




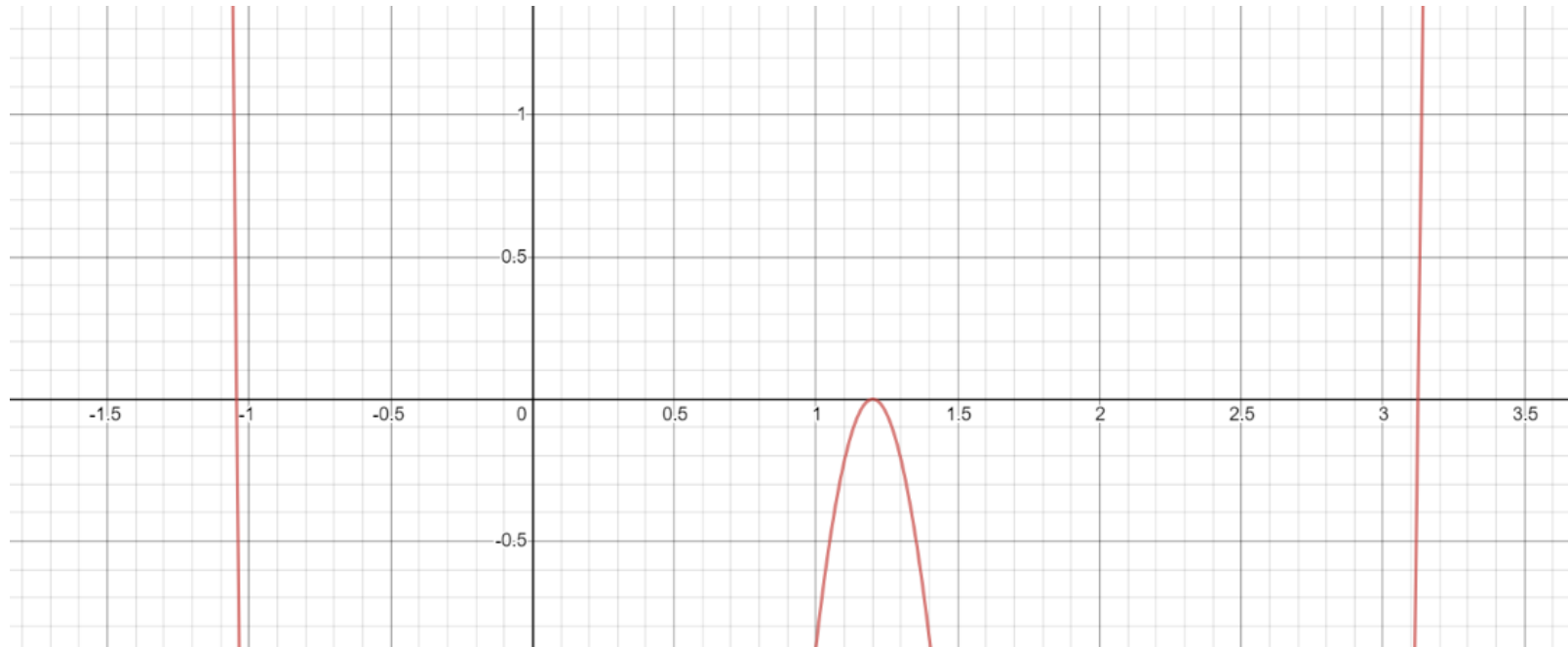
수치해석

HW1

2018008613 안상욱

그래프의 대략적인 모양

1  $5x^4 - 22.4x^3 + 15.85272x^2 + 24.161472x - 23.4824832$

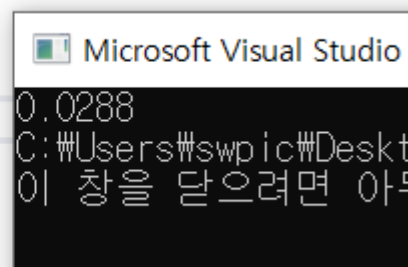


Desmos라는 사이트에서 위 식에 대한 그래프를 대략적으로 그려서
근이 -1, 1.2, 3.1 세 곳 근처에 위치한 것을 확인했습니다.

