Programming 4-2 Solution

전기공학전공 홍종혁

Problem 1

전체 코드

```
import numpy as np                                  #numpy np라는 이름으로 호출
from matplotlib import pyplot as plt                #matplotlib의 pyplot plt라는 이름으로 호출
x=np.linspace(-5,25,100)                            #x는 linspace를 활용 100개의 point 저장
f=(1/np.sqrt(50*(np.pi)))*(np.exp((-(x-10)**2)/50)) #f는 주어진 수식

###Forward Method###
ff1=np.zeros(99)
for i in range(99):                                 #99회 반복
    ff1[i]=(f[i+1]-f[i])/(x[i+1]-x[i])              #ff1의 i번째에 Forward Method 수행한 값 저장
plt.figure()                                        #figure 생성
plt.plot(x[:-1],ff1)                                #x축을 x[:-1], y축을 ff1으로 하여 plot
plt.xlim(-5,25)                                     #x축을 -5부터 25까지로 제한
plt.xlabel("x")                                     #x축의 이름을 'x'로 설정
plt.ylabel("y")                                     #y축의 이름을 'y'로 설정
plt.title("Forward Method")                         #figure의 제목을 'Forward Method'로 설정
plt.show()                                          #plot 시행

###Backward Method###
ff2=np.zeros(99)                                    #차분 값을 저장할 ff2를 선언
for i in range(1,100):                              #99회 반복
    ff2[i-1]=(f[i]-f[i-1])/(x[i]-x[i-1])            #ff2의 i-1번째에 Backward Method 수행한 값 저죠
plt.figure()                                        #figure 생성
plt.plot(x[:-1],ff2)                                #x축을 x[:-1], y축을 ff2으로 하여 plot
plt.xlim(-5,25)                                     #x축을 -5부터 25까지로 제한
plt.xlabel("x")                                     #x축의 이름을 'x'로 설정
plt.ylabel("y")                                     #y축의 이름을 'y'로 설정
plt.title("Backward Method")                        #figure의 제목을 'Backward Method'로 설정
plt.show()                                          #plot 시행

###Central Method###
ff3=np.zeros(98)                                    #차분 값을 저장할 ff3을 선언
for i in range(98):                                 #98회 반복
    ff3[i]=(f[i+2]-f[i])/(x[i+2]-x[i])              #ff3의 i번째에 Central Method 수행한 값 저장
plt.figure()                                        #figure 생성
plt.plot(x[:-2],ff3)                                #x축을 x[:-2], y축을 ff3으로 하여 plot
plt.xlim(-5,25)                                     #x축을 -5부터 25까지로 제한
plt.xlabel("x")                                     #x축의 이름을 'x'로 설정
plt.ylabel("y")                                     #y축의 이름을 'y'로 설정
plt.title("Central Method")                         #figure의 제목을 'Central Method'로 설정
plt.show()                                          #plot 시행

###Python Gradient Function###
ff4=np.gradient(f,x)                                #gradient 함수를 사용하여 얻은 값 ff4에 저장
plt.figure()                                        #figure 생성
plt.plot(x,ff4)                                     #x를 x축으로, ff4를 y축으로 하여 plot
plt.xlim(-5,25)                                     #x축을 -5부터 25까지로 제한
plt.xlabel("x")                                     #x축의 이름을 'x'로 설정
plt.ylabel("y")                                     #y축의 이름을 'y'로 설정
plt.title("Python Gradient")                        #figure의 제목을 'Python Gradient'로 설정
plt.show()                                          #plot 시행
```

```python
import numpy as np
from matplotlib import pyplot as plt
x=np.linspace(-5,25,100)
f=(1/np.sqrt(50*(np.pi)))*(np.exp((-(x-10)**2)/50))
```

numpy모듈을 np라는 이름으로 설정하여 호출하고 matplotlib에서 pyplot을 plt라는 이름으로 설정하여 호출한다. x는 linspace 함수를 활용하여 -5에서 25사이에서 100개의 point를 설정하였다. f는 주어진 함수의 수식이다.

