수치해석
HW #2

Find the min locations
using Newton method

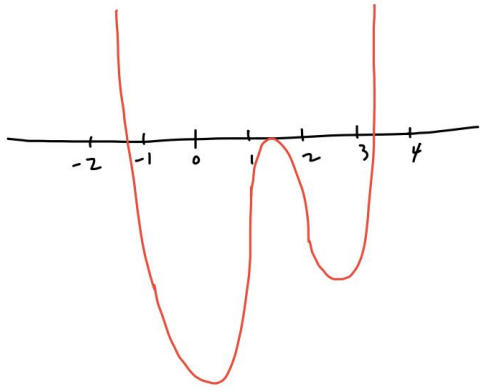
2018007956 김채아

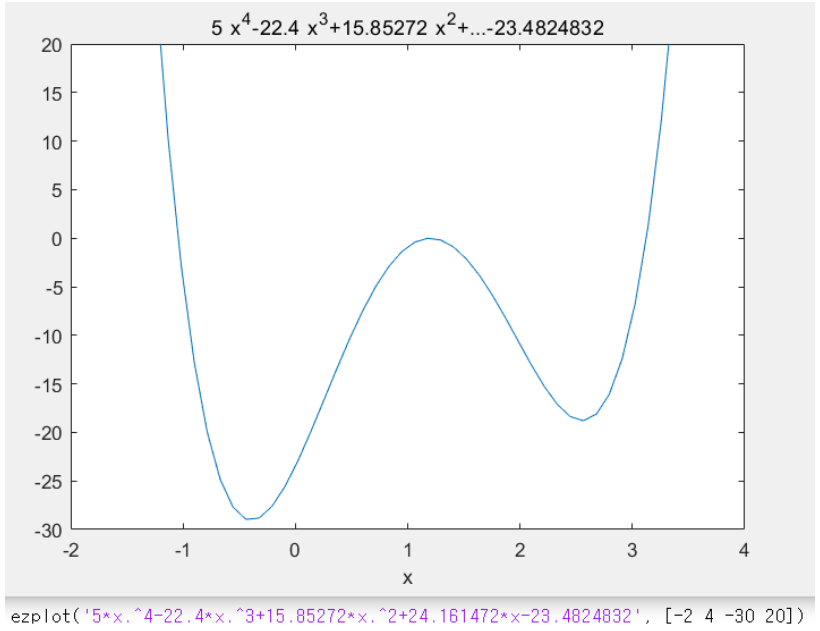
[minimum이 존재하는 구간을 파악하고 initial point 잡기]

$$f(x) = 5x^4 - 22.4x^3 + 15.85272x^2 + 24.161472x - 23.4824832$$

HW1에서의 결과를 바탕으로 함수 모양을 예측하면 아래와 같다

하지만 최소값 위치의 범위는 확정할 수 없으므로 매트랩으로 정확한 함수 그래프를 알아내어 미니멈의 위치를 파악했다





1) -1<x<0 (global minimum)
initial point : 0
(-1과 0중에 최소값에 더 가까워보이는 0으로 잡음
아무거나 잡아도 됨)

