TI DSP, MCU 및 Xilinx Zynq FPGA
프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 하성용
accept0108@naver.com

56 일차

정방행렬 // 행과 열의 숫자가같다

단위행렬// 대각선을 제외하고 다 1

전치행렬 // 대각선 대칭이동

어떤행렬을 곱해서
$A \times A^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$(I (단위행렬을 대문자 I))

$A \times A^{-1} = I$
B=R
A|R
x
y
z
→ 가우스 조르단 소거법

$A|I \Rightarrow AA^{-1}|IA^{-1} \Rightarrow i|a^{-1}$
→ $I|A^{-1}$

2 0 4 → 2 로 나누기 → 1 0 2
0 3 9
0 0 1

위에 1 0 2 에 2 제거필요
-2 를 2 밑에 밑에 1 에 곱해서 더하기

--행렬은 벡터들의 집합

determinant → 행렬의 판별식
역행렬이 존재하지않을조건

연습문제
$\begin{pmatrix} 2 & 4 & 8 \\ 16 & 8 & 4 \\ 2 & 2 & 2 \end{pmatrix} A^{-1} = ?$

컴퓨터는 가우스 소거법이 편하고
사람은 크래머공식이 편하다

다음 연립방정식을 크래머공식으로 풀이

2x + 4y + 4z =12
6x + 2y + 2z = 16
4x + 2y + 4z = 20

$A = \begin{pmatrix} 2 & 4 & 4 \\ 6 & 2 & 2 \\ 4 & 2 & 4 \end{pmatrix}$ $X=\begin{pmatrix} 12 & 4 & 4 \\ 16 & 2 & 2 \\ 20 & 2 & 4 \end{pmatrix}$ $Y=\begin{pmatrix} 2 & 12 & 4 \\ 6 & 16 & 2 \\ 4 & 20 & 4 \end{pmatrix}$ $Z=\begin{pmatrix} 2 & 4 & 12 \\ 6 & 2 & 16 \\ 4 & 2 & 20 \end{pmatrix}$

8 -64 16
$\det(A) = 2(2\cdot4-2\cdot2)+4(2\cdot4-6\cdot4)+4(6\cdot2-2\cdot4)$
$=-40$

```
                    48          -96              -32
det(X)    = 12(2·4-2·2)+4(2·20-16·4)+4(16·2-2·20)
          = -80

                det(X)    -80
          ∴ x =--------- =  ----- = 2
                 det(A)    -40

           2·24=48   12(-16)=-192    4(56)=224
det(Y)=2(16·4-2·20)+12(2·4-6·4)+4(6·20-16·4)
      =80

                det(Y)    80
          ∴ y =--------- = ----- = -2
                 det(A)    -40

           2(8)=16      4(-56)=-224    12(4)=48
det(Z)=2(2·20-16·2)+4(16·4-6·20)+12(6·2-2·4)
      =-160

                det(Z)   -160
          ∴ z =--------- = ----- = 4
                 det(A)    -40
```

프로그램 만들기
행렬의 덧셈, 뺄셈, 곱셈
역행렬 구하기(정석)
연립방정식(가우스 소거법)
역행렬 구하기(가우스 소거법) //반 구현
크래머 공식
행렬의 전치
행렬의 판별식
행렬 스케일링

```
//
#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] = {};
        float scale;
```

```
	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);
}

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);

	return 0;
}
```

```
Init A Matrix
    3.0000    0.0000    3.0000
    1.0000    0.0000    1.0000
    0.0000    1.0000    1.0000

Init B Matrix
    0.0000    2.0000    0.0000
    0.0000    3.0000    3.0000
    3.0000    1.0000    1.0000

A + B Matrix
    3.0000    2.0000    3.0000
    1.0000    3.0000    4.0000
    3.0000    2.0000    2.0000

A - B Matrix
    3.0000   -2.0000    3.0000
    1.0000   -3.0000   -2.0000
   -3.0000    0.0000    0.0000

Matrix Scale(A)
    1.5000    0.0000    1.5000
    0.5000    0.0000    0.5000
    0.0000    0.5000    0.5000

AB Matrix
    9.0000    9.0000    3.0000
    3.0000    3.0000    1.0000
    3.0000    4.0000    4.0000

det(A) = 0.000000
det(B) = 18.000000

A^T(Transpose) Matrix
    3.0000    1.0000    0.0000
    0.0000    0.0000    1.0000
    3.0000    1.0000    1.0000

B^T(Transpose) Matrix
    0.0000    0.0000    3.0000
    2.0000    3.0000    1.0000
    0.0000    3.0000    1.0000

A Inverse Matrix
역행렬 없다!
test Inverse Matrix
    0.5000    0.0000   -2.0000
    0.0000    0.3333   -3.0000
    0.0000    0.0000    1.0000

크래머 공식 기반 연립 방정식 풀기!
2x + 4y + 4z = 12
6x + 2y + 2z = 16
4x + 2y + 4z = 20
    2.0000   -2.0000    4.0000
가우스 소거법 기반 연립 방정식 풀기!(문제 위의 것과 동일함)
div_factor = -3.000000
div_factor = -2.000000
div_factor = -0.600000
    2.0000   -2.0000    4.0000
```