(1) Use the forward method to differentiate f with respect to the 100 points you selected and plot the differentiation.

```python
###Forward Method###
ff1=np.zeros(99)
for i in range(99):
    ff1[i]=(f[i+1]-f[i])/(x[i+1]-x[i])
plt.figure()
plt.plot(x[:-1],ff1)
plt.xlim(-5,25)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Forward Method")
plt.show()
```

Forward Method를 반복 사용하여 미분 그래프를 plot하는 알고리즘이다. Forward Method의 수식은 $f'(x_k) = \frac{f(x_{k+1})-f(x_k)}{x_{k+1}-x_k}$이고, 이것을 for문을 활용하여 처음 point부터 마지막 point까지 반복하였다.

이후 figure를 생성하고 f'의 range에 맞게 x축과 y축을 지정해 준 후 plot한다.

(2) Use the backward method to differentiate f with respect to the 100 points you selected and plot the differentiation.

```python
###Backward Method###
ff2=np.zeros(99)
for i in range(1,100):
    ff2[i-1]=(f[i]-f[i-1])/(x[i]-x[i-1])
plt.figure()
plt.plot(x[:-1],ff2)
plt.xlim(-5,25)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Backward Method")
plt.show()
```

Backward Method를 반복 사용하여 미분 그래프를 plot하는 알고리즘이다. Backward Method의 수식은$f'(x_k) = \frac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}$이고, 이것을 for문을 활용하여 처음 point부터 마지막 point까지 반복하였다.

이후 figure를 생성하고 f'의 range에 맞게 x축과 y축을 지정해 준 후 plot한다.

(3) Use the central method to differentiate f with respect to the 100 points you selected and plot the differentiation.

```
###Central Method###
ff3=np.zeros(98)
for i in range(98):
    ff3[i]=(f[i+2]-f[i])/(x[i+2]-x[i])
plt.figure()
plt.plot(x[:-2],ff3)
plt.xlim(-5,25)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Central Method")
plt.show()
```

Central Method를 반복 사용하여 미분 그래프를 plot하는 알고리즘이다. Central Method의 수식은 $f'(x_k) = \frac{f(x_{k+1})-f(x_{k-1})}{x_{k+1}-x_{k-1}}$이고, 이것을 for문을 활용하여 처음 point부터 마지막 point까지 반복하였다.

이후 figure를 생성하고 f'의 range에 맞게 x축과 y축을 지정해 준 후 plot한다.

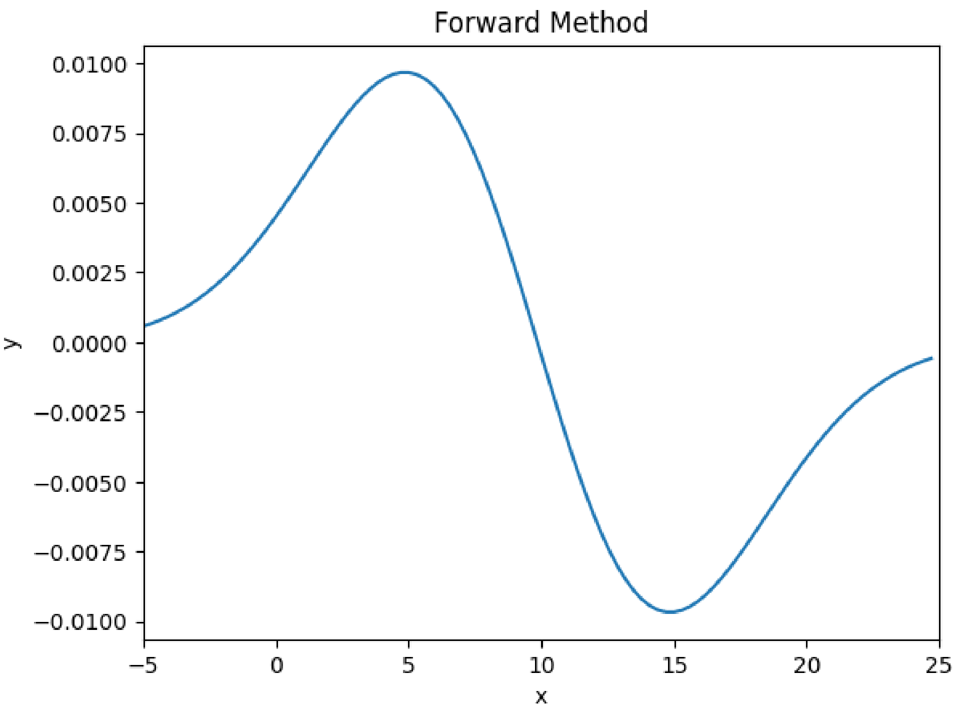
(4) Use any Python numerical differential method to differentiate f with respect to the 100 points you selected and plot the differentiation.

```
###Python Gradient Function###
ff4=np.gradient(f,x)
plt.figure()
plt.plot(x,ff4)
plt.xlim(-5,25)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Python Gradient")
plt.show()
```