2) 2<x<3 : local minimum
initial point : 3

[1. Using exact 1st and 2nd derivatives]

```
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 secondDerivFunc(x):
    return 60*pow(x,2)-134.4*x+31.70544

def newton(x):
    dx = derivFunc(x) / secondDerivFunc(x)
    while abs(dx) > 0.0000001:
        dx = derivFunc(x) / secondDerivFunc(x)
        x = x - dx
    print("x of the minimum : ", x)

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

input : 내가 설정한 initial guess 값
output : x of the minimum

exact하게 함수를 구하는 방법을 사용할 것이므로
직접 미분해서 정확한 식의 함수를 만들어준다

newton함수는 아래 두 정보를 이용하여 작성하였다

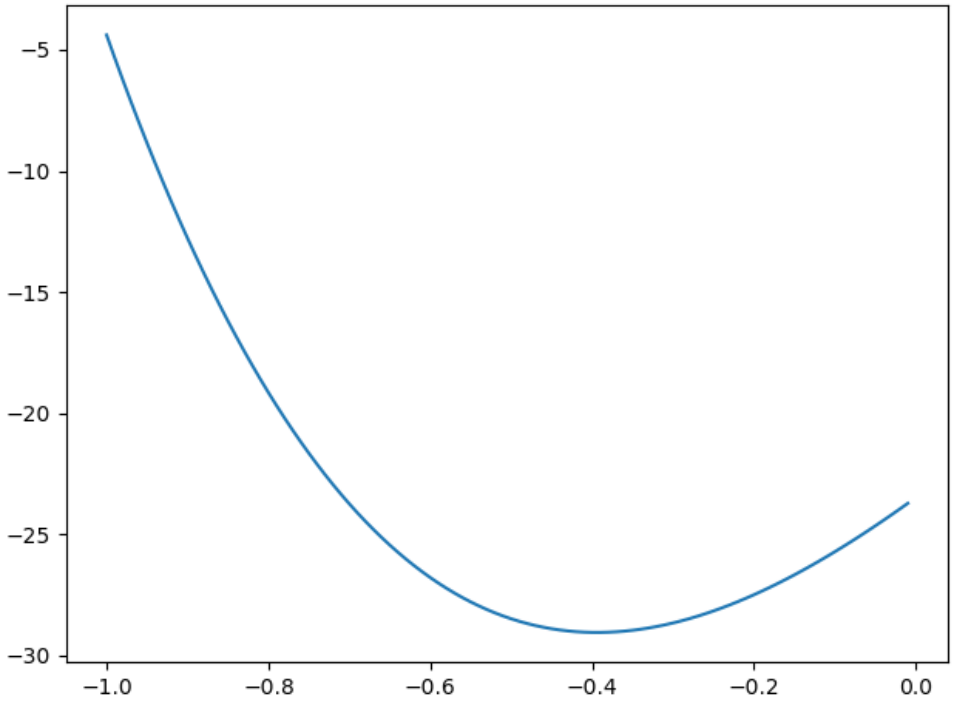
1. x(i+1) = x(i) − α * f'(x) / f''(x) + dx
을 이용하여 다음 x값을 설정한다

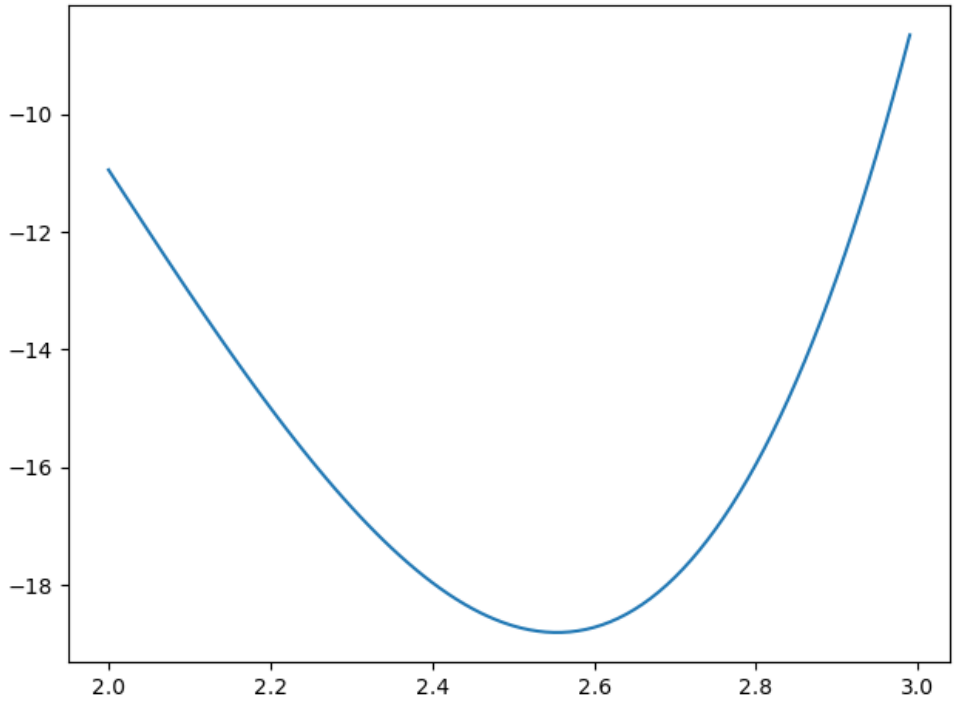
2. dx = f'(x) / f''(x) = abs(x − x1)
x1은 x의 다음 값.
즉 위 식은 x의 변화량을 뜻한다.
(x1 = x − dx ⟹ x − x1 = dx)

x의 변화량 dx가 특정 값보다 줄어들면
수렴을 멈추도록 하였고, 1번 식으로 반복문을 돌려
수렴하도록 만들어주었다
지금 이 코드는 moment 값을 넣어주지 않은 코드이다

그러면 함수값이 최소값이라고 추정되는 x가 나온다

[1. Using exact 1st and 2nd derivatives] Results of the minimum search






