

수치해석 HW #2

Numerical analysis

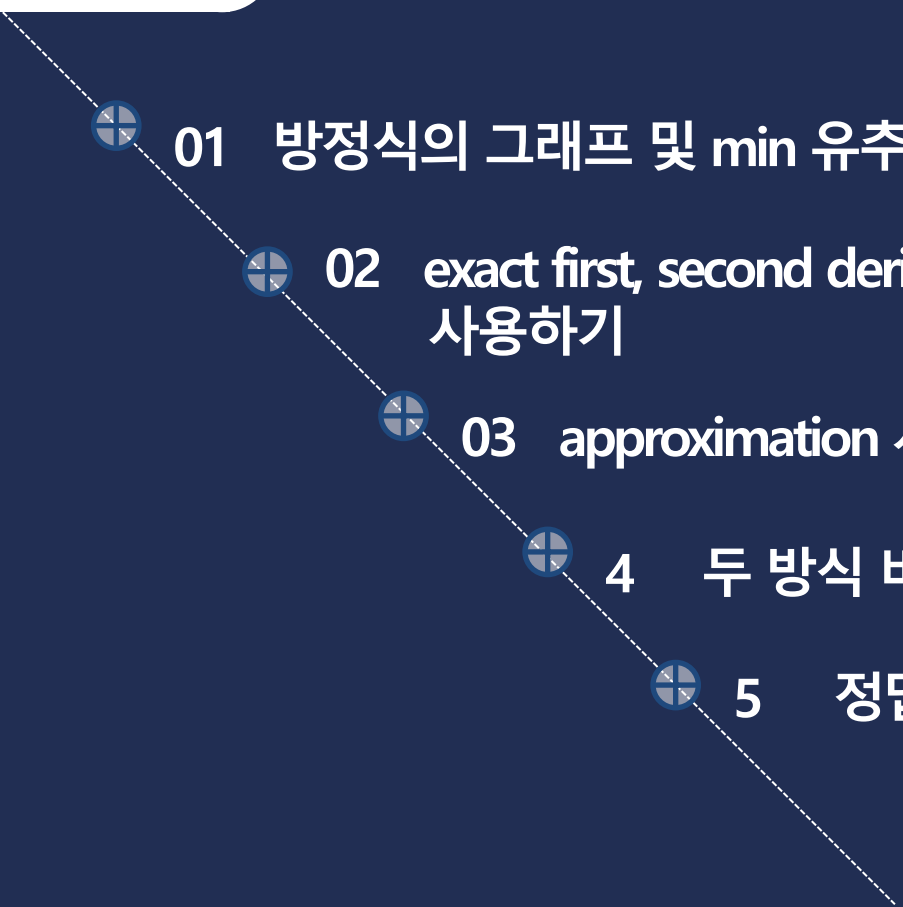
JaeHoon KANG

강 재 훈

HW #2

Find the min locations using Newton-method

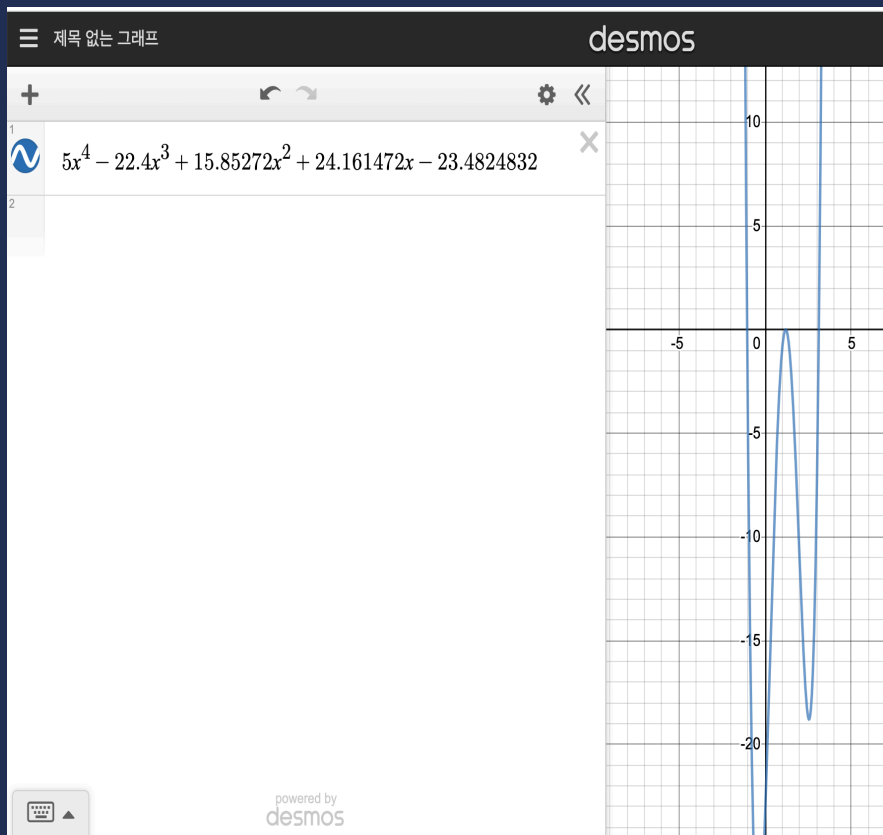
CONTENTS

- 
- 01 방정식의 그래프 및 min 유추하기
 - 02 exact first, second derivatives 사용하기
 - 03 approximation 사용하기
 - 4 두 방식 비교하기
 - 5 정답 비교 및 전체 코드

01

HW #2

그래프 및 근 유추하기



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

desmos 이용

