C언어 - HW6

임베디드스쿨2기
Lv1과정
2021. 04. 19
차현호

벡터의 내적

-벡터의 내적이란?

벡터의 내적이란 점곱 또는 스칼라 곱이라고도 말하며 두벡터로 부터 실수를 얻을 수 있는 연산이다.
벡터의 내적을 구하는 방법은 다음과 같다.

A = {a1, a2, a3} B = {b1, b2, b3} 두벡터가 있을 때

A dot B = (a1 * b1)i, (a2 * b2)j, (a3 * b3)k

또는

A dot B = |A||B| cosΘ 로 표기한다.

여기서 중요한 부분이 cosΘ 인데

두벡터가 수직이면 cosΘ가 0이 되어 내적한 결과가 0 이된다

벡터의 내적

그러면 이러한 내적은 어디서 사용하는지 간단히 알아보자.

여러가지 분야에서 사용되겠지만 푸리에 변환에서 적용되는 부분을 확인해 보자.
확인하기에 앞서 두벡터가 수직한다는것은 서로 연관성이 없다는 걸로 이해를 하고 간다.

먼저 푸리에 변환을 하는 목적은 신호가 있을때 그신호속에 있는 주파수 성분들을
분석하기 위해 사용하는데 이때 시간영역에서 주파수 영역으로 시그널을 변환한다.

아래는 푸리에 변환의 식이다.

$$\mathrm{F}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

출처 : https://pinkwink.kr/198

벡터의 내적

아래의 식을 보면 f(t)함수와 e^(-iwt)가 함수의 내적인것을 확인 할 수있다.
e^(-iwt)는 오일러 공식에의해 다음과 같다.
e^(-iwt) = cos(wt) + isin(wt)

즉 우리가 분석하려는 시그널 f(t)와 cos, sin함수를 내적하여 내적한 값을 보고 주파수 성분을 찾아 낼 수있다.

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

출처 : https://pinkwink.kr/198

또한가지 중요한 사실은 서로 수직하는 두벡터 즉 내적한 값이 0이되는 두벡터를 선형 결합하면 2차원 평면의 모든 벡터의 표현이 가능하다.

벡터의 외적

-벡터의 외적이란?

벡터의 외적이란 벡터곱이라고 부르며 결과는 내적과 다르게 벡터가 된다.

벡터의 외적을 구하는 방법은 다음과 같다.

Vector A = (a1, a2, a3,) Vector B = (b1, b2, b3) 가 존재할때

A cross B = (a2*b3 – a3*b2, a3*b1 – a1*b3, a1*b2 – a2*b1)

또는

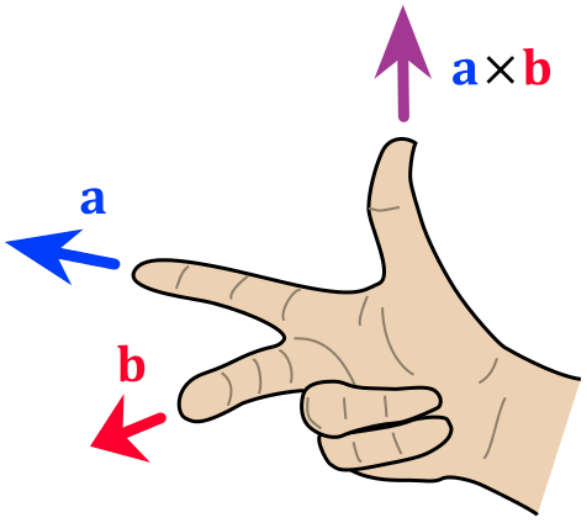
A croos B = |A||B|sinΘ n(법선 벡터)

벡터 A와 벡터 B의 외적한 결과는 A 와 B에 수직한 벡터가 된다.

벡터의 외적

-법선 벡터의 방향

벡터 A와 B의 외적한 벡터의 방향은 아래와 같이 오른손법칙을 이용하면 알 수 있다.



외적의 방향을 찾을 때 사용하는 오른손법칙 그림출처 : 위키백과

또한 두 벡터를 외적한 벡터는 두벡터에 대해 수직이므로 외적한 벡터랑 A, B 벡터를 내적하면 값이 0 이 나온다는것을 알수 있다.