```
x = np.arange(-1, 0, 0.01)
plt.plot(x, 5*pow(x,4)-22.4*pow(x,3)+15.85272*pow(x,2)+24.161472*x-23.4824832)
plt.show()
```

x범위만 바꿔서 그대로 적용(−1부터 0.01씩 더해져 0까지)

1) -1<x<0 : global minimum
initial point : 0
=> -0.39415331631414785

2) 2<x<3 : local minimum
initial point : 3
=> 2.55415331631415

[1. Using exact 1st and 2nd derivatives] moment 생성해보기

```
def newton(x):
    dx = derivFunc(x) / secondDerivFunc(x)
    while abs(dx) > 0.0000001:
        dx = derivFunc(x) / secondDerivFunc(x)
        # x(i+1) = x(i) - f'(x) / f''(x) +dx
        x = x - dx + dx/2
    print("x of the minimum : ", x)
```

moment는 x(i−1)에서 xi로 이동했을 때의 값을 보고 거기에 맞춰서
additive하게 ½만 이동하게 만들었다
xi값의 이전 변화량 그 이상으로는 변화가 생기지 않도록 제어하는 것이다.

수렴하는 속도가 느려 질 수는 있지만 수렴이 좀 더 안정적으로 되게끔 보장해줌

1) -1<x<0 : global minimum
initial point : 0
⇒ -0.39415326751914753

moment X :
=> -0.39415331631414785

moment 존재 유무에 따른 x값 차이 :
=> 0.0000000487950003

2) 2<x<3 : local minimum
initial point : 3
⇒ 2.5541533448091895

moment X :
=> 2.55415331631415

moment 존재 유무에 따른 x값 차이 :
=> 0.000000028495039

마지막 페이지 표를 참고해보면, 미세한 차이지만 moment가 있을 때 실제 값과 더 가깝다

[2. Using approximation]

```
h=0.1

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 (f(x+h)-f(x))/h

def secondDerivFunc(x):
    return (derivFunc(x)-derivFunc(x-h)) / h

def newton(x):
    dx = derivFunc(x) / secondDerivFunc(x)
    while abs(dx) > 0.0000001:
        dx = derivFunc(x) / secondDerivFunc(x)
        x = x - dx
    print("x of the minimum : ", x)


