2주차 과제1 #3

2022145079 임혜린

본 과제는 Python, VS Code를 사용하였음을 밝힙니다.

문제를 푸는데 있어 공통적으로 아래의 환경을 사용하였다.

```
###3
import numpy as np
import matplotlib.pyplot as plt
import math
plt.rcParams['font.family'] = 'Malgun Gothic'
plt.rcParams['axes.unicode_minus'] = False
```

2-1

2) Numerical differentiation

주어진 함수 $f(x) = sin((4-x)(4+x)), 0 \le x \le 8$ 에 대하여 다음 문제를 푸시오. Uniform nodes를 사용하여 33개의 격자점을 사용하도록 한다.

1. Second-order one-sided difference shceme을 사용하여 경계에서의 f''를 유도하시오 (서술)

코드

```
###2-1
def f(x):
    return np.sin((4-x)*(4+x))

N=33
h=8/(N-1)
xlist=np.linspace(0,8,N)

def fffr(x):
    global h
    return (2*f(x)-5*f(x+h)+4*f(x+2*h)-f(x+3*h))/h**2

def fffl(x):
    global h
    return (2*f(x)-5*f(x-h)+4*f(x-2*h)-f(x-3*h))/h**2

print(f"""Uniform nodes 33개, Second-order one-sided difference shceme을
사용하여 유도한 결과 f''(0)={fffr(0)}, f''(8)={fffl(8)}입니다.""")
```

경계 0에서는 오른쪽의 세 점(0.25, 0.5, 0.75)을 이용한 3step 2차 후진 차분, 경계 8에서는 왼쪽 세 점 (7.25, 7.5, 7.75)을 이용한 3step 2차 전진 차분을 이용하여 f″의 근사값을 구하였다.

이에는 각 점에서의 테일러 전개를 조합하여 유도한 f″=(2f(x)-5f(x-h)+4f(x-2h)-f(x-3h))/h^2가 사용되었다.

결과

```
Uniform nodes 33개, Second-order one-sided difference shceme을
사용하여 유도한 결과 f''(0)=2.0248034695313226, f''(8)=-19.16647580126187입니다.
```

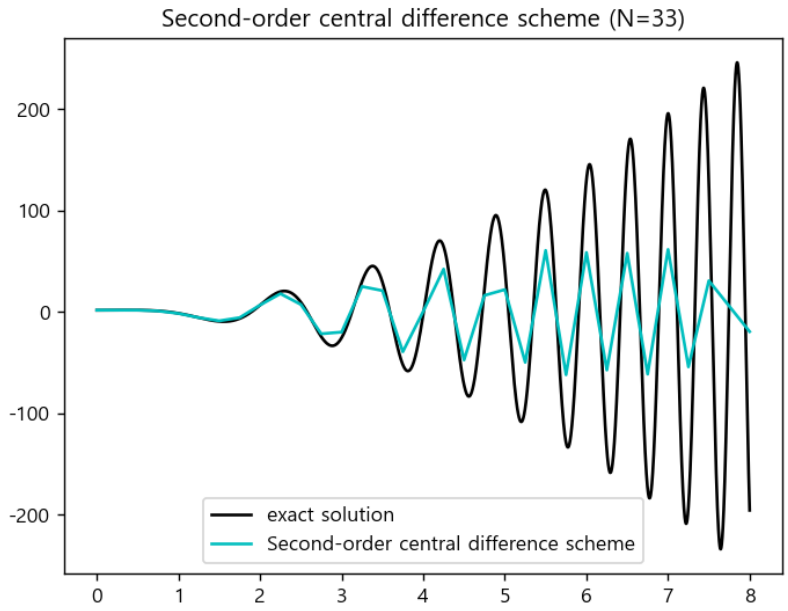
2-2

2. Second-order central difference scheme을 사용하여 exact solution 과 함께 f″를 그리시오. (정확도는 $O(\Delta x^2)$를 유지하도록 한다.)

코드

```
###2-2
N=33
h=8/(N-1)
xlist=np.linspace(0,8,N)
xor=np.linspace(0,8,3200)

def fff(x):
    global h
    return (f(x+h)-2*f(x)+f(x-h))/h**2

def exact_fff(x):
    return -2*np.cos(16-x**2)-4*x**2*(np.sin(16-x**2))

def endpoints(x):
    x[0]=fffr(0)
    x[-1]=fffl(8)

fff_xlist=[fff(x) for x in xlist]
endpoints(fff_xlist)

plt.plot(xor, exact_fff(xor), 'k', label="exact solution")
plt.plot(xlist, fff_xlist, 'c', label='Second-order central difference scheme')
plt.legend(loc='lower center')
plt.title("Second-order central difference scheme (N=33)")

plt.show()
```

