TI DSP, MCU, Xilinx Zynq FPGA
프로그래밍 전문가 과정

강사 – Innova Lee (이상훈)
gcccompil3r@gmail.com
학생 – 김형주
mihaelkel@naver.com

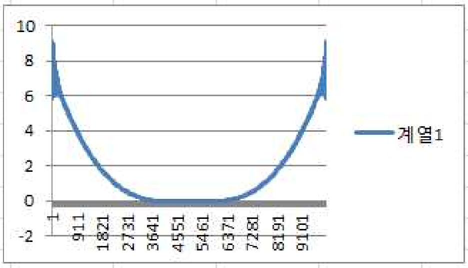
미분방정식 구현하기

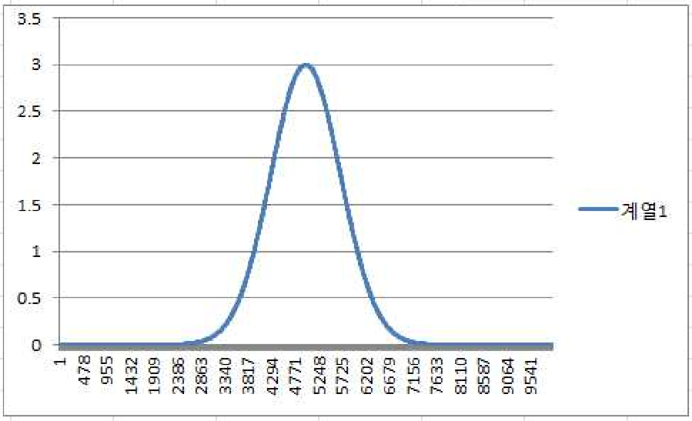
```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#define delta_x    0.001
#define start    -5.000
#define end        5.000
#define y(x)    y[(long)((x-start)/delta_x)]
#define init_val(a,b)    y(a)=b
double e;
void set_exp_val(void);
double f_x(double x);
double y_prime_equation(double x,double y);
void diff_equation(double (*p)(double x,double y), double* y);
void set_accuracy(double* y,double (*p)(double),double* accuracy);
void write_txt_data(int fd, double* y, double (*p)(double x));
int main(void)
{
    double x = 0;
    double y[10000];
    double accuracy[10000];
    int i;
    /*e = 2.713xxxxxxxxxx*/
    set_exp_val();
    /*y(0) = 3*/
    init_val(0,3);

    /*set y[0] ~ y[9999]*/
    diff_equation(y_prime_equation, y);
    /*set accuracy between y and f_x*/
    set_accuracy(y,f_x,accuracy);

    for(i=0;i<10000;i++)
        printf("y(%lf) = %.12lf, %.12lf, 오차율 : %.3lf%%\n",delta_x*i-5, y[i],f_x(delta_x*i - 5),accuracy[i]*100);
    int fd;
    fd = open("data.txt",O_CREAT|O_TRUNC| O_RDWR,0644);
    /*record the data to analyze that with excel chart*/
    write_txt_data(fd,y,f_x);
}
//e = (1+dx)^(1/dx) = (1 + 0.00000000001)^100000000000 which is about 2.713xxxxxxxx
void set_exp_val(void)
{
    e = pow(1+0.000000000001,1/0.000000000001);
}
//the origin result of differential equation, 3e^(-x^2)
double f_x(double x)
{
    return 3*pow(e,-pow(x,2));
}
//physics modeling, y'=-2xy. return value must be y'
double y_prime_equation(double x,double y)
```