g = int(input('Initial guess : '))  # Initial guess
newton(g)
```

input : 내가 설정한 initial guess 값
output : x of the minimum

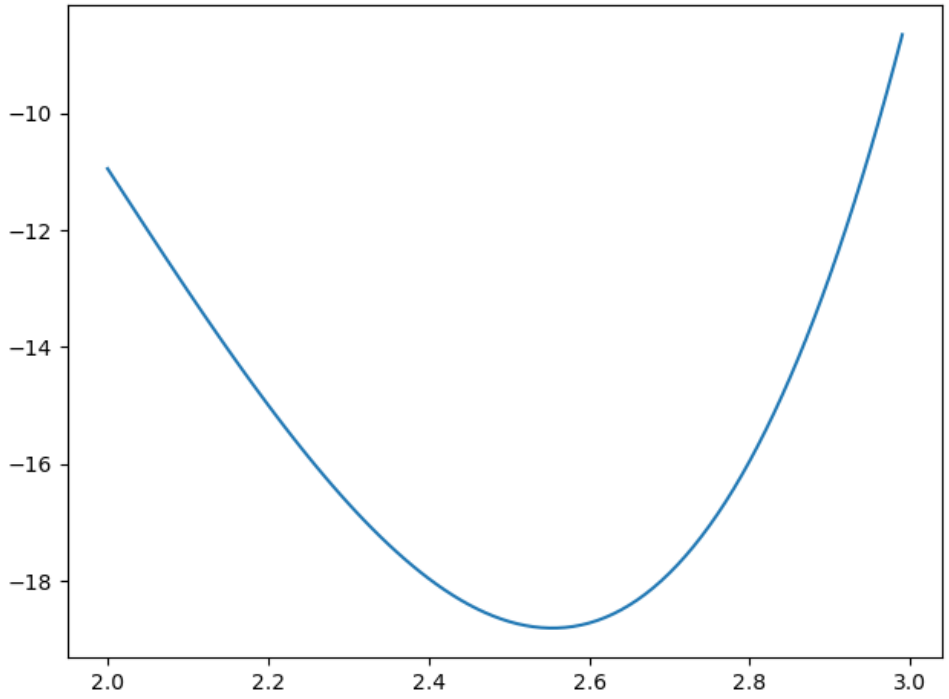
직접 미분을 하지않고,
f'(xi) = (f(x(i+h)) – f(xi)) / h
f''(xi) = (f'(xi) – f'(x(i–1))) / h
= (f(x(i+1)) – 2*f(xi) + f(x(i–1))) / h^2
를 이용한다

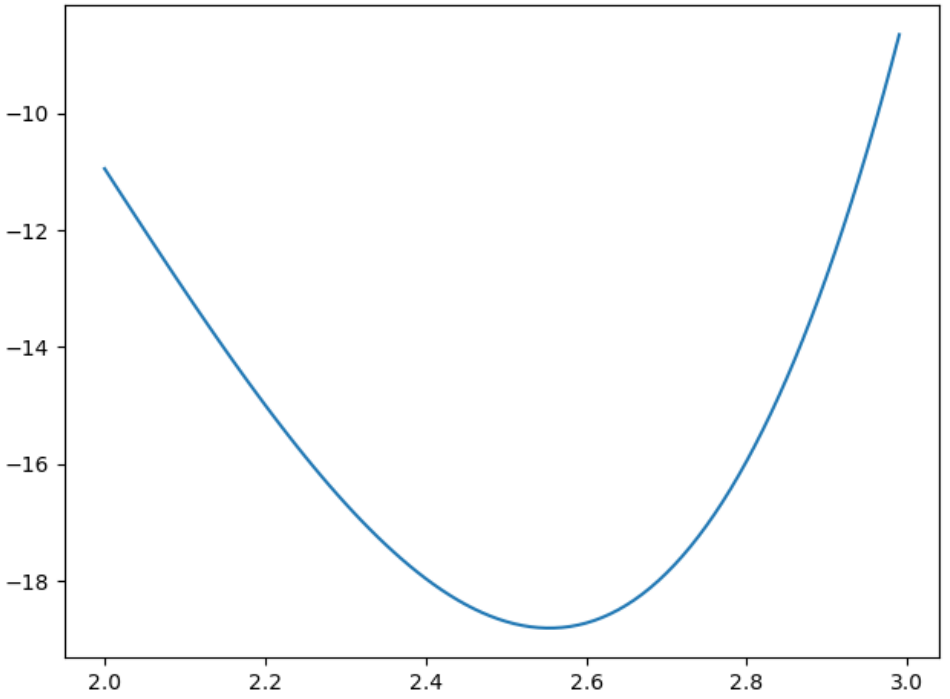
h는 내가 임의로 설정해준 작은 상수값이다
여기선 0.1로 설정하였는데 값이 더 작을수록(0에 가까울수록)
제대로 미분 되어 실제 값과 더욱 비슷하게 나온다

newton함수는 첫번째 방법인
Using exact 1st and 2nd derivatives 과 같은 내용이다

마찬가지로 함수값이 최소값이라고 추정되는 x가 나온다

[2. Using approximation] Results of the minimum search