C 프로그래밍

```
double f(double x)
{
    return 5 * x*x*x*x - 22.4*x*x*x + 15.87272*x*x + 24.161472*x - 23.4824832;
}
```

```
int main()
{
    cout << f(1.2);
}
```



위와 같은 C 프로그래밍을 통해 $f(1.2)$ 의 값이 0.288이라는 양수라는 것을 확인했고, Bisection의 구간을 $(-1, 5)$, $(-0.5, 1.0)$, $(1.2, 1.5)$, $(3.0, 3.5)$ 네 구간으로 나누어주었습니다.

Bisection 프로그램 구현

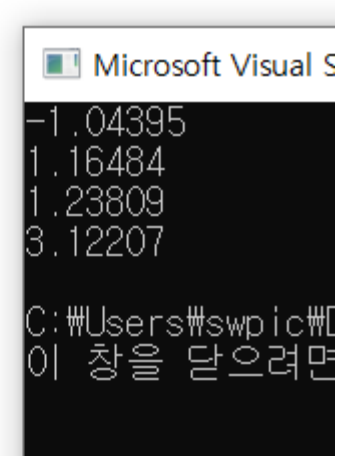
```
#define INF 0.001
```

```
double f(double x)
{
    return 5 * x*x*x*x - 22.4*x*x*x + 15.87272*x*x + 24.161472*x - 23.4824832;
}
```

```
double bisection(double left, double right)
{
    if (f(left)*f(right) > 0)
        return 0;
    double mid;
    while (right - left > INF)
    {
        mid = (left + right) / 2;
        if (f(mid)*f(left) > 0)
            left = mid;
        else
            right = mid;
    }
    return mid;
}
```

먼저 다음과 같이 INF를 0.001로 두어 left와 right의 오차가 0.001 이내로 들어가게 된다면 그 mid값을 return해주었습니다.

```
int main()
{
    cout << bisection(-1.5, -0.5) << "\n";
    cout << bisection(1.0, 1.2) << "\n";
    cout << bisection(1.2, 1.5) << "\n";
    cout << bisection(3.0, 3.5) << "\n";
}
```



```
Microsoft Visual S
-1.04395
1.16484
1.23809
3.12207

C:\Users\swpic\W
이 창을 닫으려면
```

main문에서 앞에서 정한 네 개의 구간을 통해 bisection을 실행한 결과
네 근 -1.04395 1.16484 1.23809 3.12207 을 구할 수 있었습니다.

Newton-Raphson 프로그램 구현

$$20x^3 - 67.2x^2 + 31.70544x + 24.161472$$

```
double g(double x)
{
    return 20 * x*x*x - 67.2*x*x + 31.70544*x + 24.161472;
}
```

먼저 $f(x)$ 를 미분한 $f'(x)$ 식을 구했고, 이 식을 C프로그램으로 구현했습니다.


```

#define INF 0.001

double f(double x)
{
    return 5 * x*x*x*x - 22.4*x*x*x + 15.87272*x*x + 24.161472*x - 23.4824832;
}

double g(double x)
{
    return 20 * x*x*x - 67.2*x*x + 31.70544*x + 24.161472;
}

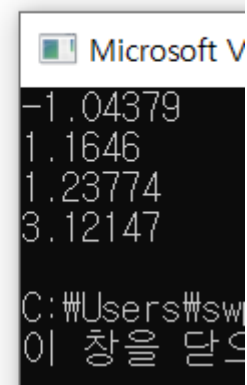
double NewtonRaphson(double x)
{
    double next_x;
    while (1)
    {
        next_x = x - f(x) / g(x);
        if (abs((next_x - x) / next_x) < INF)
            break;
        x = next_x;
    }
    return next_x;
}

