수치해석 HW#2

Newton Method를 이용한 Min이 되는 값 찾기

Python 사용

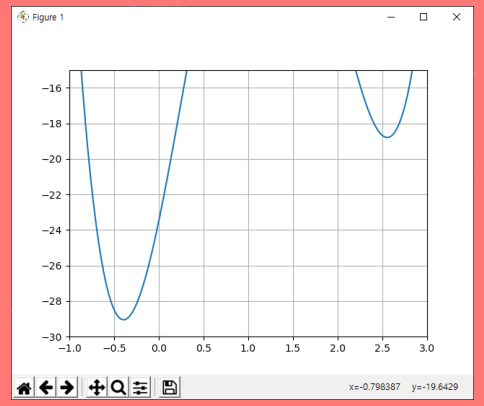
컴퓨터소프트웨어학부 2015005187 최철훈

목차

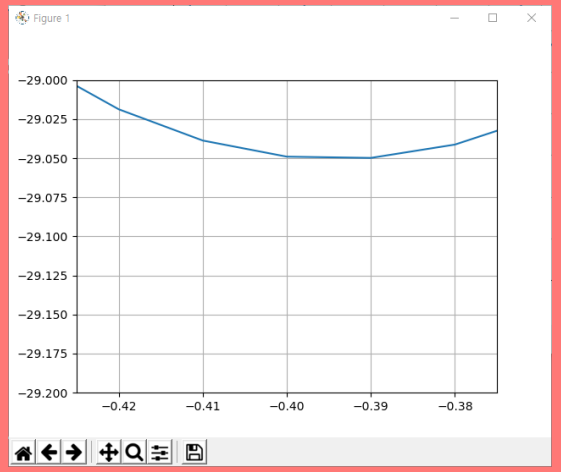
최솟값이 되는 x추측하기

먼저, f(x) = $5x^4 - 22.4x^3 + 15.85272x^2 + 24.161472x - 23.4824832$의 최솟값이 되는 x를 추측하기 위해 그래프를 그렸다.

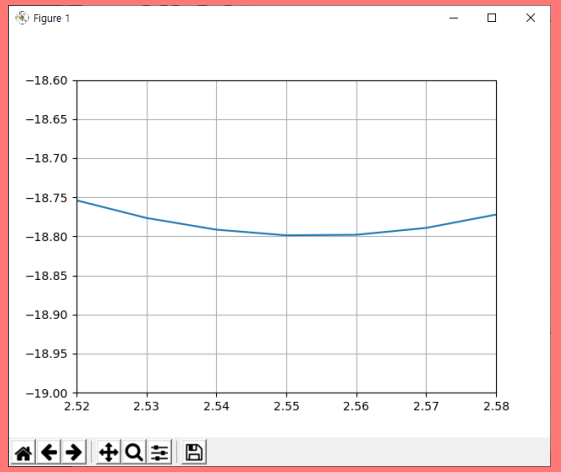
```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-2, 5, 0.01)
y = 5*x**4 - 22.4*x**3 + 15.85272*x**2 + 24.161472*x - 23.4824832

plt.figure()
plt.plot(x, y)
plt.grid()
plt.xlim(-1, 3)
plt.ylim(-30, -15)
plt.show()
```




```
plt.xlim(-0.425, -0.375)
plt.ylim(-29.2, -29)
```




```
plt.xlim(2.52, 2.58)
plt.ylim(-19, -18.6)
```



이와 같이 두 개의 최솟값이 있으며 1개는 -0,4와 -0.39사이에, 다른 1개는 2.55와 2.56사이에 존재한다.

Newton Method를 이용하여 x구하기 코드

Newton Method를 구현한 코드이다. α는 1로 설정하였고 모멘텀은 주지 않았다. 정확한 최솟값을 갖는 x를 찾지 못하면 x의 미분계수가 0.00000001보다 작을 때의 x_i를 반환하도록 하였다.

```
init = float(input("Input initial value : "))
x = symbols('x')
fx = 5*x**4 - 22.4*x**3 + 15.85272*x**2 + 24.161472*x - 23.4824832
fd = Derivative(fx, x)
fdev = fd.doit()
sd = Derivative(fdev, x)
sdev = sd.doit()
while True:
    f1 = fdev.subs({x: init})
    f2 = sd.doit().subs({x: init})
    if f1 == 0:
        print("The min location is : ", init)
        break
    elif f1 != 0:
        nv = init - (f1/f2)
        if abs(f1) < 0.00000001:
            print("The min location is : ", init)
            break
        init = nv
```