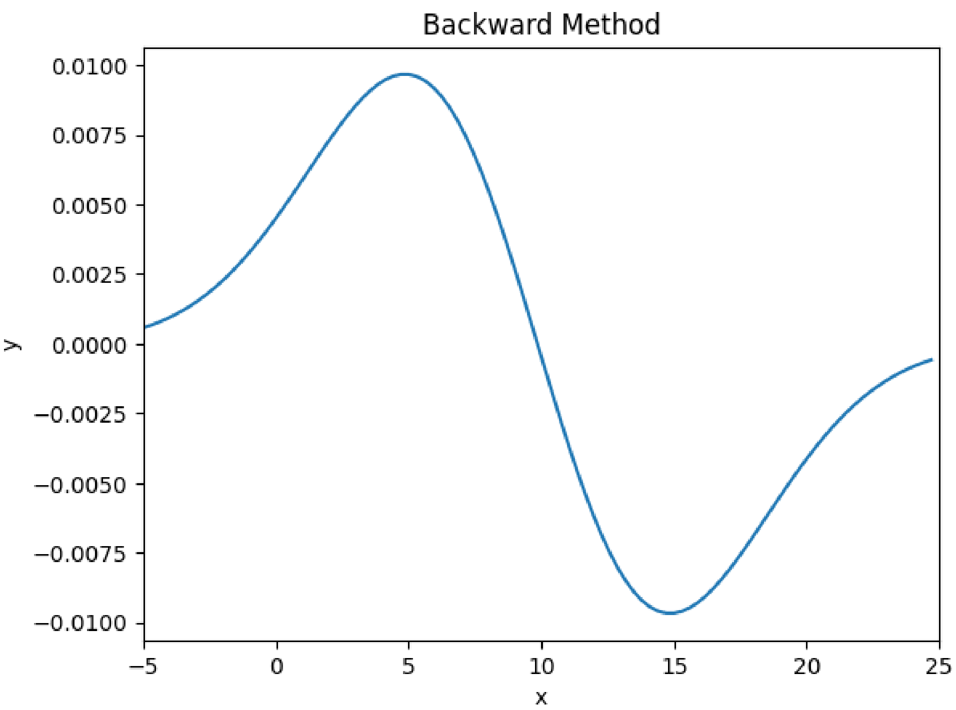
numpy 모듈 내의 gradient 함수를 사용하는 방법이다. 미분할 함수로 f, 미분 변수를 x로 지정(실제로는 차분이다.)해 준 후 범위에 맞게 plot한다.

(5) Compare each numerical differentiation results.