결과



2-3

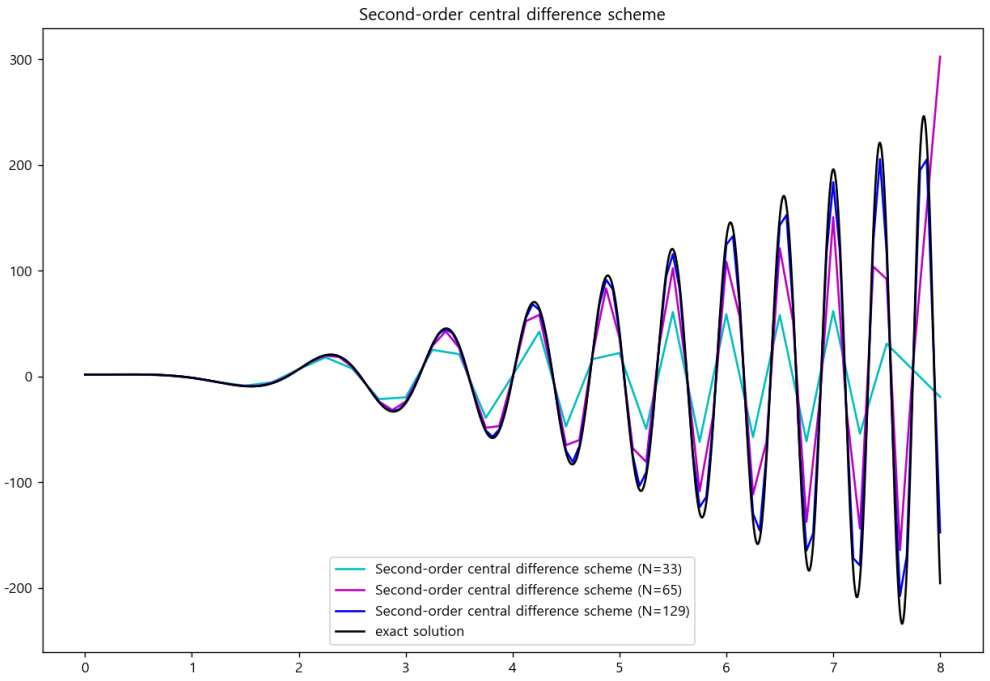
3. Second-order central difference scheme을 사용하여 격자 개수를 바꿔가며 (33,65,129) 정확도를 분석하시오. 정확도 분석은 x축을 $log(\Delta x)$, y축을 $log||e||$ 를 그리도록 한다. 본 방법에서 $||e||$ 는 L_2 $norm\ error$ 를 의미한다.

코드

```
###2-3
N33=33
h=8/(N33-1)
xlist33=np.linspace(0,8,N33)
fff_N33=[fff(x) for x in xlist33]
endpoints(fff_N33)

N65=65
h=8/(N65-1)
xlist65=np.linspace(0,8,N65)
fff_N65=[fff(x) for x in xlist65]
endpoints(fff_N65)
print(f'N=65, x=8에서의 함숫값: {fff_N65[-1]}')
print(f'x=8에서의 exact solution: {exact_fff(8)}')

N129=129
h=8/(N129-1)
xlist129=np.linspace(0,8,N129)
fff_N129=[fff(x) for x in xlist129]
endpoints(fff_N129)
```

정확도 분석에 앞서 N에 따른 세 f″ 그래프를 정의하고 한 번에 그려보았다. 각 경계에서는 중앙 차분을 사용할 수 없기 때문에 2-1처럼 후진 차분과 전진 차분을 사용하였다.

```
plt.figure(figsize=(12,8))
plt.plot(xlist33, fff_N33, 'c', label='Second-order central difference scheme (N=33)')
plt.plot(xlist65, fff_N65, 'm', label='Second-order central difference scheme (N=65)')
plt.plot(xlist129, fff_N129, 'b', label='Second-order central difference scheme (N=129)')
plt.plot(xor, exact_fff(xor), 'k', label="exact solution")
plt.legend(loc='lower center')
plt.title("Second-order central difference scheme")
plt.show()
```

