TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
56 일차 (2018. 05. 17)

목차

행렬이란?

수나 식을 직사각형 모양으로 배열한 것이다. 가로를 행, 세로를 열이라 하고 배열된 수나 식을 그 행렬의 성분이라고 한다.

정방행렬

정사각행렬. Square matrix. 열과 행의 개수가 같은 행렬을 말한다. 1 개의 수 a 도 1 행 1 열의 정방행렬로 볼 수 있다.

대각행렬

Diagonal matrix. 행렬에서 주대각 원소를 제외한 나머지 모든 원소가 0 인 정방행렬이다. 통계, 칼만 필터

단위행렬

특수한 대각행렬의 일종으로 곱셈에 대한 항등식이다. 단위행렬과 행렬을 곱하면 그 값은 곱해진 행렬이 나오는 것이다.

전치행렬

transposed matrix. 행렬의 행과 열을 서로 맞바꾼 행렬이다.

mXn의 크기를 갖는 행렬 $A=[a_{ij}]$에 대하여

nXm의 크기를 갖는 행렬 $A^T=[a_{ij}^T]$로서 $a_{ij}^T=a_{ji}$ 를 만족할 때

A^T를 행렬 A의 '전치행렬'이라고 한다.

$$A=\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} \Rightarrow A^T=\begin{bmatrix}1&4&7\\2&5&8\\3&6&9\end{bmatrix}$$

$$B=\begin{bmatrix}x&y\\z&w\end{bmatrix} \Rightarrow B^T=\begin{bmatrix}x&z\\y&w\end{bmatrix}$$

$$C=\begin{bmatrix}1&1&1&1\\-3&5&-2&7\end{bmatrix} \Rightarrow C^T=\begin{bmatrix}1&-3\\1&5\\1&-2\\1&7\end{bmatrix}$$

1. $(A^T)^T = A$
2. $(A+B)^T = A^T+B^T$
3. $(AB)^T = B^TA^T$
4. $(kA)^T = k\,A^T$ (k는 임의의 상수)

행렬의 덧셈

- 같은 위치에 있는

행렬의 곱셈

> 벡터와 마친가지로 스칼라 배가 가능하다. 즉, 특정값을 전체 성분에 곱해주는 것이 가능하다.

연립방정식
Guass-Jordan 소거법

역행렬
단위행렬을 이용한다

행렬의 판별식
3 차원 판별식
수반행렬
adjoint
크래머 공식

```
/*
 행렬의 덧셈, 뺄셈, 곱셈
역행렬 구하기(정식)
연립방정식
역행렬 구하기
크래머 공식
행렬의 전치
행렬의 판별식
*/
/*
#ifndef _matrix_3D_H__
#define _matrix_3D_H__
*/

/*되도록이면 for 문을 안 쓰는 것이 좋다. branch 가 들어가기 때문에
파이프라인이 깨지므로 성능이 안 좋다.*/
#include <stdio.h>
#include <stdlib.h>

//임의의 행렬
//랜덤으로 돌려주는 것이 더 좋은 것 같다.
int A[3][3] = {{1, 2, 3}, {2, 2, 4}, {0, 1, 0}};
int B[3][3] = {{2, 2, 2}, {1, 1, 0}, {1, 2, 1}};
int C[3][3] = {};
/*
```

```
void rand_arr(int A[3][3], int B[3][3])
{
  int i, j;

  for(i=0; i<3; i++)
   {
     for(j=0; j<3; j++)
       {
            A[3][3] = rand()%;
       }
    }
}

// for 문 안 돌리는 방법이....
*/

//행렬의 덧셈
void add(int A[3][3], int B[3][3])
{
  int i, j;

  for(i = 0; i < 3; i++)
   {
    for(j = 0; j < 3; j++)
     {
        C[i][j]  = A[i][j]+B[i][j];
        printf("%d₩t", C[i][j]);
     }
      printf("₩n");
    }
}

//행렬의 뺄셈
void sub(int A[3][3], int B[3][3])
{
int i, j;

  for(i = 0; i < 3; i++)
   {
    for(j = 0; j < 3; j++)
     {
        C[i][j]  = A[i][j]-B[i][j];
```

```
        printf("%d₩t", C[i][j]);
     }
      printf("₩n");
    }

}

//행렬의 곱셈
void mul(int A[3][3], int B[3][3])
{
  int i, j, k;
  int sum = 0;

    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
        {
            sum = 0;
            for(k=0; k<3; k++)
            {
              sum += A[i][j]*B[j][k];
            }
            C[i][k] = sum;
              printf("%d₩t", C[i][k]);
        }
        printf("₩n");
    }

}

//전치행렬
int transposed(int (*A)[3])
{
  int i, j;
  int D[3][3];

 for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
        {
              D[i][j] = A[j][i];
            printf("%d₩t", D[i][j]);
```

```
        }
        printf("₩n");
    }
  return A[3][3];
}

//역행렬(정석)
void inverse(int A[3][3])
{

}


int main(void)
{
   printf("행렬의 덧셈₩n");
   add(A, B);

   printf("행렬의 뺄셈₩n");
   sub(A, B);

   printf("행렬의 곱셈₩n");
   mul(A, B);

   printf("A 의 전치행렬₩n");
   transposed(A);
   printf("B 의 전치행렬₩n");
   transposed(B);

   return 0;
}


/*
#endif */

선생님꺼
#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");
}

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("%10.4f", R[i][j]);
        printf("₩n");
    }
    printf("₩n");
}

void adjust_3x4_mat(float (*A)[4], int idx, float (*R)[4])
{
        int i, j;
        float div_factor;

        for(i = idx + 1; i < 3; i++)
        {
                //div_factor = -A[idx][idx] / A[idx + 1][idx];
                //div_factor = -A[idx + 1][idx] / A[idx][idx];
                //div_factor = -A[i][0] / A[idx][0];
                div_factor = -A[i][idx] / A[idx][idx];
                printf("div_factor = %f₩n", div_factor);

                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);
#if __DEBUG__
	print_3x4_mat(R);
#endif

	adjust_3x4_mat(R, 0, R);
#if __DEBUG__
	print_3x4_mat(R);
#endif

	adjust_3x4_mat(R, 1, R);
#if __DEBUG__
	print_3x4_mat(R);
#endif

	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[idx][idx] / A[idx + 1][idx];
        //div_factor = -A[idx + 1][idx] / A[idx][idx];
        //div_factor = -A[i][0] / A[idx][0];
        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);
#if __DEBUG__
```

```
        print_3x6_mat(mid);
#endif

        adjust_3x6_mat(mid, 0, mid);
#if __DEBUG__
    print_3x6_mat(mid);
#endif

        adjust_3x6_mat(mid, 1, mid);
#if __DEBUG__
    print_3x6_mat(mid);
#endif
}

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;
}
```