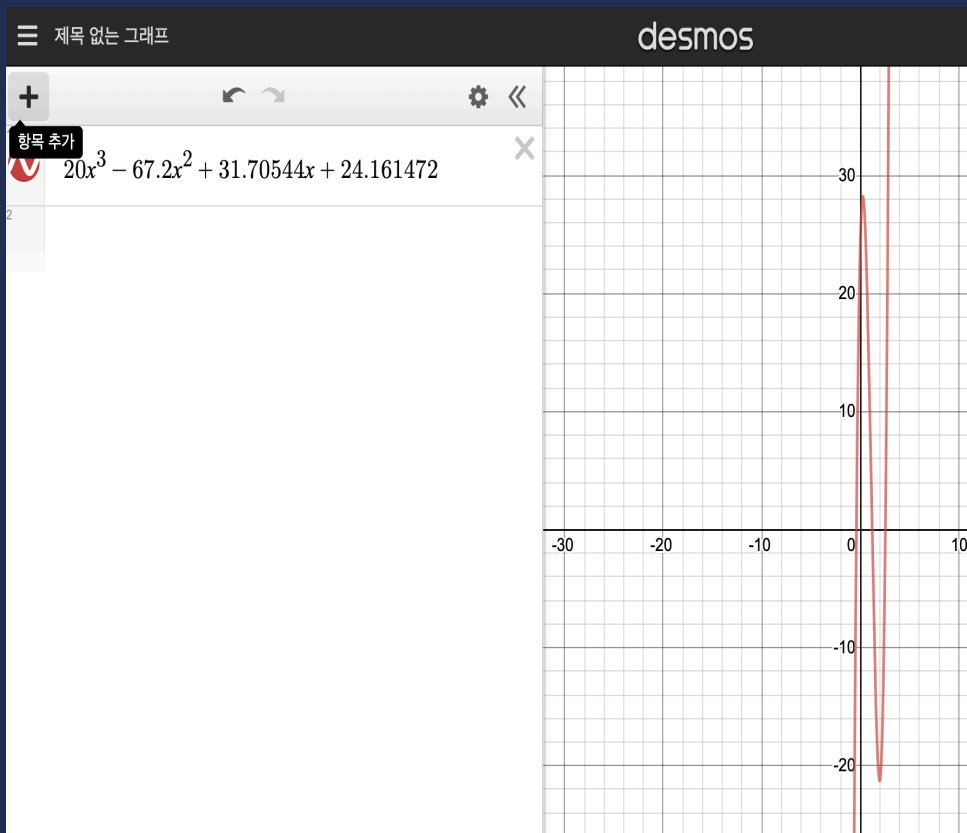
왼쪽 사진은 해당 수식의 그래프로써 전형적인 4차원 방정식의 그래프이다. 특이한 점으로는 2개의 일반적인 근과 1개의 중근을 갖는다는 점이다.

또한, 1개의 local max와 2개의 local min을 가지며, 2개의 local min 중 1개는 global min 값이다.

01

HW #2

그래프 및 근 유추하기



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

desmos 이용

왼쪽 사진은 해당 수식의 그래프로써 전형적인 3차원 방정식의 그래프이다. 앞선 슬라이드에서 보았듯이 미분하기 전 식의 그래프가 2개의 극솟값, 1개의 극댓값을 가지므로 미분한 해당 식의 그래프는 3개의 근을 가질 것이다. 이 근을 찾으면 원래 식의 극솟값, 극댓값을 찾을 수 있다.

