 공식 기반 연립 방정식 풀기 예제 1.

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

void init_mat(float (*A)[3])
{
	int i, j;
```

```
	for(i = 0; i < 3; i++)
		for(j = 0; j < 3; 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("%10.4f", 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];
}

#if 0
A[0][0]	A[0][1]	A[0][2]		B[0][0]	B[0][1]	B[0][2]
A[1][0]	A[1][1]	A[1][2]		B[1][0]	B[1][1]	B[1][2]
A[2][0]	A[2][1]	A[2][2]		B[2][0]	B[2][1]	B[2][2]

A[0][0]*B[0][0]+A[0][1]*B[1][0]+A[0][2]*B[2][0]		A[0][0]*B[0][1]+A[0][1]*B[1][1]+A[0][2]*B[2][1]		A[0][0]*B[0]
[2]+A[0][1]*B[1][2]+A[0][2]*B[2][2]
A[1][0]*B[0][0]+A[1][1]*B[1][0]+A[1][2]*B[2][0]		A[1][0]*B[0][1]+A[1][1]*B[1][1]+A[1][2]*B[2][1]		A[1][0]*B[0]
[2]+A[1][1]*B[1][2]+A[1][2]*B[2][2]
A[2][0]*B[0][0]+A[2][1]*B[1][0]+A[2][2]*B[2][0]		A[2][0]*B[0][1]+A[2][1]*B[1][1]+A[2][2]*B[2][1]		A[2][0]*B[0]
[2]+A[2][1]*B[1][2]+A[2][2]*B[2][2]
#endif

void mul_mat(float (*A)[3], float (*B)[3], float (*R)[3])
{
	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];
}
```

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

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

	R[2][0] = A[0][1] * A[1][2] - A[0][2] * A[1][1];
	R[2][1] = A[0][2] * A[1][0] - A[0][0] * A[1][2];
#endif

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];
}

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__
	printf("Adjoint Matrix\n");
	print_mat(R);
#endif
	scale_mat(1.0 / det, R, R);

	return true;
}

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];
		}

		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);
#ifdef __DEBUG__
	print_mat(R);
#endif
	detX = det_mat(R);

	molding_mat(A, ans, 1, R);
#ifdef __DEBUG__
	print_mat(R);
#endif
	detY = det_mat(R);

	molding_mat(A, ans, 2, R);
#ifdef __DEBUG__
	print_mat(R);
#endif
	detZ = det_mat(R);

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

void print_vec3(float *vec)
{
	int i;
```

```
	for(i = 0; i < 3; i++)
		printf("%10.4f", vec[i]);

	printf("\n");
}

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);

	return 0;
}
```