수치해석
HW #1

Find the roots of the polynomial equation
using Bisection, Newton-Raphson

2018007956 김채아

$f(x) = 5x^4 - 22.4x^3 + 15.852\dot{7}2x^2 + 24.161\dot{4}\dot{7}2x - 23.4824832$

$f(0) = -23.\sim$

$f(1) = -0.8\sim < 0$

$f(2) = 80 - 179.2 + 63.41088 + 48.322\sim$
$-23\sim = -10.9\sim < 0$

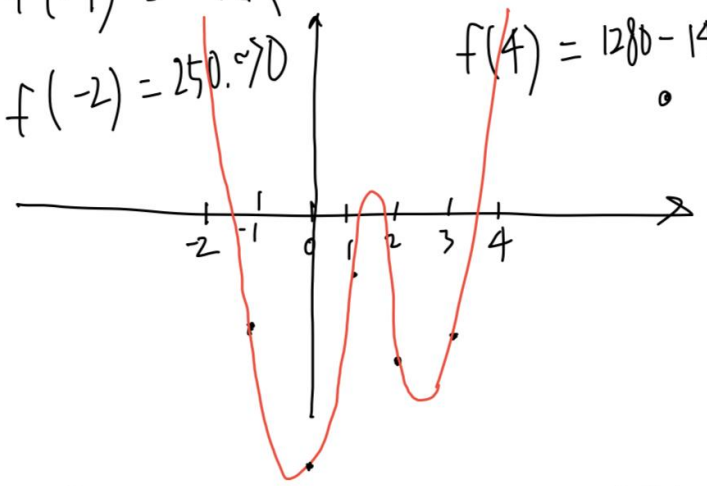
$f(3) = 405 - 604.8 + 142.67\sim + 17.\dot{4}8\dot{4}2$
$-23.\sim = 8.1\sim < 0$

$f(4) = 1280 - 1433.6 + 253.643\sim + 96.\dot{6}5\dot{8}\dot{8}\dot{8}$
$-23.\sim = 173.2 > 0$

(Bracket)

$f(-1) = -4. < 0$

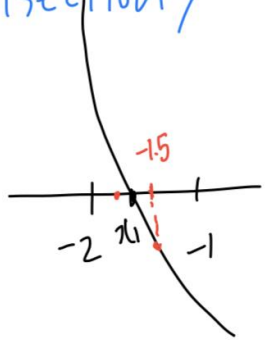
$f(-2) = 250.\sim > 0$



→ 근: $-2 < x_1 < -1$, $1 < x_2, x_3 < 2$, $3 < x_4 < 4$

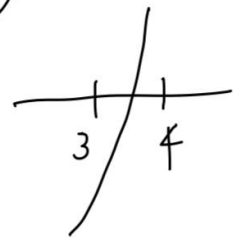
<bisection>

①



②, ③

④



함수 f의 개형을 알아내기 위해 특정 점을 대입해서
함수값의 부호를 파악한다

함수의 부호가 바뀌는 구간을 찾아서 그래프의 개형을 그린다

다항함수는 모든 실수에서 연속이므로
Bracket 개념을 이용하여
위에서 그래프 개형을 알아내기 위해 알아낸
함수의 부호가 바뀌는 구간을
Brackets (intervals)로 설정한다

[근이 있다고 예상되는 구간 (intervals)]
1) $-2 < x < -1$
2) $1 < x < 2$ (2개)
3) $3 < x < 4$

[Bisection Method]