Newton Method를 이용하여 x구하기 결과

각각 추측한 x값과 가장 가까운 정수 2개를 택하여 initial value로 설정하였다.

```
We will find the min locations fx = 5x^4 - 22.4x^3 + 15.85272x^2 + 24.161472x - 23.4824832
What method do you want?
Using first and second derivatives : 1, Using approximation : 2
Your Choice : 1
Your choice is Using derivatives
Input initial value : 0
The min location is :  -0.394153316314148
```

```
We will find the min locations fx = 5x^4 - 22.4x^3 + 15.85272x^2 + 24.161472x - 23.4824832
What method do you want?
Using first and second derivatives : 1, Using approximation : 2
Your Choice : 1
Your choice is Using derivatives
Input initial value : -1
The min location is :  -0.394153316314306
```

첫번째 최솟값이 되는 x는 -0.394153316314로 수렴한다.

```
We will find the min locations fx = 5x^4 - 22.4x^3 + 15.85272x^2 + 24.161472x - 23.4824832
What method do you want?
Using first and second derivatives : 1, Using approximation : 2
Your Choice : 1
Your choice is Using derivatives
Input initial value : 2
The min location is :  2.55415331631420
```

```
We will find the min locations fx = 5x^4 - 22.4x^3 + 15.85272x^2 + 24.161472x - 23.4824832
What method do you want?
Using first and second derivatives : 1, Using approximation : 2
Your Choice : 1
Your choice is Using derivatives
Input initial value : 3
The min location is :  2.55415331631415
```

두번째 최솟값이 되는 x는 2.554153316314로 수렴한다.

도함수를 Approximation하여 x구하기 코드

도함수를 Approximation하여 최솟값이 되는 x를 찾는 방법을 구현한 코드이다. h를 0.0000001로 설정하였다. 앞의 Newton Method와 동일하게 α는 1로 설정하였고 모멘텀은 주지 않았다. 정확한 최솟값을 갖는 x를 찾지 못하면 x의 미분계수가 0.0000001보다 작을 때의 x_i를 반환하도록 하였다.

```
xi = float(input("Input initial value : "))
h = 0.0000001
x = symbols('x')
fx = 5*x**4 - 22.4*x**3 + 15.85272*x**2 + 24.161472*x - 23.4824832
fd = Derivative(fx, x)
fdev = fd.doit()
sd = Derivative(fdev, x)
sdev = sd.doit()
while True:
    ximh = xi - h
    xiph = xi + h
    fxi = fx.subs({x: xi})
    fxiph = fx.subs({x: xiph})
    fximh = fx.subs({x: ximh})
    f1xi = (fxiph - fxi)/h
    f1ximh = (fxi - fximh)/h
    f2xi = (f1xi - f1ximh)/h
    if f1xi == 0:
        print("The min location is : ", xi)
        break
    elif f1xi != 0:
        xip1 = xi - f1xi/f2xi
        if abs(f1xi) < 0.0000001:
            print("The min location is : ", xi)
            break
        xi = xip1
```

도함수를 Approximation하여 x구하기 결과

각각 추측한 x값과 가장 가까운 정수 2개를 택하여 initial value로 설정하였다.

```
We will find the min locations fx = 5x^4 - 22.4x^3 + 15.85272x^2 + 24.161472x - 23.4824832
What method do you want?
Using first and second derivatives : 1, Using approximation : 2
Your Choice : 2
Your choice is Using approximation
Input initial value : 0
The min location is :  -0.394153365523111
```

```
We will find the min locations fx = 5x^4 - 22.4x^3 + 15.85272x^2 + 24.161472x - 23.4824832
What method do you want?
Using first and second derivatives : 1, Using approximation : 2
Your Choice : 2
Your choice is Using approximation
Input initial value : -1
The min location is :  -0.394153366292683
```

첫번째 최솟값이 되는 x는 -0.39415336으로 수렴한다.

```
We will find the min locations fx = 5x^4 - 22.4x^3 + 15.85272x^2 + 24.161472x - 23.4824832
What method do you want?
Using first and second derivatives : 1, Using approximation : 2
Your Choice : 2
Your choice is Using approximation
Input initial value : 2
The min location is :  2.55415326985379
```

```
We will find the min locations fx = 5x^4 - 22.4x^3 + 15.85272x^2 + 24.161472x - 23.4824832
What method do you want?
Using first and second derivatives : 1, Using approximation : 2
Your Choice : 2
Your choice is Using approximation
Input initial value : 3
The min location is :  2.55415326580263
```

두번째 최솟값이 되는 x는 2.55415326으로 수렴한다.