벡터의 외적

앞에서 말한 가정이 맞는지 프로그램으로 테스트를 진행 하였다.

```
printf("vector A: \n");
print_vector(vectorA, len_A);
printf("vector B: \n");
print_vector(vectorB, len_B);

crossvector = crossproduct(vectorA, vectorB, len_A, len_B);
printf("A cross B : \n");
print_vector(crossvector, len_A);

result = dot_product(vectorA, crossvector, len_A, len_B);
printf("A dot corssvector : %f\n", result);

result = dot_product(vectorB, crossvector, len_A, len_B);
printf("B dot corssvector : %f\n", result);
```

```
vector A:
   8   7   1
vector B:
   4   6   6
A cross B :
  36 -44  20
A dot corssvector : 0.000000
B dot corssvector : 0.000000
```

벡터의 외적

-벡터 외적 C 코드

```c
int *crossproduct(int *vectorA, int *vectorB, int len_A, int len_B)
{
    int i;
    int *tmp = (int*)malloc(sizeof(int*) * VECTOR_SIZE);


    tmp[0] = vectorA[1] * vectorB[2] - vectorA[2] * vectorB[1];
    tmp[1] = vectorA[2] * vectorB[0] - vectorA[0] * vectorB[2];
    tmp[2] = vectorA[0] * vectorB[1] - vectorA[1] * vectorB[0];

    return tmp;
}
```

행렬의 곱셈

행렬의 곱셈 방법은 다음과 같다.

$$A=\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix},\ B=\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}\text{일때,}$$

$$AB=\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}=\begin{pmatrix} a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22} \\ a_{21}b_{11}+a_{22}b_{21} & a_{21}b_{12}+a_{22}b_{22} \end{pmatrix}$$

출처 : https://j1w2k3.tistory.com/575

위 식을 보면 A 행렬의 행 벡터와 B 행렬의 열벡터를 내적한 값이 행렬을 곱한 결과 행렬의 원소가 된다는것을 확인 할 수 있다.

한가지 중요한 부분은 앞의 A 행렬의 열과 B행렬의 행의 숫자가 일치해야 행렬의 곱셈이 가능 한다는 것이다.

또한 행렬의 곱셈은 AB = BA 같은 교환 법칙이 성립 되지 않는다.

행렬의 곱셈

-행렬 곱셈 C 코드

```c
int (* matrix_mul(int (*matrixA)[2], int (*matrixB)[2]))[2]
{
    int i,j;
    int (*tmp)[2] = (int (*)[2])malloc(sizeof(int) * 4);
    int vectorA[2] = { 0 };
    int vectorB[2] = { 0 };

    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 2; j++)
        {
            vectorA[0] = matrixA[i][0];
            vectorA[1] = matrixA[i][1];
            vectorB[0] = matrixB[0][j];
            vectorB[1] = matrixB[1][j];

            tmp[i][j] = dot_product(vectorA, vectorB, 2, 2);
        }
    }

    return tmp;
}
```

```
mat A:
   1   1
   1   1
mat B:
   3   3
   3   3
mat AB:
   6   6
   6   6
```

역행렬

-역행렬이란

어떤행렬 A가 존재할때 A와 행렬곱을 했을때 항등항렬 E가 나오는 행렬을 A의 역행렬이라고 정의한다.

항등행렬 E의 정의는 어떠한 행렬과 행렬곱을 진행하였을때 자기 자신이 나오는 행렬이며 다음과 같다.

$$\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \quad \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \quad \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

I_{2X2} I_{3X3} I_{4X4}

출처 : https://brunch.co.kr/@linecard/454

-역행렬 계산 방법

2x2의 역행렬 계산 방법은 아래와 같다.

$$B = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

$$B^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

출처 : http://physics2.mju.ac.kr/juhapruwp/?p=2035

-역행렬 계산 방법

3x3이상의 행렬에서의 역행렬 구하는 방법은 다음과 같고 크래머 공식을 이용해야한다.

