TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
58 일차 (2018. 05. 21)

목차

- 미분
- 코드화
- 행렬 코드화 진행중....

미분 방정식 Differential Equation
미분 또는 도함수를 포함한 방정식
계수(order)와 차수(degree)
미분 방정식에 나타난 가장 높은 미분 횟수가 미분 방정식의 계수가 된다. 차수는 계수가 가장 높은 도함수의 거듭제곱수를 말한다.
$(y'')^3 + 8y = 0$ 일 때, 이는 2 계 3 차 미분방정식이 된다.
편미분방정식
상미분방정식과 다르게 두 개 이상의 독립변수에 대한 도함수(편미분)를 포함하는 방정식을 편미분방정식이라고 한다. 편미분방정식의 형태에서 풀고자 하는 함수의 편도함수들과 함수의 차수가 모두 1차로 나타나면 선형 편미분방정식이라 하고, 그렇지 않은 경우 비선형 편미분방정식이라고 한다. 선형방정식에 비해 비선형방정식의 해를 구하는 것이 어렵지만, 실제 현상에 대한 모델링을 할 경우 선형방정식보다는 비선형방정식으로 표현되는 경우가 많아서 비선형방정식의 해를 구하는 것은 중요하다.

- 변수 분리형 미분방정식
  상미분방정식의 일종이다. 변수 분리형 미분방정식에서 종속변수는 y, 독립 변수는 x 이고 먼저종속변수를 좌변으로 독립변수를 우변으로 '분리'한다.
  -종속변수(dependent variable) : 서로 관계가 있는 둘 이상의 변수가 있을 경우, 어느 한 쪽의 영향을 받아 변하는 변수
  -독립변수(independent variable) : 함수 y 가 y=f(x)꼴로 주어질 때, 변수 x⇒함수를 이루는 변수
- 완전 미분형 미분방정식
- 일계 선형 미분방정식
- 이계 미분방정식

```
#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....    */
//전치행렬 = 행렬에서 행과 열을 바꾼 행렬 = 대각선을 기준으로 대칭 한 행렬
/* int trans (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 D[3][3];
}*/
//for 문 대신.. 대각선 기준으로 대칭
void trans (int A[3][3], int D[3][3])
{
        D[0][0] = A[0][0];   //대각선 기준
        D[1][1] = A[1][1];
        D[2][2] = A[2][2];

        D[0][1] = A[1][0];
        D[1][0] = A[0][1];

        D[0][2] = A[2][0];
        D[2][0] = A[0][2];

        D[2][1] = A[1][2];
        D[1][2] = A[2][1];
  printf("%d₩t %d₩t %d₩t", D[0][0], D[0][1], D[0][2]");
  printf("₩n");
  printf("%d₩t %d₩t %d₩t", D[1][0], D[1][1], D[1][2]");
  printf("₩n");
```

```
    printf("%d₩t %d₩t %d₩t", D[2][0], D[2][1], D[2][2]");
    printf("₩n");
}

//행렬의 판별식 = 역행렬의 유무를 판별
void det(int A[3][3])
{
  return   A[0][0]*(A[1][1]*A[2][2] – A[1][2]*A[2][1]) – A[0][1](A[1][0]*A[2][2] - A[1][2]*A[2][0]) +
A[0][2](A[1][0]*A[2][1] – A[1][1]A[2][0]);
}

//수반행렬 = A 행렬의 여인수 행렬의 전치행렬
void adj (float A[3][3], float D[3][3])
{
            D[0][0] = A[1][1] * A[2][2] - A[1][2] * A[2][1];
            D[0][1] = A[0][2] * A[2][1] - A[0][1] * A[2][2];
            D[0][2] = A[0][1] * A[1][2] - A[0][2] * A[1][1];

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

            D[2][0] = A[1][0] * A[2][1] - A[1][1] * A[2][0];
            D[2][1] = A[0][1] * A[2][0] - A[0][0] * A[2][1];
            D[2][2] = A[0][0] * A[1][1] - A[0][1] * A[1][0];
}
// 수반행렬 구하는 법
```

$$A = \begin{pmatrix} 1 & 2 & 5 \\ 2 & 3 & 7 \\ 1 & 5 & 6 \end{pmatrix}$$

$$adjA = \begin{pmatrix} +\begin{vmatrix} 3 & 7 \\ 5 & 6 \end{vmatrix} & -\begin{vmatrix} 2 & 7 \\ 1 & 6 \end{vmatrix} & +\begin{vmatrix} 2 & 3 \\ 1 & 5 \end{vmatrix} \\ -\begin{vmatrix} 2 & 5 \\ 5 & 6 \end{vmatrix} & +\begin{vmatrix} 1 & 5 \\ 1 & 6 \end{vmatrix} & -\begin{vmatrix} 1 & 2 \\ 1 & 5 \end{vmatrix} \\ +\begin{vmatrix} 2 & 5 \\ 3 & 7 \end{vmatrix} & -\begin{vmatrix} 1 & 5 \\ 2 & 7 \end{vmatrix} & +\begin{vmatrix} 1 & 2 \\ 2 & 3 \end{vmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} -17 & -5 & 7 \\ 13 & 1 & -3 \\ -1 & 3 & 1 \end{pmatrix}^T = \begin{pmatrix} -17 & 13 & -1 \\ -5 & 1 & 3 \\ 7 & -3 & 1 \end{pmatrix}$$

```
//스칼라 곱
void scale_mat(float scale, float A[3][3], float D[3][3])
{
        D[0][0] = A[0][0]* scale;
        D[0][1] = A[0][1]* scale;
        D[0][2] = A[0][2]* scale;

        D[1][0] = A[1][0]* scale;
        D[1][1] = A[1][1]* scale;
        D[1][2] = A[1][2]* scale;

        D[2][0] = A[2][0]* scale;
        D[2][1] = A[2][1]* scale;
        D[2][2] = A[2][2]* scale;
}

//역행렬(정석)
void inverse(float A[3][3], float D[3][3])
{
    float y;
    y = det(A);

  if( y == 0)
        printf("역행렬이 존재하지 않는다.");
  else
  {
  adj(A, D);
  scale_mat(1/y, D, D);
  printf("%d₩t %d₩t %d₩t", D[0][0], D[0][1], D[0][2]");
  printf("₩n");
  printf("%d₩t %d₩t %d₩t", D[1][0], D[1][1], D[1][2]");
  printf("₩n");
  printf("%d₩t %d₩t %d₩t", D[2][0], D[2][1], D[2][2]");
  printf("₩n");
  }
}

int main(void)
{
```

```
    printf("A 의 전치행렬₩n");
    transposed(A);
    printf("B 의 전치행렬₩n");
    transposed(B);
    printf("A 의 역행렬₩n");
    inverse(A, D);
    printf("B 의 역행렬₩n");
    inverse(B, D);

    return 0;
}
```