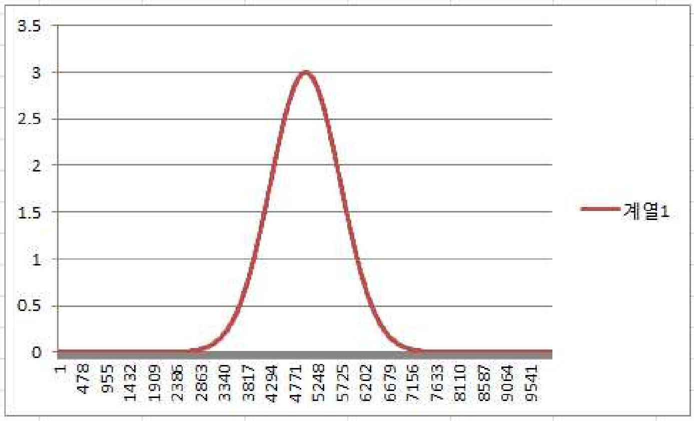
```c
{
    return -2*x*y;
}
void diff_equation(double (*p)(double x,double y), double* y)
{
    int i;
    double x = 0;
    /*x = -5 ~ 0*/
    for(i = 4999 ;i>=0;i--){
        x = delta_x*i - 5;
        y[i]= -p(x+delta_x,y[i+1])*delta_x+y[i+1];
    }
    /*x = 0 ~ 5*/
    for(i=5001;i<10000;i++){
        x = delta_x*i - 5;
        y[i] = p(x-delta_x,y[i-1])*delta_x+y[i-1];
    }
}
void set_accuracy(double* y,double(*p)(double),double* accuracy)
{
    int i;
    for(i=0;i<10000;i++)
    {
        accuracy[i] = (y[i] - p(delta_x*i - 5))/p(delta_x*i - 5);
        /*accuracy value must be a positive number*/
        if(accuracy[i] < 0)
            accuracy[i] *= (-1);
    }
}
void write_txt_data(int fd, double* y, double (*p)(double x))
{
    int i;
    double x, tmp;
    char buf[64];
    for(i=0;i<10000;i++)
    {
        x = delta_x*i - 5;
        tmp = p(x);
        sprintf(buf,"%d\t%.12lf\t%.12lf\n",i,y[i],tmp);
        buf[strlen(buf)] = '\0';
        write(fd,buf,strlen(buf));

    }
}

```

```
y(4.980000) = 0.000000000047, 0.000000000051, 오차율 : 7.276%
y(4.981000) = 0.000000000047, 0.000000000050, 오차율 : 7.280%
y(4.982000) = 0.000000000046, 0.000000000050, 오차율 : 7.285%
y(4.983000) = 0.000000000046, 0.000000000049, 오차율 : 7.289%
y(4.984000) = 0.000000000045, 0.000000000049, 오차율 : 7.294%
y(4.985000) = 0.000000000045, 0.000000000048, 오차율 : 7.298%
y(4.986000) = 0.000000000044, 0.000000000048, 오차율 : 7.302%
y(4.987000) = 0.000000000044, 0.000000000047, 오차율 : 7.307%
y(4.988000) = 0.000000000043, 0.000000000047, 오차율 : 7.311%
y(4.989000) = 0.000000000043, 0.000000000046, 오차율 : 7.316%
y(4.990000) = 0.000000000043, 0.000000000046, 오차율 : 7.320%
y(4.991000) = 0.000000000042, 0.000000000045, 오차율 : 7.325%
y(4.992000) = 0.000000000042, 0.000000000045, 오차율 : 7.329%
y(4.993000) = 0.000000000041, 0.000000000045, 오차율 : 7.334%
y(4.994000) = 0.000000000041, 0.000000000044, 오차율 : 7.338%
y(4.995000) = 0.000000000040, 0.000000000044, 오차율 : 7.343%
y(4.996000) = 0.000000000040, 0.000000000043, 오차율 : 7.347%
y(4.997000) = 0.000000000040, 0.000000000043, 오차율 : 7.352%
y(4.998000) = 0.000000000039, 0.000000000042, 오차율 : 7.356%
y(4.999000) = 0.000000000039, 0.000000000042, 오차율 : 7.361%
```