```
x = np.arange(-1, 0, 0.01) – x범위만 바꿔서 그대로 적용(–1부터 0.01씩 더해져 0까지)
plt.plot(x, 5*pow(x,4)-22.4*pow(x,3)+15.85272*pow(x,2)+24.161472*x-23.4824832)
plt.show()
```

1) -1<x<0 : global minimum
initial point : 0
=> -0.44334773555062823

2) 2<x<3 : local minimum
initial point : 3
=> 2.503254977009301

[2. Using approximation] moment 생성해보기

```
def newton(x):
    dx = derivFunc(x) / secondDerivFunc(x)
    while abs(dx) > 0.0000001:
        dx = derivFunc(x) / secondDerivFunc(x)
        # x(i+1) = x(i) - f'(x) / f''(x) +dx
        x = x - dx + dx/2
    print("x of the minimum : ", x)
```

[1. Using exact 1st and 2nd derivatives] 과 같은 방법으로 moment를 생성하였다

1) -1<x<0 : global minimum
initial point : 0
=> -0.4433476921009953

moment X :
=> -0.44334773555062823

moment 존재 유무에 따른 x값 차이 :
=> 0.0000000434496329

2) 2<x<3 : local minimum
initial point : 3
⇒ 2.503254992207833

moment X :
⇒ 2.503254977009301

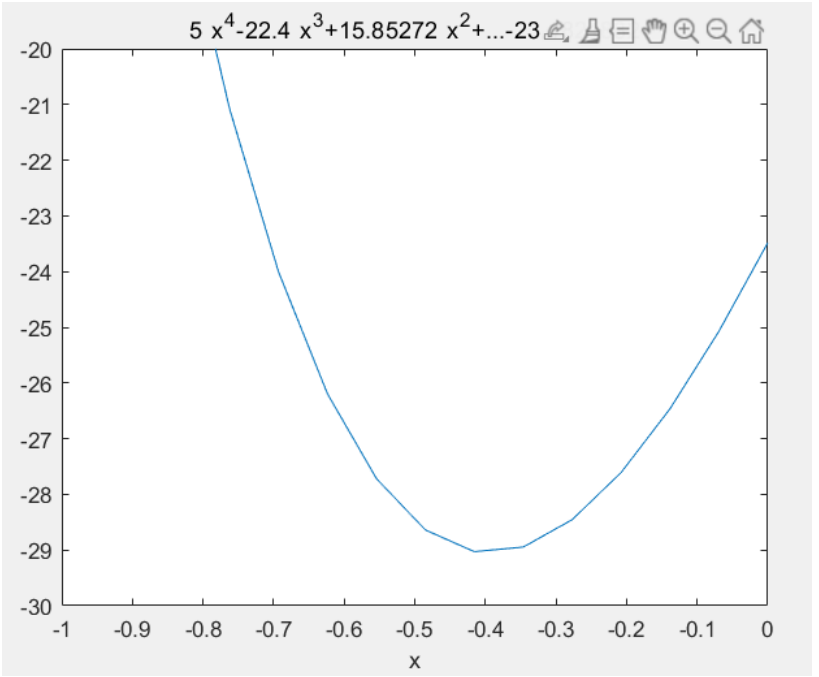
moment 존재 유무에 따른 x값 차이 :
=> 0.000000015198532

마지막 페이지 표를 참고해보면, 미세한 차이지만 moment가 있을 때 실제 값과 더 가깝다

[real value (by Matlab)]

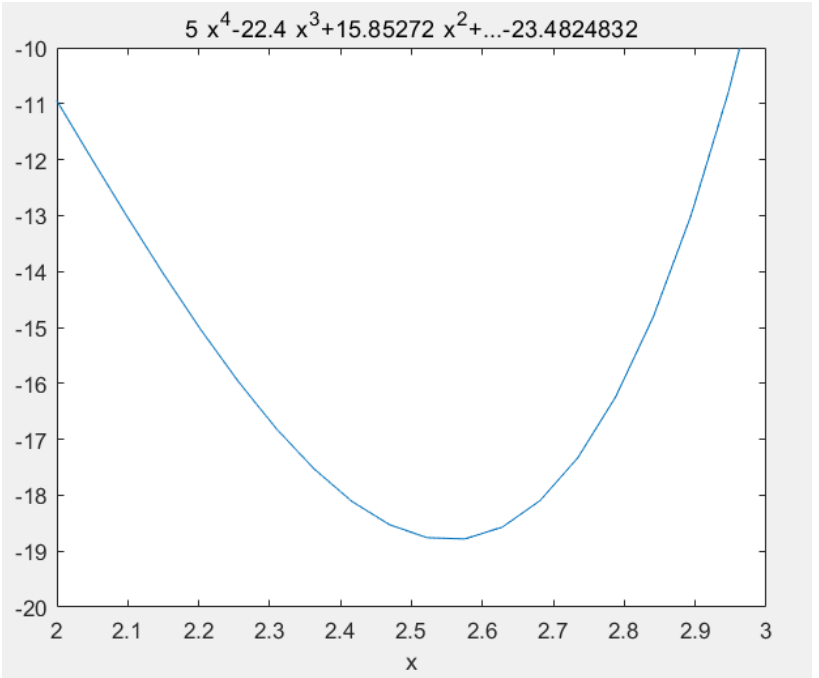
```
>> format long
>> fminsearch('5*x.^4-22.4*x.^3+15.85272*x.^2+24.161472*x-23.4824832',0)

