2019 국가암호공모전 Ⅱ 분야 답안 제출 양식

소속 : 서울시립대학교 **대표자 이름 : 방수민**

문제 05

답)

(1) 근사다항식

$$p(x) = 0.5 + 0.24999853x - 0.0208145944x^3 + 0.00204715244x^5 - 0.00018566257x^7 + 0.00001293558x^9 - 0.0000005492x^{11} + 0.00000001001x^{13}$$

(2) (3) 코드제출

문제 분석 :

1. 주어진 함수 $f(x) = \dfrac{e^x}{1+e^x}$는 모든 x에 대해 $0 \leqq f(x) \leqq 1$이다.

2. 추가 자료 파일을 통해 암호문에는 level이라는 파라미터가 있다는 사실을 알 수 있다. 동형곱셈, 동형상수곱셈(상수가 정수가 아닌 경우)의 경우에는 level 1을 소모하며, 초기 level의 값은 10이다.

즉, 동형곱셈과 동형상수곱셈을 총 10번만 사용할 수 있다는 뜻으로 해석가능하다. 따라서 동형상수곱셈의 사용 횟수를 최소화하여야 근사다항식에서 더 많은 차수의 항을 사용할 수 있다.

그러므로 근사다항식을 계산할 때, 동형상수곱셈을 한 번만 사용하는 방법을 사용한다. 예를 들어, $y = 0.5 + 0.1x - 0.01x^2 + 0.05x^3$을 계산한다고 하자.

일반적인 방법을 사용하면 $0.1x,\ -0.01x^2,\ 0.05x^3$에서 각각 동형상수곱셈을 사용해 총 3번을 사용해야 하지만 식을 변형하여 $y = 0.5 + 0.01(10x - x^2 + 5x^3)$라 생각하면 마지막에 0.01을 곱하는 것으로 동형상수곱셈을 1번만 사용하여 계산할 수 있다.

3. 동형상수곱셈을 계산하는 과정에서 상수의 크기가 소수점 11자리 이하로는 결과 값에 오차가 크게 발생하였다. 따라서 근사다항식의 계수는 소수점 12번째 자리에서 반올림하여 11자리까지 사용하였다.

풀이 :

1. 근사다항식을 얻은 방법론

1-1. 테일러급수

근사다항식을 찾는 가장 기본적인 방식인 테일러급수를 이용하였다.

테일러 급수를 다음 식으로 나타낸다고 할 때,

$$f(x)=\sum_{k=0}^{n}\frac{f^{(k)}(a)}{k!}(x-a)^k+R_{n+1}(x)$$

마지막 항인 $R_{n+1}(x)$을 f의 **나머지 항** 또는 **절단오차**라 하는데, [a, x] 또는 [x, a]에 속하는 적당한 실수 b에 대해 다음과 같이 쓸 수 있다.[1]

- $R_{n+1}(x)=\dfrac{f^{(n+1)}(b)}{(n+1)!}(x-a)^{n+1}.$

출처 : 위키백과

<그림 1. 테일러급수의 오차>

위 그림에서 나타나 있는 테일러급수의 오차공식을 사용하여, $[0,x]$ 또는 $[x,0]$에 속하는 적당한 실수 b에 대해 $|f^{(n+1)}(b)|\leqq M$ 을 만족하는 양수 M이 존재한다면 *remainder of order n*의 절댓값의 상계는 $|R_{n+1}(x)|\leqq M\dfrac{|x|^{n+1}}{(n+1)!}$이 된다.

식을 만족하는 n을 추정하기 위해 $|f(x)|\leqq 1$임을 고려해 $M\leqq 1$이라 가정하자. 그러면

$$\begin{aligned}|R_{n+1}(x)| &\leqq M\frac{|x|^{n+1}}{(n+1)!}\\ &\leqq \frac{|x|^{n+1}}{(n+1)!}\\ &\leqq \frac{4^{n+1}}{(n+1)!}\quad (\because -4\leqq x\leqq 4)\end{aligned}$$

이 되고, 오차가 0.001이하가 되어야하기 때문에

$$\frac{4^{n+1}}{(n+1)!}\leqq 0.001\ \Rightarrow\ 14\leqq n$$

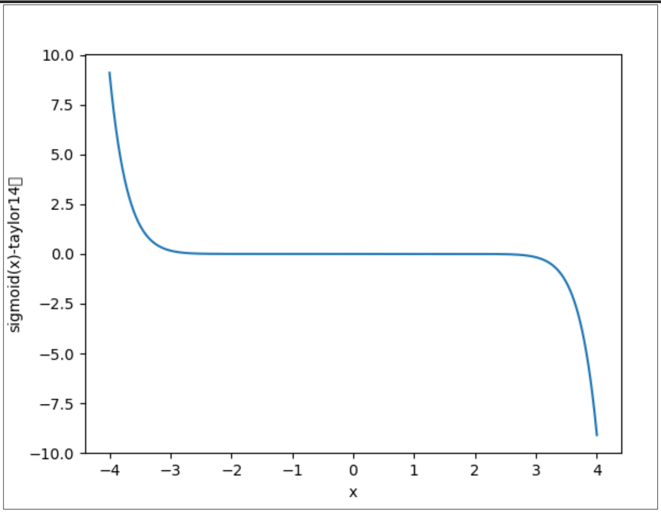
이 된다. 따라서 $14\leqq n\leqq 31$에 대해 n차 테일러급수를 구하여 오차가 0.001이하가 되는지 확인해보았다. 하지만 모든 $14\leqq n\leqq 31$에 대해 오차가 0.001이하가 되는 테일러급수는 존재하지 않았다. $n=14,31$인 경우만 설명을 덧붙인다.