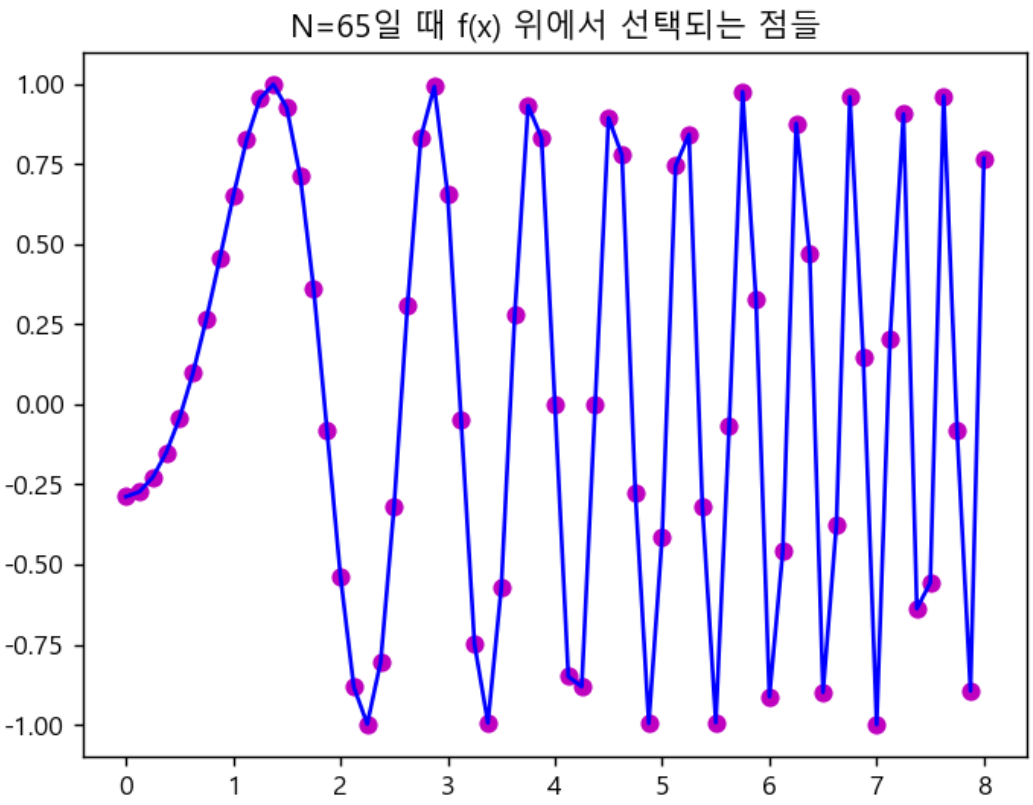



```
N=65, x=8에서의 함숫값: 302.53655332973847
```

그래프를 통해 N=65인 경우 x=8인 지점에서 함숫값이 302로 튀는 것을 확인하였다. Exact solution의 해는 -195로 매우 큰 차이를 보인다. 이 오차가 아래의 오차 검증에 큰 영향을 미쳐 오차 계산 시 이 점을 제외하고 진행하였다.

함숫값이 튀는 원인을 확인하기 위해 f(x)에 N=65인 경우 골라지는 점들을 찍어 확인해 보았다

```
y_f=[np.sin((4-x)*(4+x)) for x in xlist65]
plt.plot(xlist65, y_f, c='b', label='sin((4-x)*(4+x))')
plt.scatter(xlist65, y_f, c='m', label='sin((4-x)*(4+x))')
plt.title("N=65일 때 f(x) 위에서 선택되는 점들")
plt.show()
```