엑셀을 통한 데이터 분석







상단의 차트는 오차율로, 미분을 거듭할수록 오차가 커짐을 알 수 있다. 5000번 연산시, 오차율이 8%정도로 나타난다. 이는 sampling period가 너무 큰 결과로, sampling period와 오차는 반비례한다. 즉,

$$_{ampling} \propto \quad rror\, Rate$$

왼쪽 하단의 그래프는 $' = -2xy$ 를 통한 y의 수치 해석결과이고,

오른쪽 하단의 그래프는 $y = 3e^{\quad x}$ 의 그래프이다 (-5 < x < 5)

미분 알고리즘

일반적인 미분의 정의는 아래와 같다.

$$f'(x) = \lim_{\triangle x \to 0} \frac{f(x+\triangle x) - f(x)}{(x+\triangle x) - x}$$

이를 프로그래밍한다고 하면, $\triangle x$의 값에 제약이 생긴다.

만약 $\triangle x \to 0$을 프로그래밍한다고 하면, 필요한 데이터양이 무한대가 될 것이다.

(sampling period가 0이므로, samping frequency가 무한대가 되어야 한다)

즉, 프로그래밍으로 구현할 때에는 극한값 대신 샘플링 주기를 넣으면 된다.

$$f'(t) = \frac{f(t+T_{sampling}) - f(t)}{_{sampling}}$$

이런식으로 구현하면, T값에 따라 오차가 발생하지만 근사적으로 미분을 구현할 수 있다.

다만, 데이터값은 배열에 저장하여야 하는데, 배열의 인덱스는 integer type이여야 하므로,

$$f'[i] = (f[i+1] - f[i]) / T;$$

와 같은 로직이 필요하다.

또한, 미분방정식 $y' = -2xy$가 주어졌으므로, 이를 프로그래밍으로 구현하면,

$$f'[i] = -2xy[i]$$

이고, x 는 실제 시간이고 i는 샘플링 간격이므로

$$x = i * T_{sampling}$$

이 된다. 또한, x의 범위가 $-5 < x < 5$ 이므로, when i=0, x = -5 가 성립해야 되므로

$$x = i * T_{sampling} - 5$$

가 최종적으로 된다. 이 x값을 위의 미분방정식에 대입하면,

$'[i] = -2(i* \quad _{ampling} - 5)y[i]$

가 된다. 미분의 정의에서

$$f'[i] = (f[i+1] - f[i])/T;$$

이므로 위의 두 식을 연립하면

$$-2(i*T_{sampling} - 5)y[i] = (y[i+1] - y[i])/T_{sampling}$$

이 된다. f[i]항은 2개, f[i+1]항은 1개이므로, f[i+1]항에 대해서 정리하면,

$$y[i+1] = 2(i*T_{sampling} - 5)T_{sampling}y[i] + y[i]$$

이 되고, y'=-2xy[i]를 y'=p(x,y[i])로 일반화하면

$$y[i+1] = -p(i*T_{sampling} - 5, y[i])T_{sampling}y[i] + y[i];$$

이 된다.

이는 y[i]와 y[i+1]의 관계식, 즉, 일종의 수열의 형태를 가지고 있으므로, 초기값 부분부터 순차대로 계산해야 한다.

(초기값이 없으면 수열을 전개할 수 없다)

초기값은 y(0)=3 이므로, y[5000] = 3이 된다.

즉, i = 5000부터 증감계산을 해야 하므로,

```
for(i=5001;i<10000;i++){
        x = delta_x*i - 5;
        y[i] = p(x-delta_x,y[i-1])*delta_x+y[i-1];
}
```

x > 0 구간의 경우 위와 같은 형태의 코드가 완성된다.

x < 0 구간의 경우, y[5000]이 주어졌으므로, y[5000]을 이용하여 y[4999]를

또, y[4999]를 이용하여 y[4998]을 계산하는 i-- 형태의 for 구문이 필요하다.

즉, 좌항의 index가 우항의 index보다 1만큼 작은 형태가 되야 하므로, 위의 식을 다시 전개해야한다.

```
for(i = 4999 ;i>=0;i--){
        x = delta_x*i - 5;
        y[i]= -p(x+delta_x,y[i+1])*delta_x+y[i+1];
}
```