① $n=14$

14차 테일러급수를 구하면 다음과 같다.

$$\text{taylor14}(x)=\frac{1}{248}+\frac{x}{4}-\frac{x^3}{48}+\frac{x^5}{480}-\frac{17x^7}{80640}+\frac{31x^9}{1451520}-\frac{691x^{11}}{319334400}+\frac{5461x^{13}}{24908083200}.$$

이 함수와 $Sigmoid$함수와의 차이를 그래프로 그리면,



<그림 2. 14차 테일러급수와 sigmoid함수의 오차 그래프>

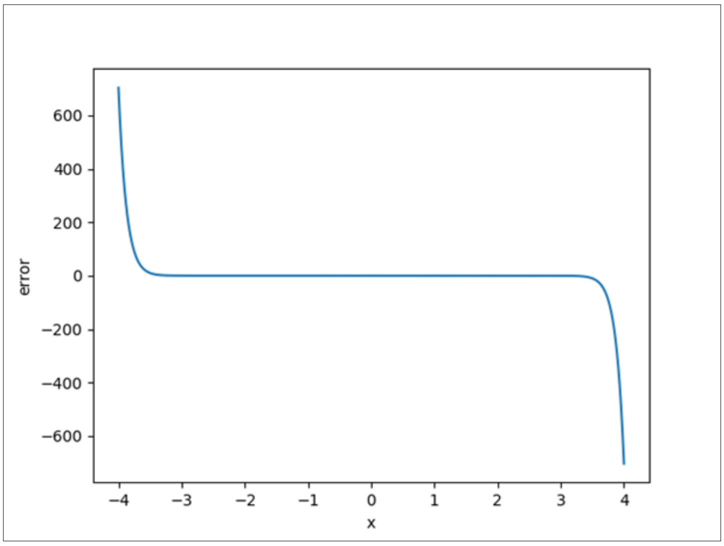
$-4 \leqq x \leqq 4$에서 오차가 0.001이하가 아님을 확인 할 수 있다.

② $n = 31$

마찬가지로 테일러급수를 구하면 다음과 같다.

$$\text{taylor31}(x) = \frac{1}{248} + \frac{x}{4} - \frac{x^3}{48} + \frac{x^5}{480} - \frac{17x^7}{80640} + \frac{31x^9}{1451520} - \frac{691x^{11}}{319334400} + \frac{5461x^{13}}{24908083200} - \frac{929569x^{15}}{41845579776000} + \frac{3202291x^{17}}{1422749712384000} - \frac{221930581x^{19}}{973160803270656000} + \frac{4722116521x^{21}}{204363768686837760000} - \frac{56963745931x^{23}}{2433130987189174272000 0} + \frac{14717667114151x^{25}}{620448401733239439360000 00} - \frac{2093660879252671x^{27}}{871109556033468172861440 00000} + \frac{8612567256320118 1x^{29}}{353670479749588078181744640000 00} - \frac{1298481636811073019 53x^{31}}{526261673867387060334436024320000000}$$

이 함수와 $Sigmoid$함수와의 차이를 그래프로 그리면,



<그림 3. 31차 테일러급수와 sigmoid함수의 오차 그래프>

역시 범위 내에서 오차가 0.001이하가 아님을 알 수 있다. 또한, $|x| \geqq 3$에 대해 $n=14$보다 $n=31$일 때, 오차가 더 커짐을 알 수 있었다. 따라서 $|f^{(n+1)}(b)| \leqq M$ 을 만족하는 양수 M은 1이하가 아니라 더 커질 것이라 생각한다.

정리하면 31차 이내의 테일러급수를 이용해 $[-4,\ 4]$에서 오차가 0.001이하인 근사다항식을 찾을 수 없고, 테일러급수의 짝수차 항 계수가 모두 0이라는 사실을 통해 $f(x)$의 근사다항식에서 짝수차 항은 필요하지 않을 수 있음을 시사한다.

1-2. $Sigmoid$함수와 그래프 형태가 비슷한 함수

$\arctan$함수와 같이 $Sigmoid$함수와 그래프 형태가 비슷한 함수를 이용하여 근사다항식을 찾는 방법을 생각해보았지만, 그래프 형태가 비슷하더라도 그 함수를 다시 테일러급수로 근사시키는 과정에서 오차가 1-1의 테일러급수보다 더 커지게 된다. 따라서 $Sigmoid$함수가 아닌 그래프 형태가 비슷한 함수를 이용하는 것은 적절하지 않다고 판단하였다.