Discussion 두 방법의 비교

1. Newton Method가 도함수를 Approximation하는 방법보다 더 정확한 값으로 수렴함을 알 수 있다.

2. 도함수를 Approximation하는 방법은 h값이 작아질수록 정확한 값을 찾을 수 있다.

Discussion α값과 모멘텀

Newton Method로 α값과 모멘텀을 주었다. 결과를 비교하기 위해 Initial value를 2로 하였을 때의 x_{i+1}을 순서대로 출력해보았다.

1. α값으로 2를 대입해보았다. x_{i+1}가 진동하는 것을 확인할 수 있다.

```
nv = init - (f1/f2)
```

```
Input initial value : 2
9.30617324742537
6.63634488329099
4.88549443603561
3.76149258193196
3.07623321409993
2.71080969828306
2.57507585829291
2.55460895082265
2.55415353986441
2.55415331631420
2.55415331631415
The min location is :  2.55415331631420
```

```
nv = init - 2 * (f1/f2)
```

...

...

```
2.60885103147588
2.50535866827925
2.60854137876036
2.50560446583855
2.60823685893092
2.50584649877655
2.60793733230134
2.50608486321364
2.60764266444025
2.50631965184748
2.60735272591995
2.50655095410876
2.60706739207966
2.50677885630844
2.60678654280243
2.50700344177692
```

...

...

Discussion α값과 모멘텀

2. 모멘텀으로 $(x_{i+1} - x_i)/2$를 대입해보았다. 좀 더 천천히 수렴하는 것을 확인할 수 있다.

```
nv = init - (f1/f2)
```

```
Input initial value : 2
9.30617324742537
6.63634488329099
4.88549443603561
3.76149258193196
3.07623321409993
2.71080969828306
2.57507585829291
2.55460895082265
2.55415353986441
2.55415331631420
2.55415331631415
The min location is :  2.55415331631420
```

```
nv = init - (f1/f2)
dx = abs(nv - init)/2
nv = nv + dx
```

```
Input initial value : 2
12.9592598711381
11.0063884596814
9.38301906075518
8.03502148687626
6.91743768023199
5.99298030070885
5.23078522700855
4.60537417053936
4.09578727855056
3.68484359787354
3.35847574197091
3.10506126601152
2.91464899571403
2.77801428551659
2.68569536684475
2.62755521357670
2.59343731169874
2.57457510057784
2.56458142854839
2.55942495224196
2.55688397600896
2.55548241504556
2.55481881537934
2.55448630421749
2.55431986997744
2.55423660808861
2.55419496593895
2.55417414206117
2.55416372942135
2.55415852292617
2.55415591963477
2.55415461797811
2.55415396714704
2.55415364173082
2.55415347902254
2.55415339766836
2.55415335699126
2.55415333665270
2.55415332648343
2.55415332139879
2.55415331885647
2.55415331758531
2.55415331694973
2.55415331663194
2.55415331647304
2.55415331639359
2.55415331635387
The min location is :  2.55415331639359
```

Discussion α값과 모멘텀

3. α값으로 2를, 모멘텀으로 $(x_{i+1} - x_i)/2$를 대입해보았다. α값이 2일 때, 진동하던 것을 모멘텀을 넣음으로써 수렴하도록 만드는 것을 확인할 수 있다.

```
nv = init - 2 * (f1/f2)
```

...

...

```
2.60885103147588
2.50535866827925
2.60854137876036
2.50560446583855
2.60823685893092
2.50584649877655
2.60793733230134
2.50608486321364
2.60764266444025
2.50631965184748
2.60735272591995
2.50655095410876
2.60706739207966
2.50677885630844
2.60678654280243
2.50700344177692
```

...

...

```
nv = init - 2 * (f1/f2)
dx = abs(nv - init)/2
nv = nv + dx
```

```
Input initial value : 2
23.9185197422761
16.3401600352686
11.2984214411719
7.95293696190464
5.74598123476699
4.30948339551932
3.40359743227759
2.87571400420985
2.62648761720727
2.55917827191486
2.55418029791347
2.55415331709865
2.55415331631415
The min location is :  2.55415331631415
```

마무리

감사합니다.

https://github.com/cheol-hoon/Numerical_Analysis