02

HW #2

Exact first and second derivatives 코드

```
def y(x):  
    ret = 5*(x**4) - 22.4*(x**3) + 15.85272*(x**2) + 24.161472*(x) - 23.4824832  
    return ret  
def yy(x):  
    ret = 5*4*(x**3) - 22.4*3*(x**2) + 15.85272*2*(x) + 24.161472  
    return ret  
def yyy(x):  
    ret = 5*4*3*(x**2) - 22.4*3*2*(x) + 15.85272*2  
    return ret
```

y 함수는 과제에 명시된 polynomial equation을 표현한 식으로 parameter 값을 주면 해당 값을 return 합니다.

yy 함수는 y 함수를 미분한 함수로서 parameter 값을 주면 해당 미분값을 return 합니다.

yyy 함수는 yy 함수를 미분한 함수로서 y의 이계도함수로서 parameter 값을 주면 해당 값을 return 합니다.

02

HW #2

Exact first and second derivatives 코드

$$x_{i+1} = x_i - \alpha \frac{f^{(1)}(x_i)}{f^{(2)}(x_i)} + \Delta x_i$$

```
def first_method(point,t):  
    while abs(yy(point)/yyy(point)) >= t :  
        point = point - (yy(point)/yyy(point))  
    return point
```

첫번째 사진의 식을 반복해서 계산하면서 local min, max를 찾는 코드입니다. Exact first and second derivatives 이라는 말 답게 도함수와 이계도함수에는 이전 슬라이드에서 설명한 yy와 yyy를 사용하였습니다.

02

HW #2

Exact first and second derivatives 코드

```
t = 0.00001
print("first_method")
for i in range(-4,4,1):
    print("i : "+str(i)+", root :",first_method(i,t))
```

이후 main에서 t의 값을 적절히 설정해준 후 for 문을 돌며 첫번째 방식을 실행합니다. For문의 구간은 처음 그래프를 그리며 근을 추정할 때 나온 적절한 범위로 설정합니다.

02

HW #2

Exact first and second derivatives 결과

```
first_method
```

```
i : -4, root : -0.3941533180910058  
i : -3, root : -0.3941536411971322  
i : -2, root : -0.3941564872722486  
i : -1, root : -0.3941537211684232  
i : 0, root : -0.39415331757185373  
i : 1, root : 1.1999989750559639  
i : 2, root : 2.554153539864415  
i : 3, root : 2.5541533550211293
```

앞선 코드를 실행시킨 결과입니다.

Initial 값으로 -4, -3, -2, -1, 0, 1, 2, 3을 주어 첫 번째 방식을 실행합니다. -4 ~ 0까지 똑같은 근을 찾고, 1일 때 새로운 근, 마지막으로 2~3일 때 마지막 근을 찾는 것을 볼 수 있습니다. 숫자가 조금씩 다르긴 하지만 맨처음 그래프를 그려 근을 유추했기에 각각의 값이 어떤 근을 나타내는 지 충분히 짐작할 수 있습니다.

따라서 -0.39쯤과 2.55쯤에서 극솟값을 가지고 1.19쯤에서 극댓값을 가집니다.

03

HW #2

Approximation 코드

```
def y(x):  
    ret = 5*(x**4) - 22.4*(x**3) + 15.85272*(x**2) + 24.161472*(x) - 23.4824832  
    return ret  
✓ def yy2(x,h):  
    ret = ( y(x+h) - y(x) ) / (h)  
    return ret  
✓ def yyy2(x,h):  
    ret = ( y(x+h) - 2*y(x) + y(x-h) ) / (h**2)  
    return ret
```

y 함수는 과제에 명시된 polynomial equation을 표현한 식으로 parameter 값을 주면 해당 값을 return 합니다.

yy2 함수는 y 함수를 미분한 함수로서 parameter 값을 주면 해당 미분값을 기존의 방식이 아닌 approximation 방식으로 구해서 return 합니다.

yyy2 함수는 y의 이계도함수로서 parameter 값을 주면 해당값을 기존의 방식이 아닌 approximation 방식으로 구해서 return 합니다.