1-3. 함수가 지나는 점을 이용해 계수 찾기

$p(x)=a_0+a_1x+a_2x^2+\ \cdots\ +a_{31}x^{31}$라 하고, 이때 찾아야 하는 미지수가 32개이므로 주어진 $f(x)$가 지나는 32개의 점 $(x_0,\,y_0),(x_1,\,y_1),(x_2,\,y_2),\ \cdots\ ,(x_{31},\,y_{31})$을 이용해 연립방정식을 풀어 계수를 찾는 방법이다.

① 1-1에서 n의 값이 커질수록 오차가 증가한다는 점에서 31차 항까지 사용하는 것보다는 가능한 낮은 차수의 항들을 이용하는 것이 더 옳을 것이라 판단했다.

② 문제 분석에서 언급한 level이라는 파라미터 때문에 동형곱셈을 9번만 사용할 수 있다 (동형상수곱셈을 1번 사용해야하기 때문에 10번 중 9번만 사용할 수 있다). 1차부터 동형곱셈을 9번 사용하여 순서대로 계산하여 만들 수 있는 최대 차수는 아래와 같이 10차이다.

1. $x \bullet x = x^2$
2. $x \bullet x^2 = x^3$
3. $x \bullet x^3 = x^4$
4. $x \bullet x^4 = x^5$
5. $x \bullet x^5 = x^6$
6. $x \bullet x^6 = x^7$
7. $x \bullet x^7 = x^8$
8. $x \bullet x^8 = x^9$
9. $x \bullet x^9 = x^{10}$

이제 $p10(x)=a_0+a_1x+a_2x^2+\ \cdots\ +a_{10}x^{10}$라 하고, $f(x)$가 지나는 점을 $x=-4$부터 $x=4$까지 똑같은 간격으로 11개의 점

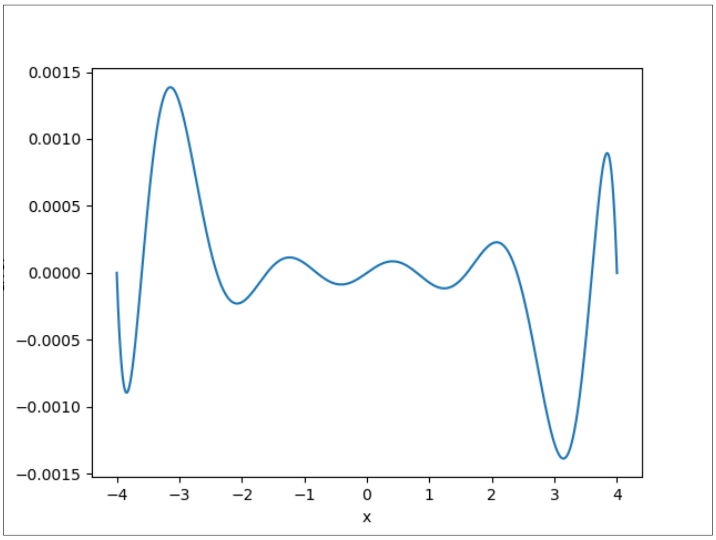
$$x=-4,\ -3.2,\ -2.4,\ -1.6,\ -0.8,\ 0,\ 0.8,\ 1.6,\ 2.4,\ 3.2,\ 4$$

을 선택해 연립방정식을 세워 $a_0,\ a_1,\ \cdots\ ,\ a_{10}$을 계산하였다. 그 결과 아래와 같다.

```
[[ 5.00000000e-01]
 [ 2.49705005e-01]
 [ 2.77555756e-15]
 [-2.01288237e-02]
 [ 1.11022302e-16]
 [ 1.62959544e-03]
 [-9.71445147e-17]
 [-8.48964955e-05]
 [ 8.67361738e-18]
 [ 1.88322804e-06]
 [-2.71050543e-19]]
```

<그림 4. 위에서부터 $a_0, a_1, \cdots, a_{10}$의 값>

문제 분석에서 언급한대로 계수의 소수점은 12번째 자리에서 반올림하여 사용하였다. 그러면 a_2, a_4, a_6, a_8의 경우 소수점에서 처음 0이 아닌 수가 나오는 자리가 소수점 11자리 이상이기 때문에 근사식에 사용하지 않는다. 따라서 상수항과 홀수차 항만을 이용해 $[-4, 4]$에서 $f(x)$와의 오차를 계산한다. 그때 $3.5 \leq |x| \leq 4$인 x에 대해서 오차가 0.001을 초과하였다.