```

다음과 같이 C 프로그램을 작성했습니다. g 함수는 f 함수를 미분한 함수를 의미하고,

next_x는 x_{i+1} , x는 x_i 를 의미합니다. 이전의 x값과 다음 x값의 변화율이 0.001보다 작아질 때 그 x값을 반환해줍니다.

```
int main()
{
    cout << NewtonRaphson(-1.5) << "\n";
    cout << NewtonRaphson(1) << "\n";
    cout << NewtonRaphson(1.5) << "\n";
    cout << NewtonRaphson(3) << "\n";
}
```



Microsoft V

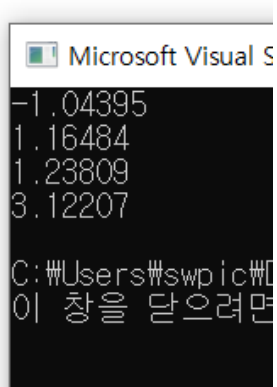
-1.04379
1.1646
1.23774
3.12147

C:\Users\sw
이 창을 닫으

그 이후 main문에서 -1.5 1 1.5 3 네 개의 지점에서 Newton Raphson을 실행한 결과
-1.04379 1.1646 1.23774 3.12147 네 개의 근을 구할 수 있었습니다.

Bisection, Newton-Raphson 비교 및 분석

```
int main()
{
    cout << bisection(-1.5, -0.5) << "\n";
    cout << bisection(1.0, 1.2) << "\n";
    cout << bisection(1.2, 1.5) << "\n";
    cout << bisection(3.0, 3.5) << "\n";
}
```

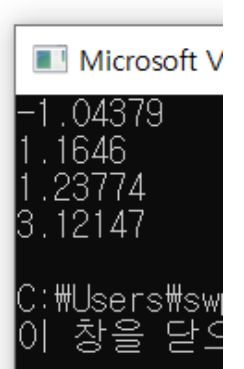


Microsoft Visual S

```
-1.04395
1.16484
1.23809
3.12207
```

C:\Users\swpic\W...
이 창을 닫으려면

```
int main()
{
    cout << NewtonRaphson(-1.5) << "\n";
    cout << NewtonRaphson(1) << "\n";
    cout << NewtonRaphson(1.5) << "\n";
    cout << NewtonRaphson(3) << "\n";
}
```



Microsoft V

```
-1.04379
1.1646
1.23774
3.12147
```

C:\Users\sw...
이 창을 닫으

다음과 같이 두 프로그램을 이용해서 4개의 근을 구해 보았을 때 거의 유사한 값을 구할 수 있었습니다.

그래프를 그려 대략적인 근의 위치를 알 수 있어서 편하게 Bisection의 구간들을 정할 수 있었습니다.

그리고 $f(x)$ 의 함수 식을 알고 있어 이를 미분한 $f'(x)$ 의 식을 쉽게 구할 수 있었고, 대략적인 근의 위치도 알고 있어서 NewtonRaphson의 시작지점 또한 쉽게 알 수 있었습니다.

원래 결과와 비교

```
int main()
{
    cout << bisection(-1.5, -0.5) << "n";
    cout << bisection(1.0, 1.2) << "n";
    cout << bisection(1.2, 1.5) << "n";
    cout << bisection(3.0, 3.5) << "n";
}
```

```
Microsoft Visual S
-1.04395
1.16484
1.23809
3.12207
C:\Users\swpic\
이 창을 닫으려면
```

$5x^4 - 22.4x^3 + 15.85272x^2 + 24.161472x - 23.4824832$

Extended Keyboard

Upload

Input interpretation:

$5x^4 - 22.4x^3 + 15.85272x^2 + 24.161472x - 23.4824832$

Result:

$5x^4 - 22.4x^3 + 15.8527x^2 + 24.1615x - 23.4825$

Roots:

$x \approx -1.044$

$x \approx 1.2$

$x \approx 1.2$

$x \approx 3.124$

근을 구해주는 사이트에서 근을 계산해 비교해본 결과 오차가 거의 없이 비슷한 값을 가지는 것을 확인해 볼 수 있었습니다.

감사합니다!