- Forward Method

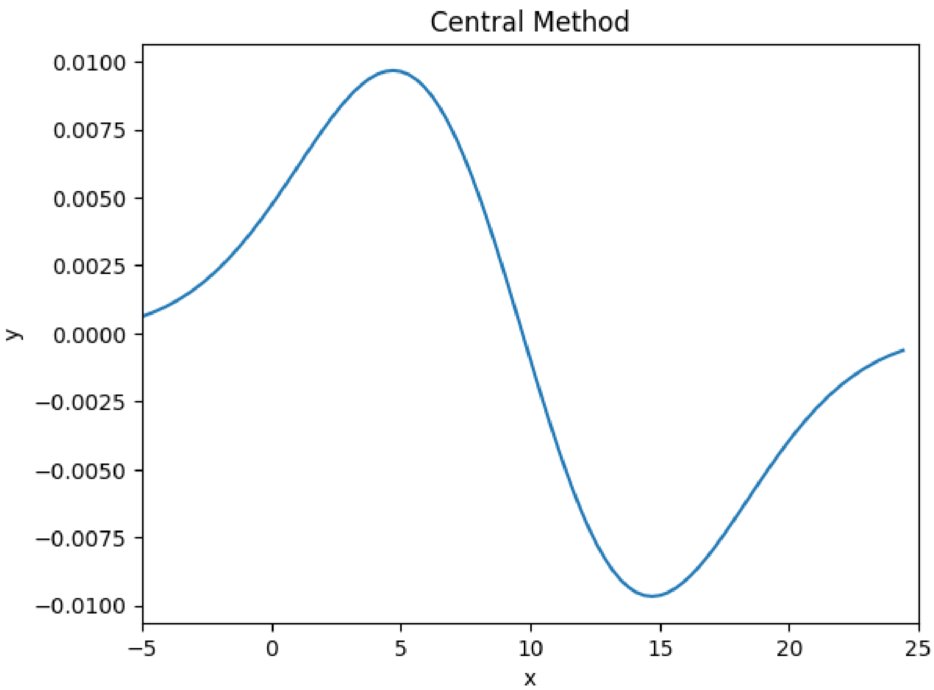


- Backward Method

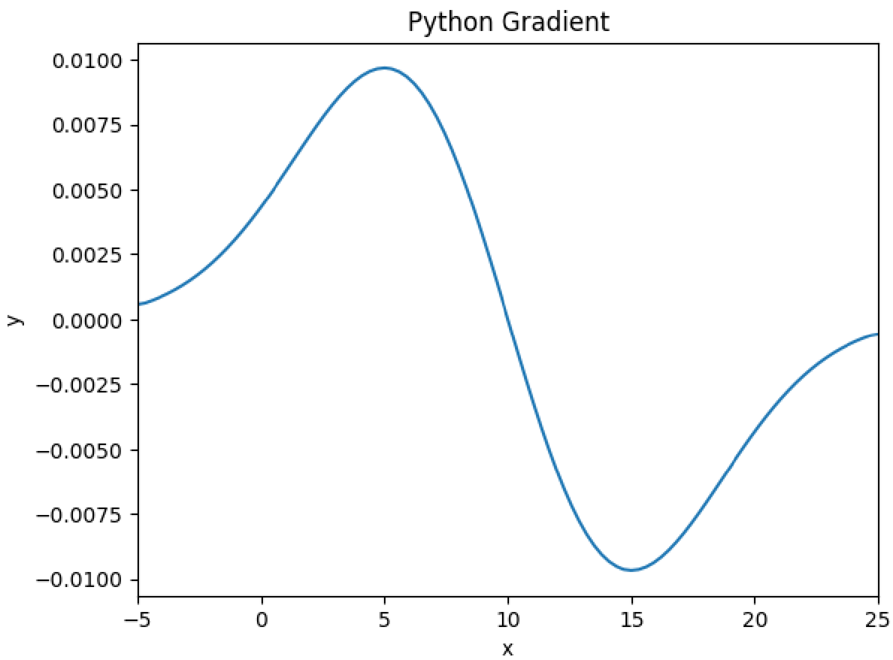


방지하기 위해서는 range를 100보다 작게 설정해야 했으므로, Python gradient를 제외한 방법들은 오른쪽 끝이 25에서 살짝 떨어져 있는 것을 볼 수 있었다. 이를 해결하기 위해서는 point의 오른쪽 끝 범위를 25보다 살짝 크게 한 후 25까지만 plot하는 방법 등을 사용할 수 있을 것이다.