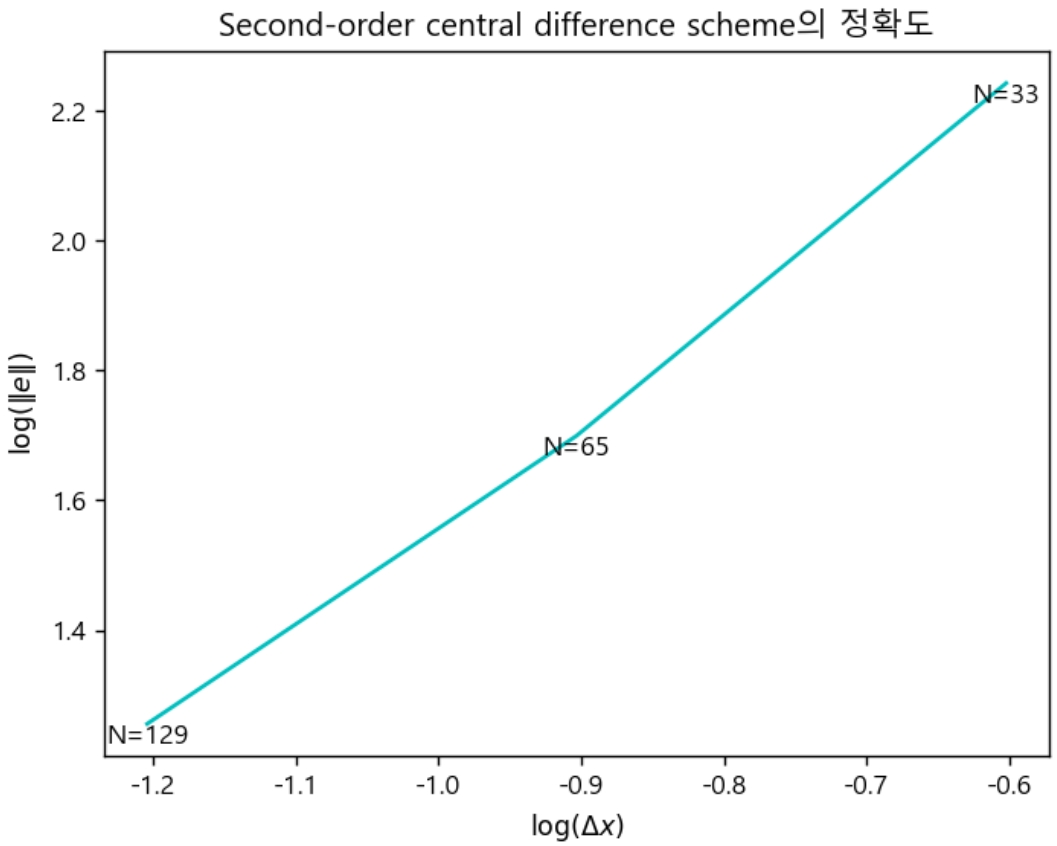


사용한 전진 차분의 식은 (2f(x)-5f(x-h)+4f(x-2h)-f(x-3h))/h^2이다. 가장 가중치가 큰 f(7.75)의 값이 극단적인 것이 주요 원인으로 보이나 직접 값을 확인해 보기 전까지는 이를 예상하기 어려울 것으로 보인다.

```
error33=[(exact_fff(xlist33[i])-fff_N33[i])**2 for i in range(33)]
# x=8일 때 값이 튀는 것을 보정하기 위하여 range(65)가 아닌 range(64)로 진행하였다.
error65=[(exact_fff(xlist65[i])-fff_N65[i])**2 for i in range(64)]
error129=[(exact_fff(xlist129[i])-fff_N129[i])**2 for i in range(129)]
error=[np.log10(math.sqrt(sum(error33)*8/(N33-1))),
       np.log10(math.sqrt(sum(error65)*8/(N65-1))),
       np.log10(math.sqrt(sum(error129)*8/(N129-1)))]
log_h=[np.log10(8/(N33-1)), np.log10(8/(N65-1)), np.log10(8/(N129-1))]



plt.plot(log_h, error, 'c', label='정확도')
plt.text(log_h[0], error[0], 'N=33', fontsize=10, ha='center', va='top')
plt.text(log_h[1], error[1], 'N=65', fontsize=10, ha='center', va='top')
plt.text(log_h[2], error[2], 'N=129', fontsize=10, ha='center', va='top')
plt.title("Second-order central difference scheme의 정확도")
plt.xlabel('$\log(\Delta x)$')
plt.ylabel('$\log(\Vert e \Vert)$')
plt.show()
```

```python
from scipy.stats import linregress

print(f'order of accuracy: {linregress(log_h, error).slope}')
# order of accuracy: 1.6388
```

