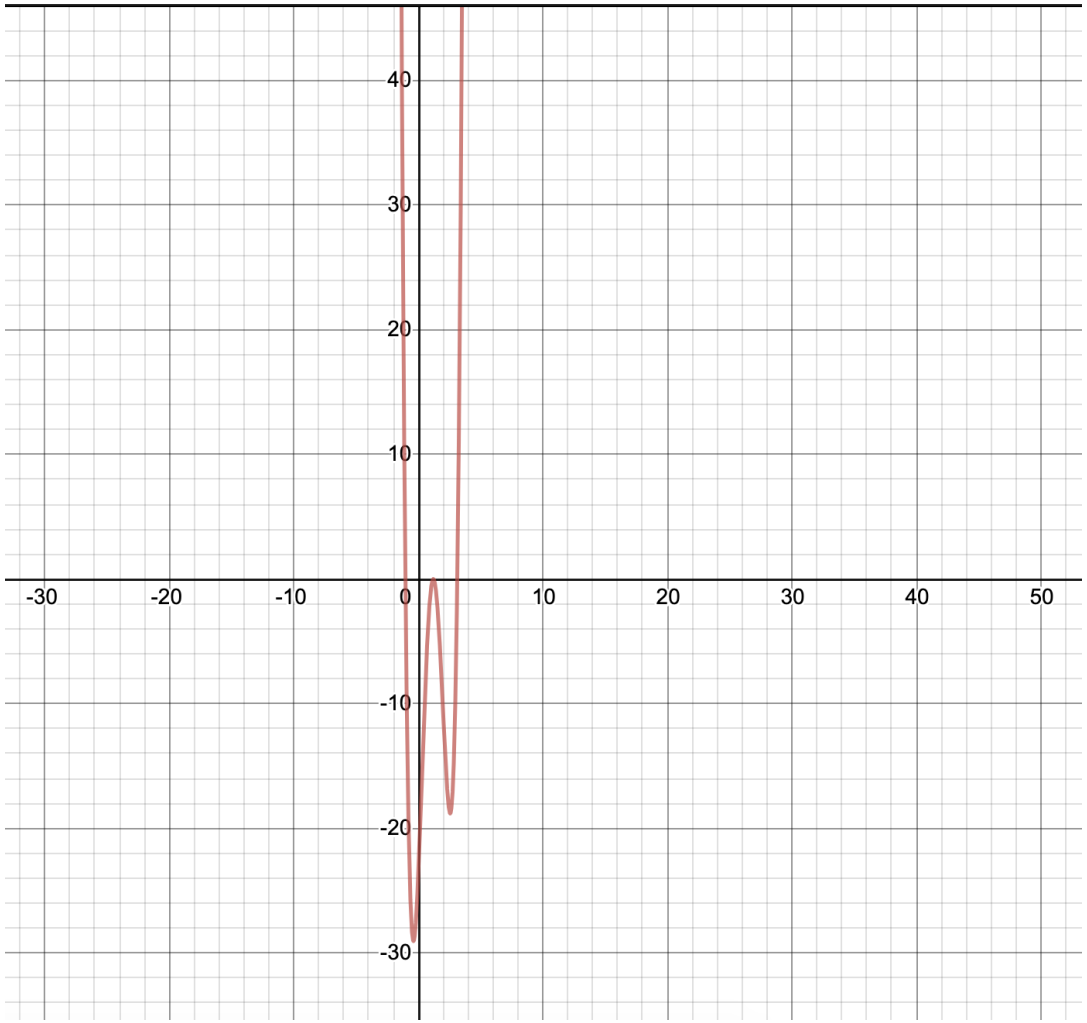
먼저 그래프의 대략적인 그림과 근의 범위를 정하기 위해 인터넷에서 함수의 그래프를 그려주는 사이트를 찾아 사용했다.



Bisection

먼저 오른쪽 사진처럼 주어진 식에 x값을 대입 했을때의 값을 구하기 위해 function 이라는 함수를 만들었다.

```
def function(x):
    return (
        5 * pow(x, 4)
        - 22.4 * pow(x, 3)
        + 15.85272 * pow(x, 2)
        + 24.161472 * x
        - 23.4824832
    )
```

Bisection

오른쪽 사진은 bisection을 구현하기위한 코드이다. 근사치 즉 에러는 0.0001범위 내에 허용했고, c값을 function에 대입 했을때의 값에 따라 firtst와 second값을 바꿔가며 근을 찾아갔다.

```
def bisection(a, b):
    first = a
    second = b
    if function(a) <= 0.0001 and function(a) >= -0.0001:
        return a
    if function(b) <= 0.0001 and function(b) >= -0.0001:
        return b
    while True:
        c = (first + second) / 2
        equation = function(c)
        if equation <= 0.0001 and equation >= -0.0001:
            return c
        elif equation > 0:
            if function(a) > 0:
                first = c
            else:
                second = c
        else:
            if function(a) > 0:
                second = c
            else:
                first = c
```

Bisection

근을 찾기위해 범위를 임의로 지정해 주었고, 밑에는 실행결과이다.

```
print(bisection(-1.5, -1))
print(bisection(1, 1.2))
print(bisection(3.1, 3.2))
```

```
-1.04400062561
1.2
3.12399902344
uwonjin-ui-MacBookPro:수치해석 과제 woowonjin$
```

Newton-Raphson

Newton-Raphson 을 사용하기 위해 원래 함수 function을 미분한 함수를 직접 구하여 derivative로 나타내었다.

```
def function(x):
    return (
        5 * pow(x, 4)
        - 22.4 * pow(x, 3)
        + 15.85272 * pow(x, 2)
        + 24.161472 * x
        - 23.4824832
    )


def derivative(x):
    return 20 * pow(x, 3) - 67.2 * pow(x, 2) + 31.70544 * x + 24.161472
```

Newton-Raphson

Bisection과 마찬가지로 오차 허용은 0.0001까지 했다. 그리고 $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$ 식을 사용하는 newton이라는 함수를 만들었다.

```
def newton(a):
    x = a
    if function(x) <= 0.0001 and function(x) >= -0.0001:
        return x
    while True:
        if derivative(x) == 0:
            return -10000
        x = x - function(x) / derivative(x)
        if function(x) <= 0.0001 and function(x) >= -0.0001:
            return x
```

Newton-Raphson

근을 찾기위해 임의의 값을
넣어주었다.

```
print(derivative(1.2))
print(newton(-1.1))
print(newton(1.1))
print(newton(1.3))
print(newton(3))
```

```
-1.06581410364e-14
-1.04400000016
1.19843093263
1.20154620278
3.12400076396
```

비교

처음 bisection에서 근을 구했을 때 프로그램에서 function에 1.2값을 넣으니 0이라는 값이 나왔기때문에 1.2가 중근이라고 생각했다. 하지만 Newton-Raphson을 사용하여 구할때 derivative값에 1.2를 넣어보니 0이 나오지 않았다. 즉 이는 컴퓨터가 엄청 작은 오차를 무시했다고 생각한다. 중근이 되려면 1.2일 때 function값이 0인 동시에 derivative값도 0이어야 하기 때문이다. 그래서 이번 과제로 알게된 점은 bisection으로는 중근의 존재여부를 확인하기 힘들지만 Newton-Raphson으로는 derivative의 값을 이용한다면 찾을 수 있겠다는 생각이 들었다.