<그림 5. $p10(x)$와 sigmoid함수의 오차 그래프 >

그래서 11개의 점 중에서 -3.2와 3.2를 오차가 큰 범위 안인 -3.7과 3.7으로 변경하여

$$x = -4,\ -3.7,\ -2.4,\ -1.6,\ -0.8,\ 0,\ 0.8,\ 1.6,\ 2.4,\ 3.7,\ 4$$

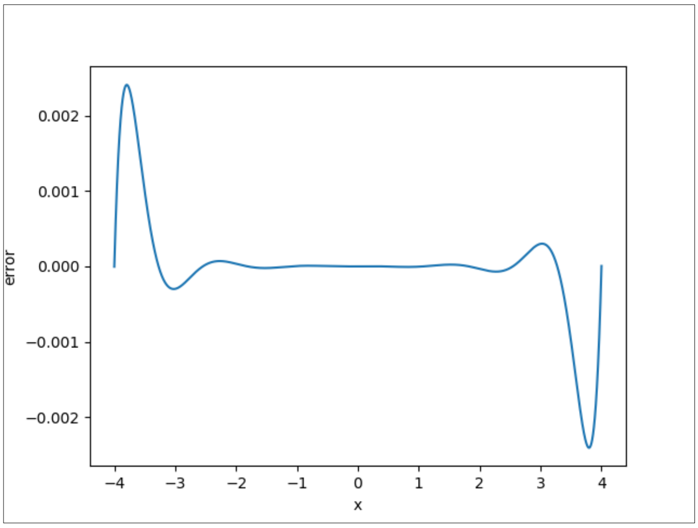
으로 수정하여 다시 계산하였지만, 이 역시 오차가 0.001을 초과하였다. 비슷하게 좌표를 옮겨가며 계산해보았지만 최고차항이 10차인 다항식에서는 근사다항식을 찾을 수 없었다.

③ 1-1에서 짝수차 항이 필요하지 않을 수 있다는 시사점과 ②에서 짝수차 항의 계수가 굉장히 작다는 것을 고려해 홀수차 항만 사용하기로 하였다. 동형곱셈 9번으로 홀수차 항만 만들어내면 최대 차수는 다음과 같이 17차이다.

1. $x \bullet x = x^2$
2. $x \bullet x^2 = x^3$
3. $x^2 \bullet x^3 = x^5$
4. $x^2 \bullet x^5 = x^7$
5. $x^2 \bullet x^7 = x^9$
6. $x^2 \bullet x^9 = x^{11}$
7. $x^2 \bullet x^{11} = x^{13}$
8. $x^2 \bullet x^{13} = x^{15}$
9. $x^2 \bullet x^{15} = x^{17}$

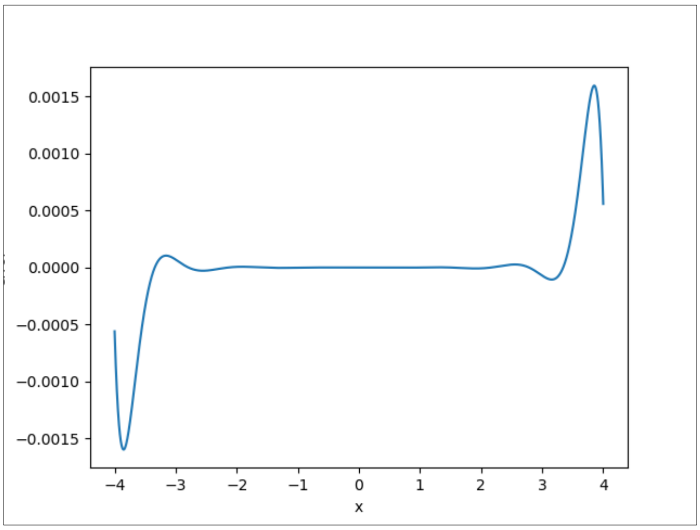
②에서 10차 항까지 사용하였으므로 먼저 11차 다항식 $p11(x)$, 13차 다항식 $p13(x)$, 15차 다항식 $p15(x)$, 17차 다항식 $p17(x)$으로 근사다항식을 찾아보았다. 이때, 홀수차 항만을 있는 근사다항식을 이용하지만 근사다항식의 계수를 찾을 때는 짝수차 항을 함께 찾는 것이 오차가 더 작았다.

예를 들어, 11차 다항식의 경우 ②에서 사용한 방법과 비슷하게 $p11(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{11}x^{11}$라고 하고, $f(x)$가 지나는 점을 $x = -4$부터 $x = 4$까지 똑같은 간격으로 12개의 점을 이용한다. 그리고 나온 계수들을 이용해 짝수차 항 계수인 $a_2, a_4, \cdots, a_{10}$을 0이라 두고 홀수차 항의 계수는 소수점 12번째에서 반올림하여 오차를 계산하면 다음과 같다.