결과



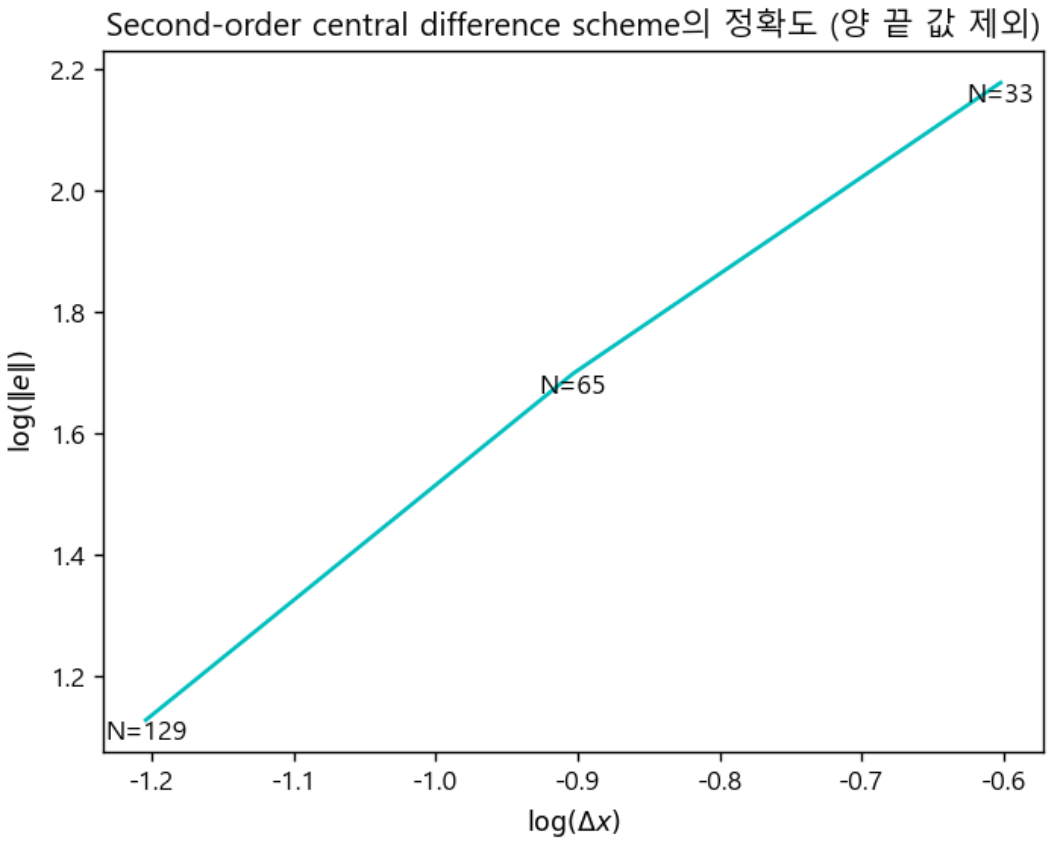
Order of accuracy: 1.6388

함수의 기울기가 유의미하지 않아 전진 차분, 후진 차분으로 만든 부분을 빼고 다시 error을 살펴보았다.

```python
error33=[(exact_fff(xlist33[i+1])-fff_N33[i+1])**2 for i in range(31)]
error65=[(exact_fff(xlist65[i+1])-fff_N65[i+1])**2 for i in range(63)]
error129=[(exact_fff(xlist129[i+1])-fff_N129[i+1])**2 for i in range(127)]
error=[np.log10(math.sqrt(sum(error33)*8/(N33-1))),
       np.log10(math.sqrt(sum(error65)*8/(N65-1))),
       np.log10(math.sqrt(sum(error129)*8/(N129-1)))]
```

```
plt.plot(log_h, error, 'c', label='정확도')
plt.text(log_h[0], error[0], 'N=33', fontsize=10, ha='center', va='top')
plt.text(log_h[1], error[1], 'N=65', fontsize=10, ha='center', va='top')
plt.text(log_h[2], error[2], 'N=129', fontsize=10, ha='center', va='top')
plt.title("Second-order central difference scheme의 정확도 (양 끝 값 제외)")
plt.xlabel('$\log(\Delta x)$')
plt.ylabel('$\log(\Vert e \Vert)$')
plt.show()

from scipy.stats import linregress

print(f'order of accuracy: {linregress(log_h, error).slope}')
# order of accuracy: 1.7441
```

결과



Order of accuracy: 1.6388

함수의 기울기가 2와 유사함을 확인하였다. 이를 통해 오차가 Δx^2에 비례하여 감소함을 알 수 있었다.