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

xl = int(input('lower bound : '))
xu = int(input('upper bound : '))

def bisection(xl, xu):
    if f(xl)*f(xu) > 0:
        print('No root found')
    else:
        while (xu-xl)/2 > 0.0000001: #tolerance: 허용오차
            m = (xu+xl)/2
            if f(m) == 0:
                return m
            elif f(xl)*f(m)>0:
                xl = m
            else:
                xu = m
        return m

root = bisection(xl, xu)
print('root: ', root)
```

앞에서 설정한 intervals를
Upper bound, Lower bound로 입력받는다

Bracket : interval의 끝 값의 부호가 서로 다른 경우,
그 구간 안에서 적어도 하나의 근이 존재한다.
=〉 f(xl)*f(xu)〈0

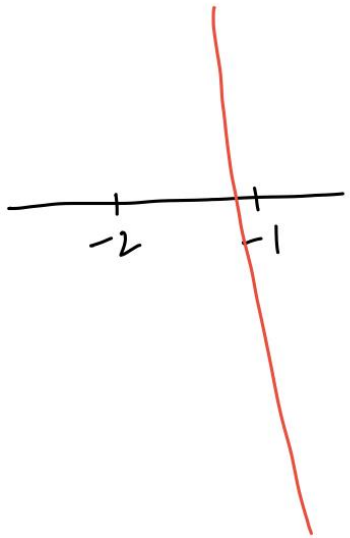
f(xl)*f(xu)〉0 이면 근이 존재하는지 장담할 수 없으므로
no root found 라고 출력
f(xl)*f(xu)〈0 이면 ½ 구간으로 계속 좁혀나가기 위해 반복문을 돌린다
허용오차를 0.000001로 설정하고 허용오차 이내에서만 반복을 수행한다

운이 좋게 f(m)=0 이 되면 m이 근인데,
대부분의 경우 근은 무리수이므로 정확하게 0이 되지 않는다
이때는 xl, xu를 sign chang가 유지되는 방향으로 Boundary를 바꿔간다

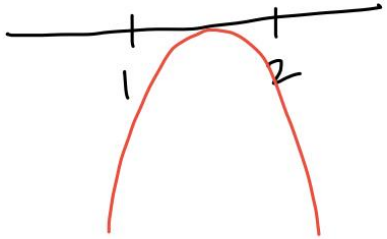
구간을 점점 줄이며 실제 근의 값으로 근 추정값이 수렴하게 된다

[Bisection Method] Results of the roots search

① $-2 < x < -1$

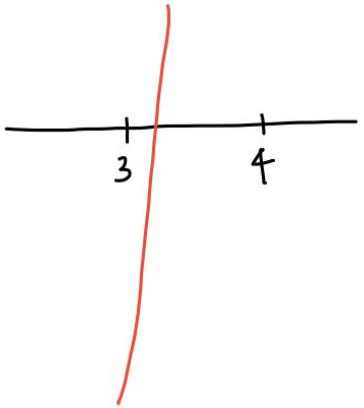


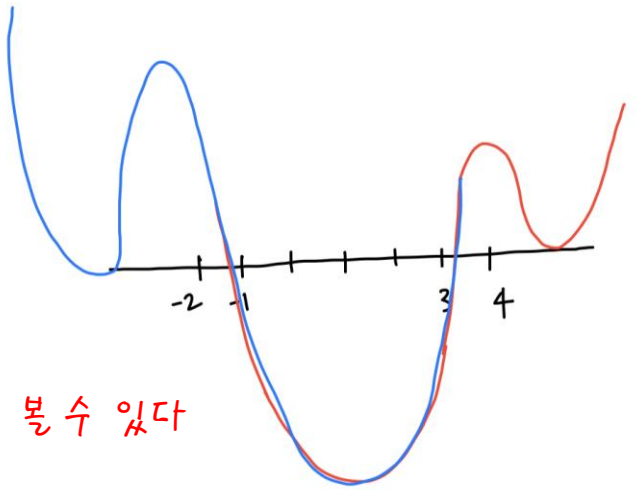