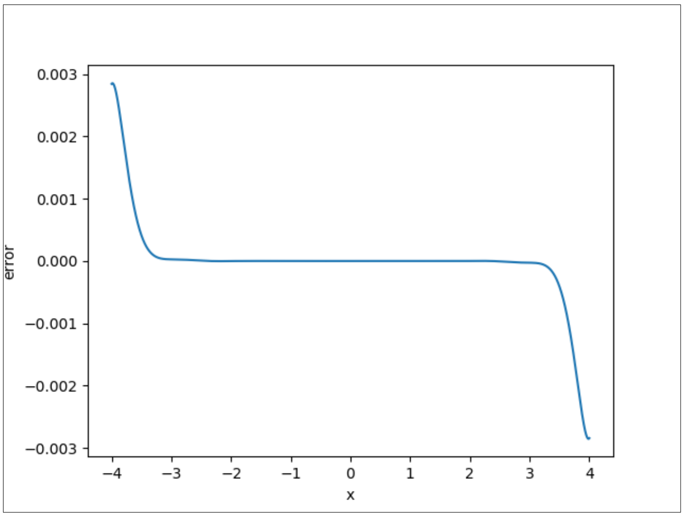


<그림 6. $p11(x)$와 sigmoid함수의 오차 그래프 >

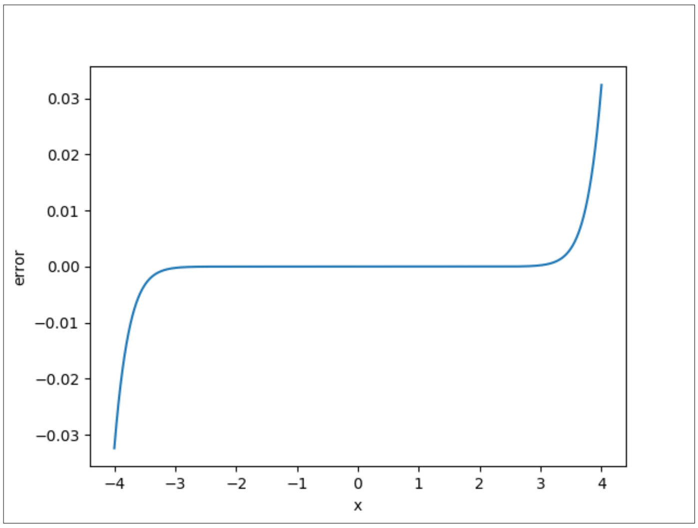
같은 방법으로 13차 다항식, 15차 다항식, 17차 다항식으로 근사다항식을 찾고 오차 그래프를 그려보면 다음과 같다.



<그림 7. $p13(x)$와 sigmoid함수의 오차 그래프 >



<그림 8. $p15(x)$와 sigmoid함수의 오차 그래프>



<그림 9. $p17(x)$와 sigmoid함수의 오차 그래프>

각각의 오차 그래프를 봤을 때, 13차 다항식 <그림 7>을 이용한 경우 오차가 가장 작게 나타났다. 따라서 13차 다항식을 더 세밀하게 이용해 근사다항식을 찾아보기로 하였다.

④ 13차 다항식을 찾을 때 $[-4,4]$범위에서 똑같은 간격으로 점 14개

$$\begin{aligned} x =& -4,\ -3.38461538,\ -2.76923077,\ -2.15384615,\ -1.53846154, \\ & -0.92307692,\ -0.30769231,\ 0.30769231,\ 0.92307692,\ 1.53846154, \\ & 2.15384615,\ 2.76923077,\ 3.38461538,\ 4 \end{aligned}$$

을 이용하였다. 하지만 <그림 7>을 보면, 오차가 $|x|=3.8$ 부근에서 크게 나타남을 알 수 있다. 그래서 $x=-3.38461538, 3.38461538$을 $x=-3.651, 3.651$ (3.3과 4사이의 임의의 수로 정하였다)으로 변경하여 계수를 구해보았다.

그 결과 다음과 같은 값을 구할 수 있었다.

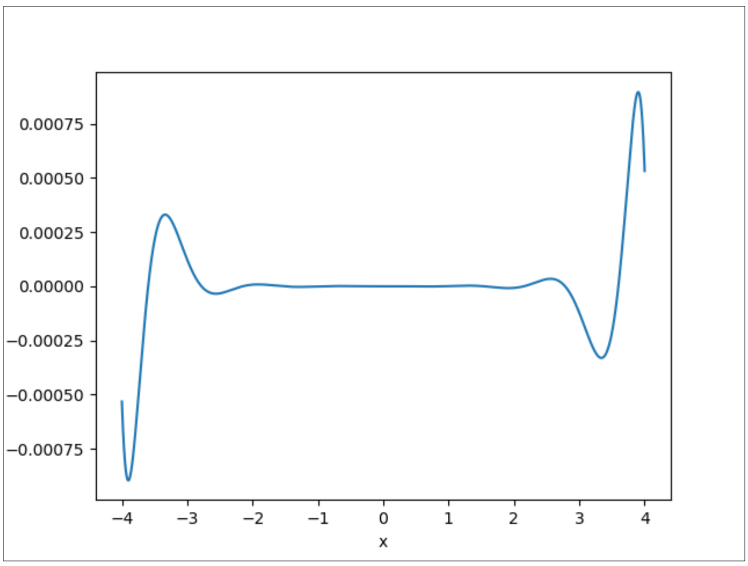
```
[[ 5.00000000e-01]
 [ 2.49998530e-01]
 [ 7.61612995e-14]
 [-2.08145944e-02]
 [-5.10702591e-15]
 [ 2.04715244e-03]
 [ 1.38777878e-17]
 [-1.85662572e-04]
 [ 0.00000000e+00]
 [ 1.29355815e-05]
 [-2.16840434e-19]
 [-5.49203198e-07]
 [ 0.00000000e+00]
 [ 1.00181295e-08]]
```

