

수치해석 HW #1

Numerical analysis

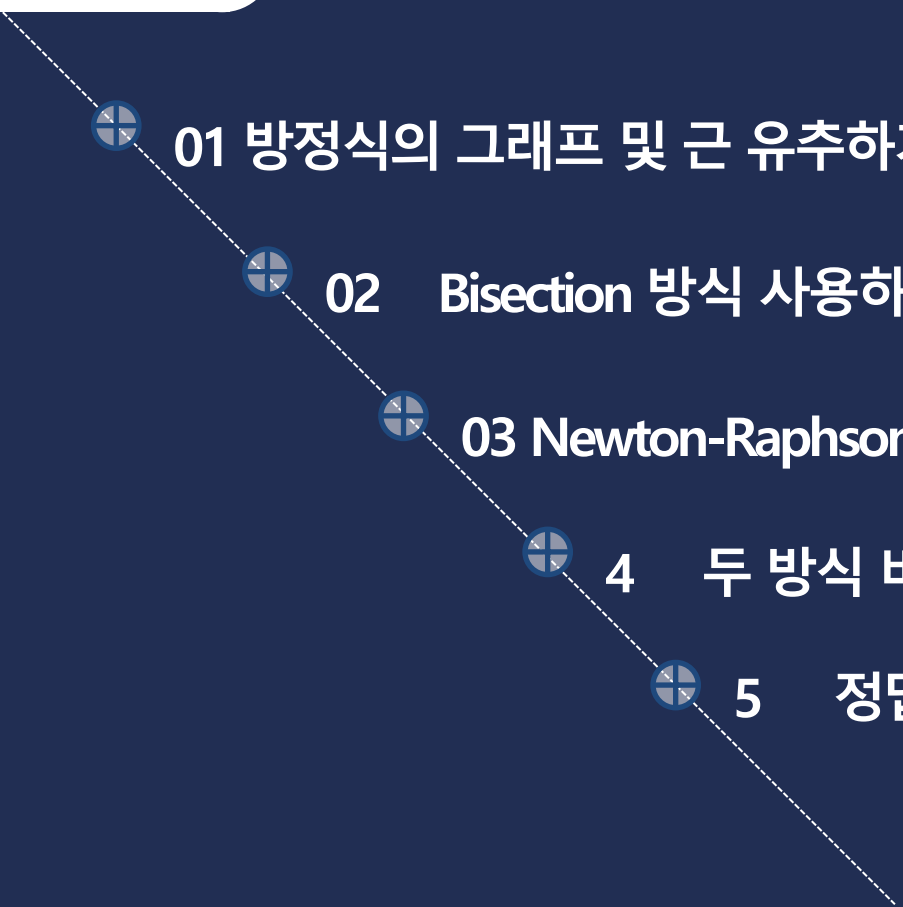
JaeHoon KANG

강 재 훈

HW #1

Find the roots of the polynomial equation

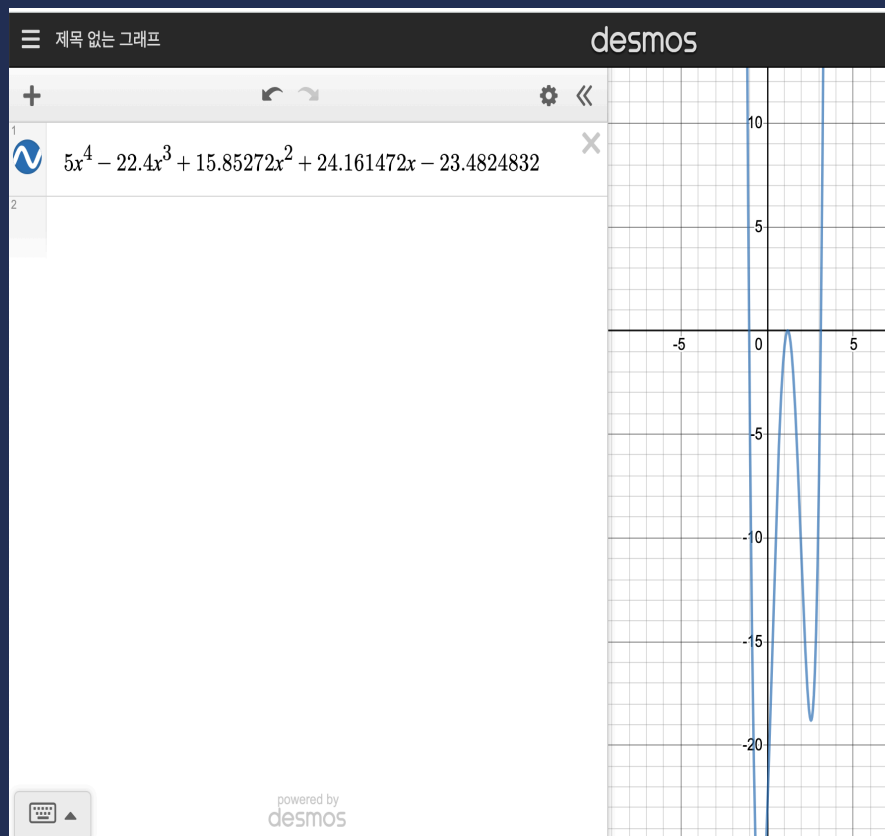
CONTENTS

- 
- 01 방정식의 그래프 및 근 유추하기
 - 02 Bisection 방식 사용하기
 - 03 Newton-Raphson 방식 사용하기
 - 4 두 방식 비교하기
 - 5 정답 비교 및 전체 코드

01

HW #1

그래프 및 근 유추하기



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

desmos 이용

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

Bisection 방식으로 근을 구할때 적절한 구간을 설정하기 위해,

Newton-Rhaphson 방식으로 근을 구할때 적절한 initial 값을 설정하기 위해 그 그래프를 그려보았다.