<3×3 행렬의 역행렬>

$$A=\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix}$$ 일 때,

$$A^{-1}=\frac{1}{D}\begin{pmatrix} b_2c_3-b_3c_2 & b_2c_3-b_3c_2 & a_2b_3-a_3b_2 \\ b_3c_1-b_1c_3 & c_3a_1-c_1a_3 & a_3b_1-a_1b_3 \\ b_1c_2-b_2c_1 & c_1a_2-c_2a_1 & a_1b_2-a_2b_1 \end{pmatrix}$$

역행렬

-역행렬 c 코드

```c
int main(void)
{
    int matrixA[2][2] = { 0 };
    float det = 0;
    float matrixA_det[2][2] = { 0 };

    int i,j;

    srand(time(NULL));

    for(i = 0; i < 2; i++)
    {
        for(j = 0; j < 2; j++)
        {
            matrixA[i][j] = rand() % 10 + 1;
        }
    }

    det = (matrixA[0][0] * matrixA[1][1]) - (matrixA[0][1] * matrixA[1][0]);

    if (det == 0)
    {
        printf("can't find det \n");
        return -1;
    }

    matrixA_det[0][0] = matrixA[1][1] / det;
    matrixA_det[0][1] = (matrixA[0][1] * -1) / det;
    matrixA_det[1][0] = (matrixA[1][0] * -1) / det;
    matrixA_det[1][1] = matrixA[0][0] / det;


    printf("matrixA : \n");
    print_mat(matrixA);

    printf("matrixA_det : \n");
    print_mat_float(matrixA_det);

    return 0;

}
```

```
matrixA :
   6    3
   5    3
matrixA_det :
1.00   -1.00
-1.67   2.00
```

-구조체란?

C언어에서는 int, char 같은 기본 데이터 타입 이외에도 구조체를 이용하여 custom 자료형을 선언 할 수있다. 선언 방법과 사용법은 아래와 같다.

```c
#include <stdio.h>

struct my_type{
    int data1;
    int data2;
};

int main(void)
{
    struct my_type data;

    data.data1 = 1;
    data.data2 = 2;

    printf("data1 : %d \n", data.data1);
    printf("data2 : %d \n", data.data2);

    return 0;
}
```

```
data1 : 1
data2 : 2
```

구조체

구조체의 선언은 struct <이름> { 데이터 }; 이런식으로 선언하면 사용시에는
struct <이름> <변수면>; 으로 사용한다.

구조체안에는 여러 타입의 데이터가 존재할 수 있는데 이러한 데이터에 접근하려면
<변수명>.<구조체데이터> 이런식으로 사용해야한다.

-구조체 변수의 크기

앞에서 선언한 구조체 변수의 사이는 어떻게 될지 확인해보자 확인해보기에 앞서 int형 변수 2개가 담겨있으므로 크기는 8바이트가 될 것이라고 예상이 가능하다.

```c
#include <stdio.h>

struct my_type{
    int data1;
    int data2;
};

int main(void)
{
    struct my_type data;

    data.data1 = 1;
    data.data2 = 2;

    printf("data1 : %d \n", data.data1);
    printf("data2 : %d \n", data.data2);
    printf("my_type sie : %d \n", sizeof(data));

    return 0;
}
```

```
my_type size : 8
```

구조체

그렇다면 아래의 코드의 구조체변수 사이즈는 어떨까?
my_type2 구조체는 int형 1개, char형 1개를 선언해놓았다.
과연 크기는 4 + 1 = 5바이트가 되는지 확인해보면 놀랍게도 처음과 같은 8바이트가 나온 것을 확인 할 수 있다.

```c
#include <stdio.h>

struct my_type2{
    int data1;
    char data2;
};

struct my_type{
    int data1;
    int data2;
};

int main(void)
{
    struct my_type data;
    struct my_type2 data2;

    data.data1 = 1;
    data.data2 = 2;

    data2.data1 = 1;
    data2.data2 = 2;

    //printf("data1 : %d \n", data.data1);
    //printf("data2 : %d \n", data.data2);
    printf("my_type2 size : %d \n", sizeof(data2));

    return 0;
}
```

```
my_type2 size : 8
```

구조체

이전 슬라이드에 보았듯이 구조체의 크기는 왜 5바이트가 아닌 8바이트가 되었을까 이유를 알아보자.

구조체는 기본적으로 레지스터의 크기(현재 64bit)를 최소 크기로 사용한다.

즉 구조체 선언을 (char, char, char } 로 하면 1 + 1 + 1 = 3이 되어야 할꺼 같지만 그렇지 않고 최소크기인 8바이트가 할당된다.