<그림 10. 위에서부터 순서대로 상수항, 1차, ... 13차항의 계수>

이를 계수로 사용한 근사다항식

$$\begin{aligned} p(x) =& 0.5+0.24999853x-0.0208145944x^3+0.00204715244x^5-0.00018566257x^7 \\ & +0.00001293558x^9-0.0000005492x^{11}+0.00000001001x^{13} \end{aligned}$$

와 $Sigmoid$함수의 오차 그래프를 그려보니 오차가 0.001이하가 될 것으로 예상되었고 오차를 확인하였다.



<그림 11. $p(x)$와 sigmoid함수의 오차 그래프>

2. 오차 확인

1-3 ④에서 찾은 근사다항식 $p(x)$의 오차가 0.001이하인지 확인하자.
먼저 오차를 $e(x) = |Sigmoid(x) - p(x)|$라 하자.

2-1. x좌표를 나누어 확인

<그림 11>을 보면, $0 \leqq |x| \leqq 3.5$에서의 오차는 0.001이하인 것이 명확하다. 그러므로 $3.5 \leqq |x| \leqq 4$에서 오차만을 확인하였다.

$[-4, -3.5]$를 1,000,000등분하여 각 x에 대해 $e(x) = |Sigmoid(x) - p(x)|$의 값을 계산하여 최댓값을 찾으면

$$x = -3.90142090142090014\text{에서 } e(x) \cong 0.000896909619349489 7 < 0.001$$

이다.

마찬가지로 $[3.5, 4]$를 1,000,000등분하여 각 x에 대해 $e(x) = |Sigmoid(x) - p(x)|$의 값을 계산하여 최댓값을 찾으면

$$x = 3.90142090142090014\text{에서 } e(x) \cong 0.0008969096193495174 < 0.001$$

이다.

2-2. Wolfram Alpha를 이용하여 확인

$$\min\left\{\frac{e^x}{1+e^x} - (0.5 + 0.24999853000\,x - 0.02081459440\,x^3 + 0.00204715244\,x^5 - 0.00018566257\,x^7 + 0.00001293558\,x^9 - 5.492\times10^{-7}\,x^{11} + 1.001\times10^{-8}\,x^{13}) \,\middle|\, -4 \le x \le 4\right\} \approx -0.00089691 \text{ at } x \approx -3.90142$$

$$\max\left\{\frac{e^x}{1+e^x} - (0.5 + 0.24999853000\,x - 0.02081459440\,x^3 + 0.00204715244\,x^5 - 0.00018566257\,x^7 + 0.00001293558\,x^9 - 5.492\times10^{-7}\,x^{11} + 1.001\times10^{-8}\,x^{13}) \,\middle|\, -4 \le x \le 4\right\} \approx 0.00089691 \text{ at } x \approx 3.90142$$

출처 : Wolfram Alpha

2가지 방법으로 오차를 확인하였을 때, $|e(x)|$의 최댓값은 약 0.0008969이였다.

3. 구현

3-1. Problem2

① Encrypt code

```
// Step 1 : Encrypt x

// your code //
encryptor.Encrypt(x,ctxt_in);

///////////////
```

<그림 12. Problem2의 Encrypt 부분>

주어진 코드에서 암호화를 위한 객체 encryptor 안 함수 Encrypt를 사용하여 x를 암호화해서 ctxt_in에 저장한다.

② sigmoid code

```
// Step 2 : Evaluate Sigmoid
sigmoid(ctxt_in, ctxt_out, evaluator);
```

```
void sigmoid(Ciphertext& ctxt_in, Ciphertext& ctxt_out, HomEvaluator& eval)
{
    // your code //
    Ciphertext ctxt2,ctxt3,ctxt5,ctxt7,ctxt9,ctxt11,ctxt13;
    Ciphertext mulctxt1,mulctxt3,mulctxt5,mulctxt7,mulctxt9,mulctxt11,mulctxt13;
    Ciphertext add1,add2,add3,add4,add5,add6,addctxt,result;

    eval.Square(ctxt_in,ctxt2);
    eval.Mult(ctxt_in,ctxt2,ctxt3);
    eval.Mult(ctxt2,ctxt3,ctxt5);
    eval.Mult(ctxt2,ctxt5,ctxt7);
    eval.Mult(ctxt2,ctxt7,ctxt9);
    eval.Mult(ctxt2,ctxt9,ctxt11);
    eval.Mult(ctxt2,ctxt11,ctxt13);

    eval.Mult(ctxt_in,(long)24999853000,mulctxt1);
    eval.Mult(ctxt3,(long)-2081459440,mulctxt3);
    eval.Mult(ctxt5,(long)204715244,mulctxt5);
    eval.Mult(ctxt7,(long)-18566257,mulctxt7);
    eval.Mult(ctxt9,(long)1293558,mulctxt9);
    eval.Mult(ctxt11,(long)-54920,mulctxt11);
    eval.Mult(ctxt13,(long)1001,mulctxt13);

    eval.Add(mulctxt1,mulctxt3,add1);
    eval.Add(mulctxt5,mulctxt7,add2);
    eval.Add(mulctxt9,mulctxt11,add3);
    eval.Add(add1,add2,add5);
    eval.Add(add3,mulctxt13,add6);
    eval.Add(add5,add6,addctxt);

    eval.Mult(addctxt,0.00000000001,result);
    eval.Add(result,0.5,ctxt_out);

    ///////////////
}
```