02

HW #1

Bisection 방식 코드

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

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

```
def bisection(left, right, t):  
    if y(left) * y(right) > 0 :  
        pass  
    else :  
        while (right-left)/2.0 > t :  
            mid = (right+left)/2.0  
            if y(mid) == 0 :  
                return mid  
            elif y(left) * y(mid) < 0 :  
                right = mid  
            else :  
                left = mid  
        return mid
```

Bisection을 구현한 함수입니다.

Bisection의 양 구간 값을 left, right 인자 값으로 받고 stop condition으로 사용할 값을 t로 받습니다. 수업에서 배운대로 Bisection의 양쪽 구간의 끝의 값이 같은 부호라면 그 구간에는 근이 없으므로 pass 합니다. $(right-left)/2.0$ 이 설정한 t만큼보다 클때만 loop를 계속 돌며 근을 찾습니다. 즉 구간이 매우 작아질때까지 loop를 돌며 근을 찾습니다.

02

HW #1

Bisection 방식 코드

```
t = 0.00001  
print("Bisection")  
for i in range(-4,4):  
    print(i,bisection(i,i+1,t))
```

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

02

HW #1

Bisection 방식 결과

```
Bisection
i : -4, root : None
i : -3, root : None
i : -2, root : -1.0439910888671875
i : -1, root : None
i : 0, root : None
i : 1, root : None
i : 2, root : None
i : 3, root : 3.1240081787109375
```

앞선 코드를 실행시킨 결과입니다.

i 의 값부터 $i+1$ 의 값까지의 구간으로 bisection
을 진행하고 그 구간에 근이 있을 경우 근을 출
력합니다.

수업시간에 배운대로 bisection 방식으로는 중근
을 구해낼 수 없었고 2개의 일반근과 1개의 중근
중 2개의 일반근만 검출된 모습입니다.

03

HW #1

Newton-Raphson 방식 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
```

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.

03

HW #1

Newton-Raphson 방식 코드

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

Newton-Raphson 방식을 구현한 함수입니다.
수업시간에 배운대로 $\Delta x(f(x)/f'(x))$ 가 매우 작아지는 순간을 stop condition으로 사용하였습니다.

03

HW #1

Newton-Raphson 방식 코드

```
t = 0.00001  
print("Newton_raphson")  
for i in range(-4,5,2):  
    print("i : "+str(i)+", root :",newton_raphson(i,t))
```

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

03

HW #1

Newton-Raphson 방식 결과