ans =

  -0.394125000000000
```

1) -1<x<0 (global minimum)
initial point : 0
=> -0.394125000000000…




```
>> f = @(x) 5*x.^4-22.4*x.^3+15.85272*x.^2+24.161472*x-23.4824832

f =

  다음 값을 갖는 function_handle:

    @(x)5*x.^4-22.4*x.^3+15.85272*x.^2+24.161472*x-23.4824832

>> fminsearch(f,3)

ans =

   2.554174804687499
```

2) 2<x<3 : local minimum
initial point : 3
=> 2.554174804687499…



[Compare] 1st method & 2nd method & real value (by Matlab)

최소값 존재 범위	Using exact 1st and 2nd derivatives (moment X / moment O)		Using approximation (moment X / moment O)		Real value
-1<x<0 (initial point: 0)	-0.39415331631414785	-0.3941532675191475 3	-0.4433477355506 2823	-0.4433476921009953	-0.394125000000000
Real value - estimation	0.0000283163141478	0.0000282675191475	0.0492227355506282	0.0492226921009953	
2<x<3 (initial point: 3)	2.554153316314 15	2.554153344809 1895	2.503254977009 301	2.503254992207 833	2.554174804687499
Real value - estimation	0.0000214883733 49	0.00002145987831	0.0509198276781 98	0.0509198124796 66	

1. 지금 결과에선 미분함수를 exact하게 넣어줘서 계산하는 것이 approximation하는 것보다 실제 값에 더 가깝게 나왔는데 임의의 상수 h값이 0에 가까울수록 approximation 할 때 더 정확한 값이 나올 수 있다
2. moment를 쓰지 않은 것보다 쓴 것이 더 실제 값과 가깝다.
   실제 값 – 측정 값의 차이가 더 작은 쪽은 모멘트를 쓴 쪽이었다.