<그림 13. Problem2의 sigmoid 부분>

ctxt_in를 다항식의 x라 하면, 동형연산을 위한 객체 evaluator에 있는 곱셈함수들 Square와 Mult를 이용해 다항식 계산에 필요한 ctxt2:=x^2, ctxt3:=x^3, ctxt5:=x^5, ⋯ ,ctxt13:=x^{13}을 계산한다. 그리고 동형상수곱셈을 최소화하기 위해

$$p(x) = 0.00000000001(24999853000x - 2081459440x^3 + 204715244x^5 - 18566257x^7 + 1293558x^9 - 5492x^{11} + 1001x^{13}) + 0.5$$

으로 상수항을 제외한 항들을 0.00000000001으로 묶어 계산하였다. 결과 값은 ctxt_out에 저장된다.

③ Decrypt code

```
// Step 3 : Decrypt
double y;

// your code //
decryptor.Decrypt(ctxt_out, y);

///////////////
```

<그림 14. Problem2의 Decrypt 부분>

주어진 코드에서 복호화를 위한 객체 decryptor 안 함수 Decrypt를 사용하여 ctxt_out를 복호화해서 y에 저장한다.

3-2. Problem3

① Encrypt code

```
vector<double> x(128);
for(size_t i = 0; i < 128; ++i) {
    x[i] = (8. * ((double)rand() / RAND_MAX)) - 4.;
}
Ciphertext ctxt_in, ctxt_out;

// Step 1 : Encrypt x

// your code //
Message mvec(128);
for(size_t i = 0; i < 128; ++i) {
    mvec[i] = x[i];
}
encryptor.Encrypt(mvec,ctxt_in);
///////////////
```

<그림 15. Problem3의 Encrypt 부분>

암호화를 위해 x에 만들어진 128개의 수를 Message mvec(128)에 다시 저장한다. 성분을 각각 암호화하여, 그 값을 ctxt_in에 저장한다.

② sigmoid code

Problem2와 동일하다.

③ Decrypt code

```
// Step 3 : Decrypt
Message y(128);

// your code //
decryptor.Decrypt(ctxt_out,y);
///////////////
```

<그림 16. Problem3의 Decrypt 부분>

결과 값이 저장되어 온 ctxt_out의 성분을 각각 복호화 해 Message y(128)에 저장한다.

4. 실행결과

$0.001 \cong 2^{-10}$이므로 $\log_2(|f(x)-y|) < -10$이면 문제에서 원하는 조건에 만족한다.

4-1. Problem2

```
problem@cryptocontest:~/Documents/Problem2$ ./run
x = 2.89357
f(x) = 0.947528
y = 0.947778
log2(|f(x)-y|) = -11.9651
```

<그림 17. Problem2 파일 실행 결과>

구현한 Problem2를 한번 실행한 결과이다. 결과 값이 조건을 만족하였다.