03

HW #2

Approximation 코드

$$x_{i+1} = x_i - \alpha \frac{f^{(1)}(x_i)}{f^{(2)}(x_i)} + \Delta x_i$$

```
def second_method(point,t,h):  
    while abs(yy2(point,h)/yyy2(point,h)) >= t :  
        point = point - (yy2(point,h)/yyy2(point,h))  
    return point
```

첫번째 사진의 식을 반복해서 계산하면서 local min, max를 찾는 코드입니다. Approximation이라는 말 답게 도함수와 이계도함수에는 이전 슬라이드에서 설명한 yy2와 yyy2를 사용하였습니다.

03

HW #2

Approximation 코드

```
t = 0.00001
h = 0.0001
print("second_method")
for i in range(-4,4,1):
    print("i : "+str(i)+", root :",second_method(i,t,h))
```

이후 main에서 t와 h의 값을 적절히 설정해준 후
for문을 돌며 두번째 방식을 실행합니다. For문
의 구간은 처음 그래프를 그리며 근을 추정할 때
나온 적절한 범위로 설정합니다.

03

HW #2

Approximation 결과

```
second_method
```

```
i : -4, root : -0.3942033215726233  
i : -3, root : -0.3942037003569945  
i : -2, root : -0.39420668166036854  
i : -1, root : -0.39420378691394636  
i : 0, root : -0.3942033203688336  
i : 1, root : 1.1999490097768053  
i : 2, root : 2.5541034871262642  
i : 3, root : 2.554103333207549
```

앞선 코드를 실행시킨 결과입니다.

Initial 값으로 -4, -3, -2, -1, 0, 1, 2, 3을 주어 두 번째 방식을 실행합니다. -4 ~ 0까지 똑같은 근을 찾고, 1일 때 새로운 근, 마지막으로 2~3일 때 마지막 근을 찾는 것을 볼 수 있습니다. 숫자가 조금씩 다르긴 하지만 맨처음 그래프를 그려 근을 유추했기에 각각의 값이 어떤 근을 나타내는 지 충분히 짐작할 수 있습니다.

따라서 -0.39쯤과 2.55쯤에서 극솟값을 가지고 1.19쯤에서 극댓값을 가집니다.

1. Use exact first and second derivatives : 본래의 함수에서 도함수와 이계도함수를 직접 계산해서 구하는 방법. closed-form 형태에서 주로 사용하며 직접 계산해야 하는 번거로움이 있지만 approximation 방법보다 오차가 적음

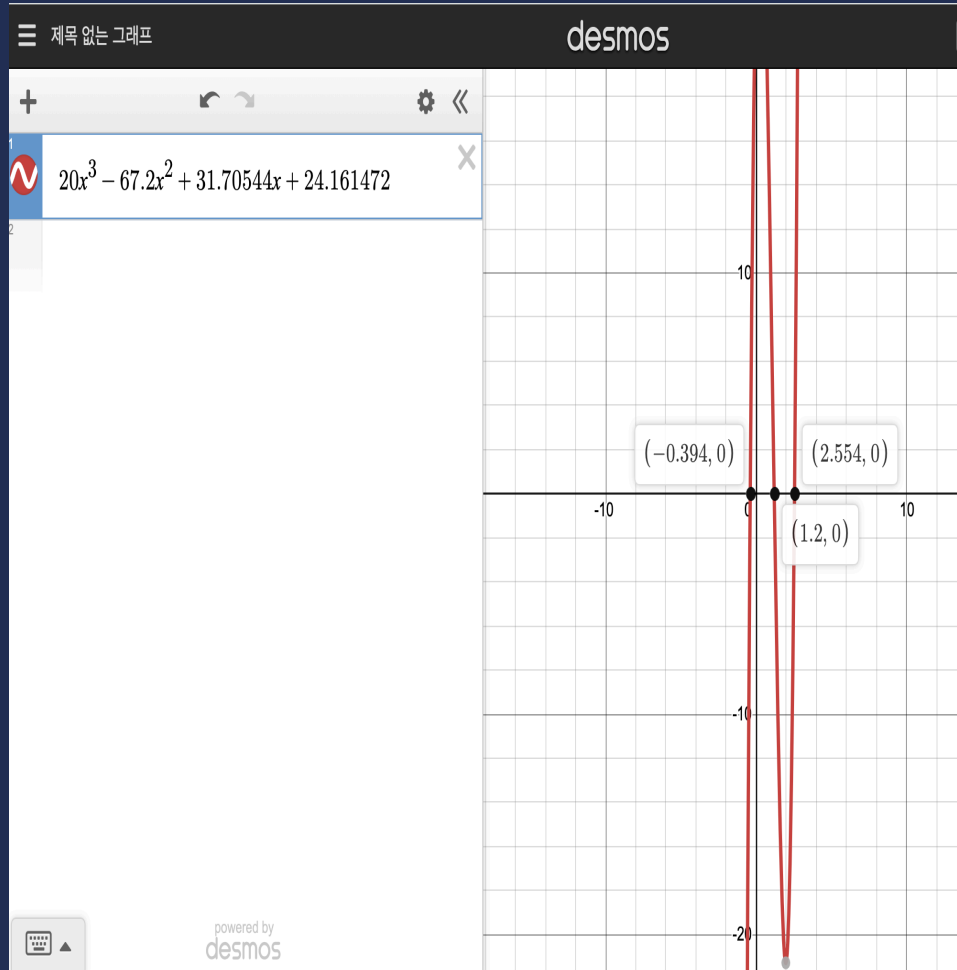
2. Approximation : 기울기를 구하는 식을 이용하여 도함수와 이계도함수를 본래의 함수로만 나타냄. 두 점 사이의 거리를 h 로 두어 이 h 값이 0에 가까워질수록 해당 점에서 미분한 것으로 생각 할 수 있다.