- Central Method



- Python Gradient Method



각 방법을 사용하여 구한 값들을 print 함수를 이용해서 출력해보면 세부적인 값들은 달랐으나, 그래프의 개형은 동일하게 나타나는 것을 확인할 수 있었다. 공통적인 것은 for문의 index 오류를 방지하기 위해서는 range를 100보다 작게 설정해야 했으므로, Python gradient를 제외한 방법들은 오른쪽 끝이 25에서 살짝 떨어져 있는 것을 볼 수 있었다. 이를 해결하기 위해서는 point의 오른쪽 끝 범위를 25보다 살짝 크게 한 후 25까지만 plot하는 방법 등을 사용할 수 있을 것이다.

Problem 2

전체 코드

```
import numpy as np                                                 #numpy를 np라는 이름으로 호출
import scipy as sp                                                 #scipy를 sp라는 이름으로 호출
x=np.linspace(-200,200,401)                                        #linspace를 사용하여 -200,200사이에서 401개의 point x에 저장
def f(x):                                                          #x를 입력 받는 함수 f 정의
    return (1/np.sqrt(50*(np.pi)))*(np.exp((-(x-10)**2)/50))#주어진 수식에 x 대입한 값 return
g=f(x)                                                             #g에 f(x)값 저장

###Trapezoidal Method###
I1=0                                                               #변수 I1 선언
for i in range(len(x)-1):                                          #len(x)-1 만큼 반복
    I1+=((x[i+1]-x[i])*(g[i+1]+g[i]))/2                            #I1에 Trapezoidal Method 수행한 값 저장
print(f'Trapezoidal Method : {I1}')                                #Trapezoidal Method를 통해 얻은 적분 값 출력

###Simpson's Method###
I2=0                                                               #변수 I2 선언
x2=np.linspace(-200,200,801)                                       #linspace를 사용하여 -200,200사이에서 801개의 point를 x2에 저장
tempg=(1/np.sqrt(50*(np.pi)))*(np.exp((-(x2-10)**2)/50))           #tempg 변수에 주어진 수식에 x2를 대입한 값 저장
for i in range(len(x)-1):                                          #len(x)-1 만큼 반복
    I2+=((x[i+1]-x[i])/6)*(g[i]+4*tempg[2*i+1]+g[i+1])             #I2에 Simpson's Rule 수행한 값 저장
print(f"Simpson's Method : {I2}")                                  #Simpson's Method를 통해 얻은 적분 값 출력

###Euler's Method###
I3=0                                                               #변수 I3 선언
for i in range(len(x)-1):                                          #len(x)-1만큼 반복
    I3+=g[i]*(x[i+1]-x[i])                                         #I3에 Euler's Method 수행한 값 저장
print(f"Euler's Method : {I3}")                                    #Euler's Method를 통해 얻은 적분 값 출력

###Python Numerical Integration Method###
I4,I4E=sp.integrate.quad(f,-200,200)                               #변수 I4,I4E에 각각 적분 값, 오차 저장
print(f"Python Numerical Integration Method : {I4}")               #Python의 Numerical Integration Method를 통해 얻은 적분 값 출력
```

알고리즘 설명

```
import numpy as np
import scipy as sp
x=np.linspace(-200,200,401)
def f(x):
    return (1/np.sqrt(50*(np.pi)))*(np.exp((-(x-10)**2)/50))
g=f(x)
```

numpy와 scipy를 각각 np, sp라는 이름으로 지정하여 호출한다. x는 linspace함수를 활용해 -200부터 200사이에서 401개의 point를 얻었다. 그 후 x를 입력 받는 함수 f를 정의하고, 수식에 x를 대입한 값을 return하도록 한다. 그리고 f(x)를 수행한 값을 g라는 변수에 저장하였다.