4-2. Problem3

```
problem@cryptocontest:~/Documents/Problem3$ ./run
log2(|f(x)-y|) = -12.593
log2(|f(x)-y|) = -13.6468
log2(|f(x)-y|) = -14.8771
log2(|f(x)-y|) = -12.8637
log2(|f(x)-y|) = -10.5225
log2(|f(x)-y|) = -15.1915
log2(|f(x)-y|) = -13.9456
log2(|f(x)-y|) = -10.6007
log2(|f(x)-y|) = -11.8478
log2(|f(x)-y|) = -11.337
log2(|f(x)-y|) = -12.8437
log2(|f(x)-y|) = -13.9395
log2(|f(x)-y|) = -10.7621
log2(|f(x)-y|) = -15.0955
log2(|f(x)-y|) = -20.5936
log2(|f(x)-y|) = -11.0119
log2(|f(x)-y|) = -12.7039
log2(|f(x)-y|) = -12.4918
log2(|f(x)-y|) = -11.3373
log2(|f(x)-y|) = -12.3937
log2(|f(x)-y|) = -10.5304
log2(|f(x)-y|) = -13.179
log2(|f(x)-y|) = -12.5813
log2(|f(x)-y|) = -12.7018
log2(|f(x)-y|) = -12.9061
log2(|f(x)-y|) = -13.1613
log2(|f(x)-y|) = -12.9482
log2(|f(x)-y|) = -12.5482
log2(|f(x)-y|) = -13.509
log2(|f(x)-y|) = -12.6263
log2(|f(x)-y|) = -10.5213
log2(|f(x)-y|) = -12.6217
log2(|f(x)-y|) = -11.5225
log2(|f(x)-y|) = -14.4252
log2(|f(x)-y|) = -15.9553
log2(|f(x)-y|) = -11.2489
log2(|f(x)-y|) = -10.76
log2(|f(x)-y|) = -11.7323
log2(|f(x)-y|) = -11.4426
log2(|f(x)-y|) = -12.3788
log2(|f(x)-y|) = -11.2827
log2(|f(x)-y|) = -14.1591
log2(|f(x)-y|) = -12.629
log2(|f(x)-y|) = -14.0585
log2(|f(x)-y|) = -11.4573
```

```
log2(|f(x)-y|) = -12.4643
log2(|f(x)-y|) = -11.1734
log2(|f(x)-y|) = -12.9144
log2(|f(x)-y|) = -13.4962
log2(|f(x)-y|) = -11.1963
log2(|f(x)-y|) = -16.1664
log2(|f(x)-y|) = -13.6906
log2(|f(x)-y|) = -12.478
log2(|f(x)-y|) = -12.5793
log2(|f(x)-y|) = -12.6354
log2(|f(x)-y|) = -10.9486
log2(|f(x)-y|) = -13.1222
log2(|f(x)-y|) = -13.2874
log2(|f(x)-y|) = -13.033
log2(|f(x)-y|) = -17.2747
log2(|f(x)-y|) = -10.8982
log2(|f(x)-y|) = -12.9356
log2(|f(x)-y|) = -12.4627
log2(|f(x)-y|) = -12.5402
log2(|f(x)-y|) = -12.5089
log2(|f(x)-y|) = -12.4611
log2(|f(x)-y|) = -12.8869
log2(|f(x)-y|) = -12.4611
log2(|f(x)-y|) = -12.6996
log2(|f(x)-y|) = -14.3058
log2(|f(x)-y|) = -13.4755
log2(|f(x)-y|) = -12.4634
log2(|f(x)-y|) = -11.4076
log2(|f(x)-y|) = -13.6346
log2(|f(x)-y|) = -12.6169
log2(|f(x)-y|) = -12.2094
log2(|f(x)-y|) = -12.6128
log2(|f(x)-y|) = -10.8884
log2(|f(x)-y|) = -15.5421
log2(|f(x)-y|) = -12.4875
log2(|f(x)-y|) = -10.6895
log2(|f(x)-y|) = -13.818
log2(|f(x)-y|) = -13.1507
log2(|f(x)-y|) = -12.4648
log2(|f(x)-y|) = -12.4614
log2(|f(x)-y|) = -13.0653
log2(|f(x)-y|) = -12.5349
log2(|f(x)-y|) = -11.3919
log2(|f(x)-y|) = -12.5398
log2(|f(x)-y|) = -12.6499
log2(|f(x)-y|) = -10.8902
log2(|f(x)-y|) = -12.6511
log2(|f(x)-y|) = -16.5083
```

```
log2(|f(x)-y|) = -12.5022
log2(|f(x)-y|) = -11.715
log2(|f(x)-y|) = -12.6585
log2(|f(x)-y|) = -10.8606
log2(|f(x)-y|) = -13.1116
log2(|f(x)-y|) = -12.0981
log2(|f(x)-y|) = -13.3051
log2(|f(x)-y|) = -12.6072
log2(|f(x)-y|) = -13.033
log2(|f(x)-y|) = -13.3311
log2(|f(x)-y|) = -12.8382
log2(|f(x)-y|) = -11.1676
log2(|f(x)-y|) = -12.6524
log2(|f(x)-y|) = -15.0041
log2(|f(x)-y|) = -12.4653
log2(|f(x)-y|) = -10.9517
log2(|f(x)-y|) = -11.4226
log2(|f(x)-y|) = -10.5374
log2(|f(x)-y|) = -11.3379
log2(|f(x)-y|) = -10.9822
log2(|f(x)-y|) = -13.2296
log2(|f(x)-y|) = -11.4809
log2(|f(x)-y|) = -12.0781
log2(|f(x)-y|) = -10.7768
log2(|f(x)-y|) = -12.6147
log2(|f(x)-y|) = -11.0175
log2(|f(x)-y|) = -12.6678
log2(|f(x)-y|) = -11.1031
log2(|f(x)-y|) = -12.1388
log2(|f(x)-y|) = -15.4793
log2(|f(x)-y|) = -13.6158
log2(|f(x)-y|) = -11.2477
log2(|f(x)-y|) = -14.293
log2(|f(x)-y|) = -12.7136
log2(|f(x)-y|) = -11.4865
```

<그림 18. Problem3 파일 실행 결과>

구현한 Problem3를 한번 실행한 결과이다. 128개의 결과 값이 모두 조건을 만족하였다.