3. 두 방식의 비교 : Approximation 방법에서 사용되는 h 값의 크기가 줄어들수록 use exact first and second derivatives로 구한 값에 가까워 지는 것을 확인 할 수 있었다. 하지만 h 값을 너무 작게 설정해 버리면 답을 구하는 과정에서 파이썬 내부의 수 처리 방식에 의해 도함수를 이계도함수로 나눈 식에서 0으로 나누게 되는 일이 생겨 런타임 에러가 발생한다. 하지만 이렇게 너무 작은 수가 아니더라도 충분히 의미있는 수를 답으로 구할 수 있기 때문에 approximation 방법도 매우 유용하다. 특히 직접 미분 할 수 없는 $f(x)$ 라면 approximation 방식이 더 의미가 있다.

05

HW #2

정답 비교



```
first_method
i : -4, root : -0.3941533180910058
i : -3, root : -0.3941536411971322
i : -2, root : -0.3941564872722486
i : -1, root : -0.3941537211684232
i : 0, root : -0.39415331757185373
i : 1, root : 1.1999989750559639
i : 2, root : 2.554153539864415
i : 3, root : 2.5541533550211293
```

```
second_method
i : -4, root : -0.3942033215726233
i : -3, root : -0.3942037003569945
i : -2, root : -0.39420668166036854
i : -1, root : -0.39420378691394636
i : 0, root : -0.3942033203688336
i : 1, root : 1.1999490097768053
i : 2, root : 2.5541034871262642
i : 3, root : 2.554103333207549
```

위 결과는 h를 0.001로
아래 결과는 h를 0.0000001로

```
second_method
i : -4, root : -0.39415320701681794
i : -3, root : -0.3941539566215469
i : -2, root : -0.39415958545656793
i : -1, root : -0.3941579121732268
i : 0, root : -0.3941532535012693
i : 1, root : 1.2000011301194775
i : 2, root : 2.5541598400516574
i : 3, root : 2.554148279587534
```

05

HW #2

Python 전체 코드

```
def y(x):
    ret = 5*(x**4) - 22.4*(x**3) + 15.85272*(x**2) + 24.161472*(x) - 23.4824832
    return ret
def yy(x):
    ret = 5*4*(x**3) - 22.4*3*(x**2) + 15.85272*2*(x) + 24.161472
    return ret
def yyy(x):
    ret = 5*4*3*(x**2) - 22.4*3*2*(x) + 15.85272*2
    return ret
def yy2(x,h):
    ret = ( y(x+h) - y(x) ) / (h)
    return ret
def yyy2(x,h):
    ret = ( y(x+h) - 2*y(x) + y(x-h) ) / (h**2)
    return ret

def first_method(point,t):
    while abs(yy(point)/yyy(point)) >= t :
        point = point - (yy(point)/yyy(point))
    return point

def second_method(point,t,h):
    while abs(yy2(point,h)/yyy2(point,h)) >= t :
        point = point - (yy2(point,h)/yyy2(point,h))
    return point

t = 0.00001
h = 0.0001
print("first_method")
for i in range(-4,4,1):
    print("i : "+str(i)+" , root :",first_method(i,t))
print("\nsecond_method")
for i in range(-4,4,1):
    print("i : "+str(i)+" , root :",second_method(i,t,h))
```

감사합니다

THANK YOU

JaeHoon KANG

강 재 훈