앞에서 살펴본 my_type 구조체는 int형이 2개 4 + 4 = 8 이므로 8바이트가 된 것이다.

구조체 포인터

구조체도 역시 주소를 저장하는 포인터 변수를 선언할 수 있다.
기존의 구조체와 사용법은 비슷하며 선언시 *를 붙이는 것과 구조체 포인터 안의 데이터에 접근할 때는 . 대신에 ->를 사용한다.

```
struct my_type *data4;

data4 = &data;

data4->data1 = 1;
data4->data2 = 2;

printf("data1 : %d \n", data4->data1);
printf("data2 : %d \n", data4->data2);
```

-포인터 배열

포인터 배열은 말그대로 포인터변수들의 배열이다. 앞에서 배운 포인터 변수와 배열의 정의를 복습해보자

◆포인터 변수 : 특정한 타입의 메모리 주소를 저장할 수 있는 메모리 공간
◆배열 : 특정한 타입의 데이터를 연속된 메모리공간에 저장

즉 포인터 배열은 특정한 타입의 메모리 주소를 연속된 메모리공간에 저장한다는 것을 알 수 있다.

포인터배열과 배열 포인터

크기가 10인 int형 포인터 배열을 나타내면 아래 그림과 같다.

현재 사용하는 cpu는 64비트 cpu이므로 주소의 사이즈는 8바이트이다. 때문에 크기가 10인 int형 포인트 배열을 선언하면 배열의 크기는 80바이트가 된다.

8Byte

변수 주소	변수 주소	변수 주소	변수 주소	변수 주소	변수 주소	변수 주소	변수 주소	변수 주소	변수 주소

포인터배열과 배열 포인터

실제 크기가 80바이트인지 코드를 작성하여 확인해보았다.

```c
#include <stdio.h>

int main(void)
{
    int* p_array[10] = { 0 };

    printf("p_array size : %d \n", sizeof(p_array));

    return 0;
}
```

```
p_array size : 80
```

-배열 포인터

이전 슬라이드에서 본 포인터 배열과는 다르게 배열포인터는 특정 사이즈의 배열만 가리킬 수 있는 포인터 변수이다.

선언방법은 아래와 같다.

<데이터타입> (*변수명) [배열크기]

가령 int (*tmp) [2] 로 선언되었다면 크기가 2인 int형 데이터타입 배열을 가리키는 포인터 변수이다.

포인터배열과 배열 포인터

-배열 포인터와 포인터 배열의 포인터 연산 비교

1. 포인터 배열은 배열 변수명 + 숫자를하면 8바이트(64비트) 반큼 주솟값이 더해진다.
2. 배열포인터는 변수명 + 숫자를 하면 (변수의 사이즈 * 배열의 크기)만큼 주솟값이 더해진다.

이를 gdb를 이용하여 확인해보면 다음과 같다.
p_array1 + 1 은 8바이트가 더해진것을 확인 할 수 있고
p_array2 + 1 은 40바이트가 더해진것을 확인할 수 있다.

```c
#include <stdio.h>
#include <time.h>


int main(void)
{
    int *p_array1[10] = { 0 };
    int (*p_array2)[10];
    int array[10] = { 0 };

    p_array2 = array;

    printf("p_array1 = %x \n", p_array1);
    printf("p_array1 + 1 = %x \n", p_array1 + 1);

    printf("p_array2 = %x \n", p_array2);
    printf("p_array2 + 1 = %x \n", p_array2 + 1);


    return 0;
}
```

```
(gdb) x p_array1
0x7fffffffdf00: 0x00000000
(gdb) x p_array1 + 1
0x7fffffffdf08: 0x00000000
(gdb) x p_array2
0x7fffffffded0: 0x00000000
(gdb) x p_array2 + 1
0x7fffffffdef8: 0x00000000
```

질문

Q1. 구조체 크기와 관련하여 최소 8바이트 이후에는 4바이트씩 더해집니다.
가령
{int, char} -> 8바이트
{int, int, char} -> 12바이트 (예상 16바이트)
8바이트(64비트)씩 구조체 크기가 커질꺼 같은데 4바이트씩 올라가네요. 이건 컴파일과정에서 최적화를 해주기 때문인가요?