②, ③ $1 < x < 2$



중근(multiple root)로 예상

④ $3 < x < 4$



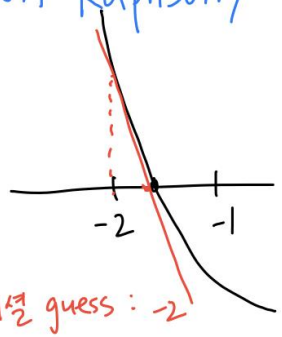


또는 오른쪽과 같은 모양의 그래프를 생각해 볼 수 있다
(파란색과 빨간색 두 종류)

Roots : -1.0440000295639038 No root found 3.1239999532699585

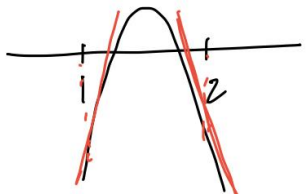
〈Newton-Raphson〉

①



이니셜 guess : -2

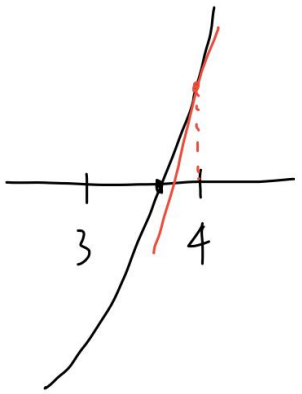
②, ③



이니셜 guess : 1

〃 : 2

④



이니셜 guess : 4

$f(x) = 5x^4 - 22.4x^3 + 15.852172x^2 + 24.161472x - 23.4824832$

$f'(x) = 20x^3 - 67.2x^2 + 31.705444x + 24.161472$

Bisection 방법과 마찬가지로
함수 모양을 통해 근이 어디쯤 존재할지 예측하고
initial guess를 설정해 근의 수렴 값을 찾는다

같은 함수이므로 당연히 intervals는 앞과 같다
[근이 있다고 예상되는 구간 (intervals)]

볼록한 부분으로 접선이 생기게끔 이니셜 guess를 잡아줘야 Estimation이 수렴가능하다

1) $-2 < x < -1$
   initial guess : -2
2) $1 < x < 2$ (2개)
   initial guess : 1, 2
3) $3 < x < 4$
   initial guess : 4

[Newton-Raphson Method]

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

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

def newtonRaphson(x):
    dx = f(x) / derivFunc(x)
    while abs(dx) > 0.0000001:
        dx = f(x) / derivFunc(x)
        x = x - dx
    print("root : ", x)

g = int(input('Initial guess : '))
newtonRaphson(g)
```