```
Newton_raphson
```

```
i : -4, root : -1.0440000781152665  
i : -2, root : -1.0440000545594397  
i : 0, root : 1.1999860663401256  
i : 2, root : 1.2000165313075668  
i : 4, root : 3.1240000809790938
```

앞선 코드를 실행시킨 결과입니다.

Initial 값으로 -4, -2, 0, 2, 4를 주어 Newton-Raphson 방식을 실행합니다. -4와 -2에서 첫 번째 일반근을, 0과 2에서 1개의 중근을, 4에서 두 번째 일반근을 찾는 모습을 볼 수 있습니다. 숫자가 조금씩 다르긴 하지만 맨처음 그래프를 그려 근을 유추했기에 각각의 값이 어떤 근을 나타내는지 충분히 짐작할 수 있습니다.

두 방식 비교하기

1. Bisection : 구간법(Bracketing method) 중 한가지의 방법으로서 근이 존재하는 구간을 정하여 근을 추정, 특히 구간을 반으로 나눈 소구간이 다음 반복을 위한 새로운 구간이 됨.

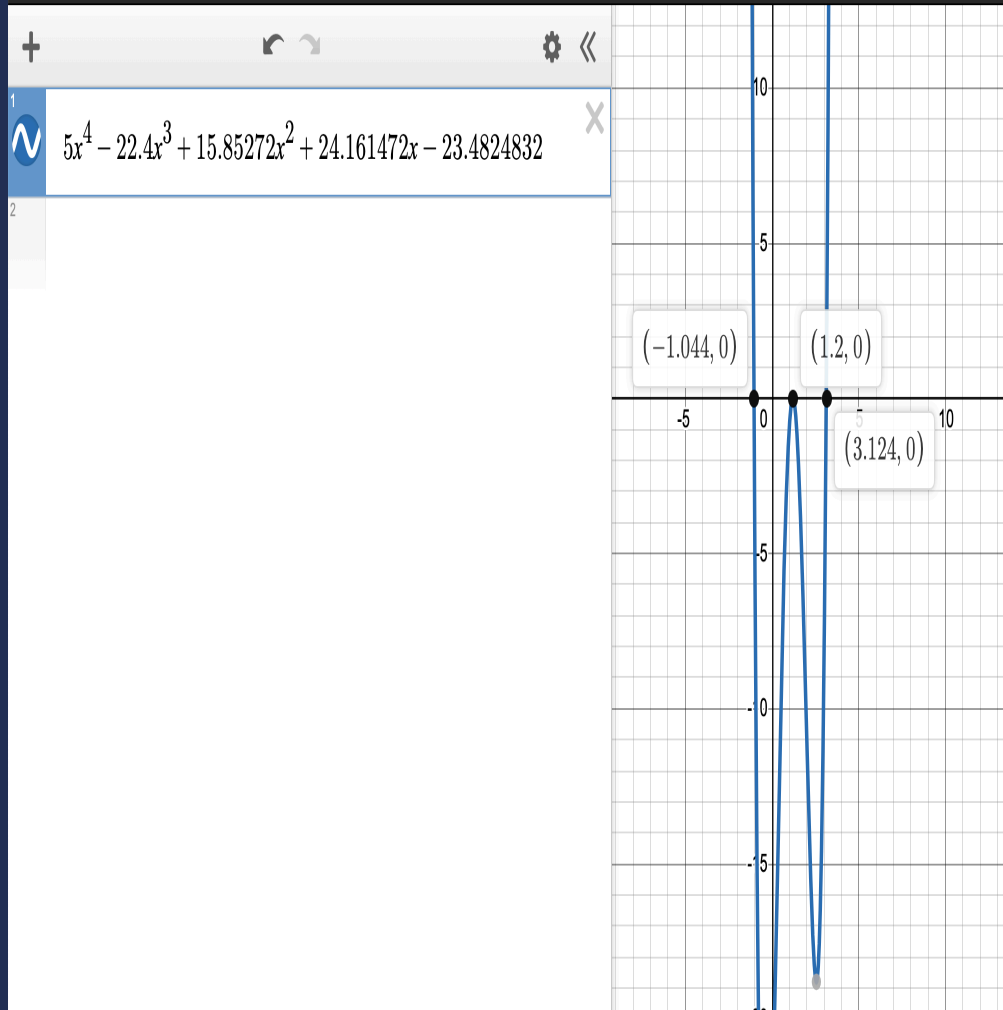
2. Newton-Raphson : 개방법 (Open method) 중 한가지 방법으로서 임의 선택점을 고르고 점차 근을 유추해 나간다. 점 $(x_0, f(x_0))$ 에서의 접선을 구하고 그 접선이 x 축과 만나는 점을 x_1 이라 하면 다시 $(x_1, f(x_1))$ 에서의 접선을 이용하는 방식을 반복해 근을 구한다.

3. 두 방식의 차이 : 일단 bisection은 Initial 구간을 정하기 위해 구간 양 끝의 2개의 값을 설정해야 한다. 그에 반해 Newton-Raphson 방식은 Initial 점 1개의 값만 설정해주면 된다. Bisection의 구간은 근이 있을 법한 구간 즉 구간의 양쪽 끝의 함수값의 부호가 다른 구간을 설정해주어야 한다. 이러한 구간설정법으로 인해 bisection 방식에 의해서 중근을 찾아낼 수 없다. Newton-Raphson 방식에서도 Initial 점을 신중하게 고르는 노력이 필요한데 근의 위치와 너무 무관한 위치를 고르면 안되기 때문이다. 일반적으로 Newton-Raphson 방식이 bisection 방식보다 빠르긴 하나 각각의 방식이 가지는 치명적인 단점이 있으므로 두 방식을 시의적절하게 사용해야 한다.

05

HW #1

정답 비교



Bisection

```
i : -4, root : None
i : -3, root : None
i : -2, root : -1.0439910888671875
i : -1, root : None
i : 0, root : None
i : 1, root : None
i : 2, root : None
i : 3, root : 3.1240081787109375
```

Newton_raphson

```
i : -4, root : -1.0440000781152665
i : -2, root : -1.0440000545594397
i : 0, root : 1.1999860663401256
i : 2, root : 1.2000165313075668
i : 4, root : 3.1240000809790938
```

05

HW #1

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 bisection(left, right, t):
    if y(left) * y(right) > 0 :
        pass
    else :
        while (right-left)/2.0 > t :
            mid = (right+left)/2.0
            if y(mid) == 0 :
                return mid
            elif y(left) * y(mid) < 0 :
                right = mid
            else :
                left = mid
        return mid

def newton_raphson(point,t):
    while abs(y(point)/yy(point)) >= t :
        point = point - (y(point)/yy(point))
    return point

t = 0.00001
print("Bisection")
for i in range(-4,4):
    print("i : "+str(i)+", root :",bisection(i,i+1,t))
print()
print("Newton_raphson")
for i in range(-4,5,2):
    print("i : "+str(i)+", root :",newton_raphson(i,t))
```

감사합니다

THANK YOU

JaeHoon KANG

강 재 훈