(1) Use the composite (or recursive) trapezoidal rule to integrate f.

```
###Trapezoidal Method###
I1=0
for i in range(len(x)-1):
    I1+=((x[i+1]-x[i])*(g[i+1]+g[i]))/2
print(f'Trapezoidal Method : {I1}')
```

적분 값을 저장할 변수 I1을 선언한 뒤, for문을 활용하여 x의 길이에서 1을 뺀 만큼 반복하였다. Trapezoidal Rule을 사용하여 계산한 값을 I1에 계속해서 더해준다. 반복문이 종료되면 print함수를 활용하여 적분 값이 어떻게 나왔는지 출력하도록 한다.

(2) Use the composite (or recursive) Simpson's rule to integrate f.

```
###Simpson's Method###
I2=0
x2=np.linspace(-200,200,801)
tempg=(1/np.sqrt(50*(np.pi)))*(np.exp((-(x2-10)**2)/50))
for i in range(len(x)-1):
    I2+=((x[i+1]-x[i])/6)*(g[i]+4*tempg[2*i+1]+g[i+1])
print(f"Simpson's Method : {I2}")
```

적분 값을 저장할 변수 I2를 선언한 뒤, Simpson's rule 계산에서 사용할 x2를 linspace 함수를 사용하여 -200과 200사이에서 801개의 point를 얻는다. 이는 x2에 기존 x의 각 값들 사이 사이에 0.5가 더해진 값들이 존재하도록 설정한 것이다. for문을 활용하여 x의 길이에서 1을 뺀 만큼 반복하였다. Simpson's rule을 사용하여 계산한 값을 I2에 계속해서 더해준다. 반복문이 종료되면 print함수를 활용하여 적분 값이 어떻게 나왔는지 출력하도록 한다.

(3) Use the composite (or recursive) Euler's method to integrate f.

```
###Euler's Method###
I3=0
for i in range(len(x)-1):
    I3+=g[i]*(x[i+1]-x[i])
print(f"Euler's Method : {I3}")
```

적분 값을 저장할 변수 I3을 선언한 뒤, for문을 활용하여 x의 길이에서 1을 뺀 만큼 반복하였다. Euler's Method를 사용하여 계산한 값을 I3에 계속해서 더해준다. 반복문이 종료되면 print함수를 활용하여 적분 값이 어떻게 나왔는지 출력하도록 한다.

(4) Use any Python numerical integration function to integrate f.

```
###Python Numerical Integration Method###
I4,I4E=sp.integrate.quad(f,-200,200)
print(f"Python Numerical Integration Method : {I4}")
```

scipy 모듈에 내장된 integrate.quad를 이용하여 f를 적분하여, 적분 값을 I4에, 적분 오차를 I4E에 저장한다. 그 후 print 함수를 이용하여 값을 출력한다.

(5) Compare each numerical integration results.

```
Trapezoidal Method : 1.0000000000000004
Simpson's Method : 1.0
Euler's Method : 1.0000000000000002
Python Numerical Integration Method : 1.0
```

소수점 아래의 극히 작은 오차를 제외하면 모든 값들이 비슷하게 나왔다. 문제에서 주어진 함수는 정규분포 $N(10,5^2)$를 따르는 확률밀도함수(pdf)이다. 따라서 -200에서 200까지 적분을 해주면, -200과 200은 충분히 큰 숫자이므로, 1에 근사하는 값이 나올 수밖에 없다. (pdf를 $-\infty\ to + \infty$ 적분하면 1이 되기 때문이다.)