guess 사이의 거리, dx값이 0.0000001보다 클때까지만 옆에서 구한, 다음 guess의 식을 돌린다

(허용오차를 0.0000001이라고 설정한 이유는
따로 설정해주지 않는 이상 파이썬에선 소수 16자리까지 출력되는데
허용오차가 0.00000001 또는 0.000000001 … 로 되어버리면
뒷부분이 0000…으로 되어있어 차이가 안 보이기 때문이다
소수점자리를 지정해주지 않고 볼 수 있는 최대한 작은 오차를 가진 결과값을 보고 싶었다.
– X의 위치에 따라 함수의 모양이 달라서 조금씩 차이가 있지만 중요한 부분 아님)

Initial guess (x로 가정)부터 시작해서
X에서의 접선의 방정식을 구해 x축과 만나는 근이
다음에 해당하는 second guess이다

계속 수렴하다보면 더 이상 x가 크게 변하지 않는 점이 근이다

접선의 방정식을 이용해 second guess를 구해보면 아래와 같고

$$f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}}$$

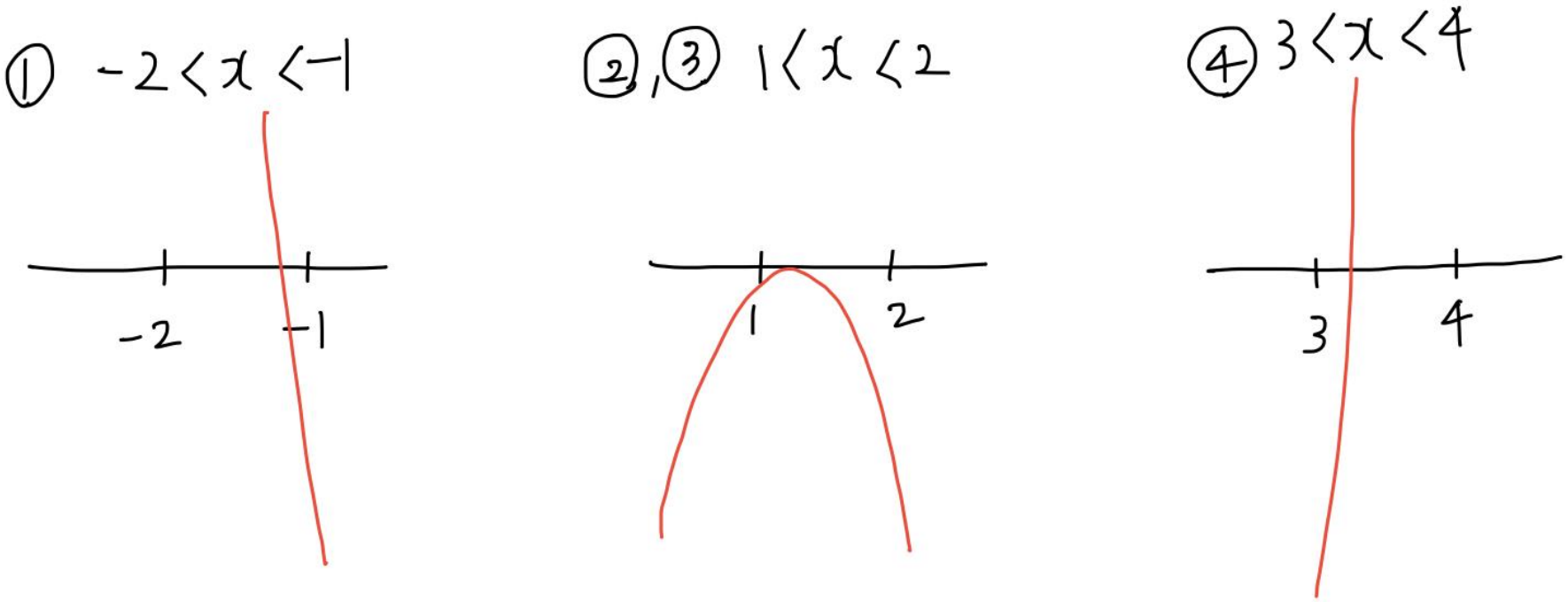
$$f'(x_i)\,x_i - f'(x_i)\,x_{i+1} = f(x_i)$$

$$f'(x_i)\,x_{i+1} = f'(x_i)\,x_i - f(x_i)$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = \Delta x$$

x(i+1) = x(i) – f(x) / f’(x) 라는 식을 얻을 수 있다

[Newton-Raphson Method] Results of the roots search

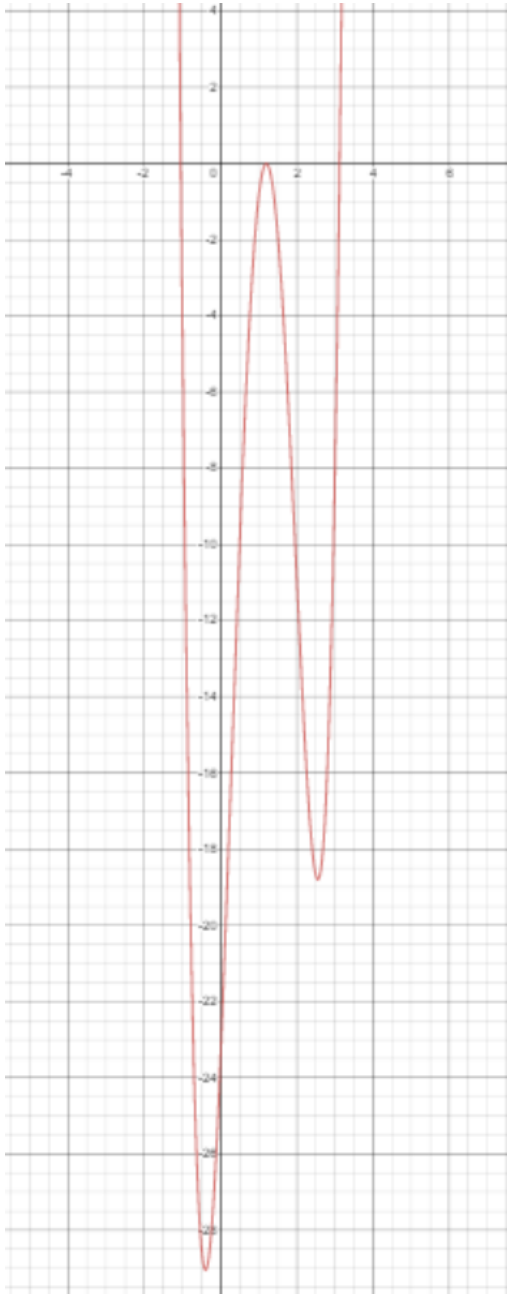


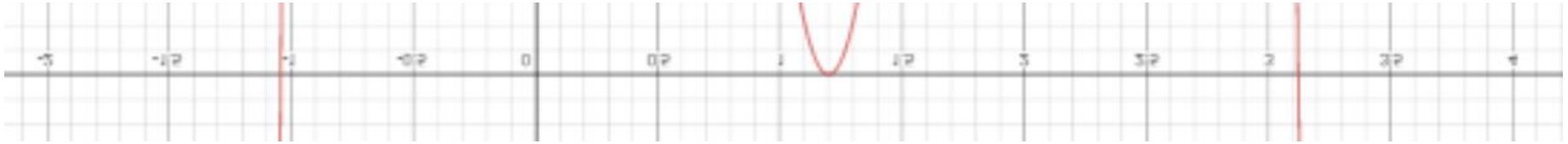
Initial guess :	-2	1	2	4
Roots :	-1.0440000000000056	1.1999991287422416	1.2000003955615655	3.1240000000000214

Bisection결과를 통해 1<x<2 또는 x<−2 또는 x>4에 중근이 있다는 것을 알았다
Newton−Raphson 으로 1<x<2에 수렴하는 값이 존재함을 알 수 있고,
4이상 또는 −2이하의 값을 아무거나 넣어보면 (100, −100을 넣어도)
각각 3.124, −1.044로만 수렴한다

따라서 그래프는 1<x<2에 중근이 존재한다고 말할 수 있다

[실제 근의 값과 측정값 비교]





Bisection	Newton Raphson	True value
-1.0440000295639038	-1.0440000000000056	-261/250 (-1.0440000000000003908)
No root found	1.1999991287422416, 1.2000003955615655	6/5 (1.1999999999999995559)
3.1239999532699585	3.1240000000000214	781/250 (3.1240000000000011013)

[Compare two methods]

	Bisection	Newton-Raphson
중근	함수값의 부호를 통해 근의 존재 유무를 판별하므로 중근이 존재하는 경우 부호가 바뀌지 않아 그 범위에 근이 존재하는지도 알 수 없는 문제가 있다	함수값의 부호가 아닌 initial guess 하나를 가지고 점점 근의 방향으로 수렴해가는 방법이므로 중근이 존재함을 파악할 수 있다
정확도	Bisection 방법 보다 Newton-Raphson방법이 더 실제 값과 유사하다	
코딩	둘 다 코딩이 어렵진 않았지만 입력값의 개수가 적어서 그런지 상대적으로 Newton-Raphson방식이 간결했다	
(이론상으로배운) 장단점	측정값과 근의 실제 위치의 오차의 최대값이 매번 한번씩 반복할때마다 ½씩 줄어들게 된다 (10번만해도 1/1000로 줄어듬)	함수의 모양에 따라 빠르게 근에 수렴할 수 있다 그래프에 변곡점이 있거나, 접선이 기울기가 0이 되거나, 여러 굴곡에 따라 값이 수렴하지 않을 수 있다는 문제점이 있다
이유는 모르겠지만 코딩을 하며 느낀 차이	허용오차를 조금만 건드려도 소수점이하 특정 자리수부터 16자리까지 변화가 있었다	허용오차에 따른 root값의 차이가 Bisection에 비해 거의 없었다