1주차 과제2 #2

2022145079 임혜린

본 과제는 Python, VS Code를 사용하였음을 밝힙니다.

1-1

1) Numerical Interpolation

주어진 함수 $f(x)=\frac{1}{1+16x^2}, [-1,1]$ 에 대하여 다음 문제를 풀어 제출하시오.

1. h=0.2로 동일하게 간격이 나누어진 uniform nodes를 사용하여, 본 함수의 Lagrange interpolating polynomial p(x) 를 찾고 그리시오.

Lagrange interpolation은 n+1의 데이터 점을 모두 통과하는 최대 n차 다항식을 찾는 보간법이다. 다항식을 P(x)이라고 했을 때 P(x)는 모든 데이터 점을 만족하며 다음과 같이 나타낼 수 있다.

$$P(x)=\sum_{k=0}^{n} y_k \cdot L_k(x)$$

이때 Lk(x)는 다음과 같이 정의된다.

$$L_k(x)=\prod_{\substack{j=0 \\ j\neq k}}^{n} \frac{(x-x_j)}{(x_k-x_j)}$$

정의와 같이 Lk(x)는 k 번째 점에서 1, 다른 점에서 0이 된다. 따라서 Lk(x)의 계수인 yk이 k번째 점에서의 y값이 된다.

```python
def f(x):
    return 1/(1+16*x**2)

h=0.2
N=int((1+1)/h)
xlist=[-1+h*i for i in range(N+1)]
xlist=[round(i,2) for i in xlist]
ylist=[f(x) for x in xlist]

def p(x):
    pp=0
    for n in range(len(xlist)):
        ppp=1
        for i in range(len(xlist)):
            if i!=n:
                ppp*=(x-xlist[i])/(xlist[n]-xlist[i])
        pp+=ppp*ylist[n]
    return pp
```

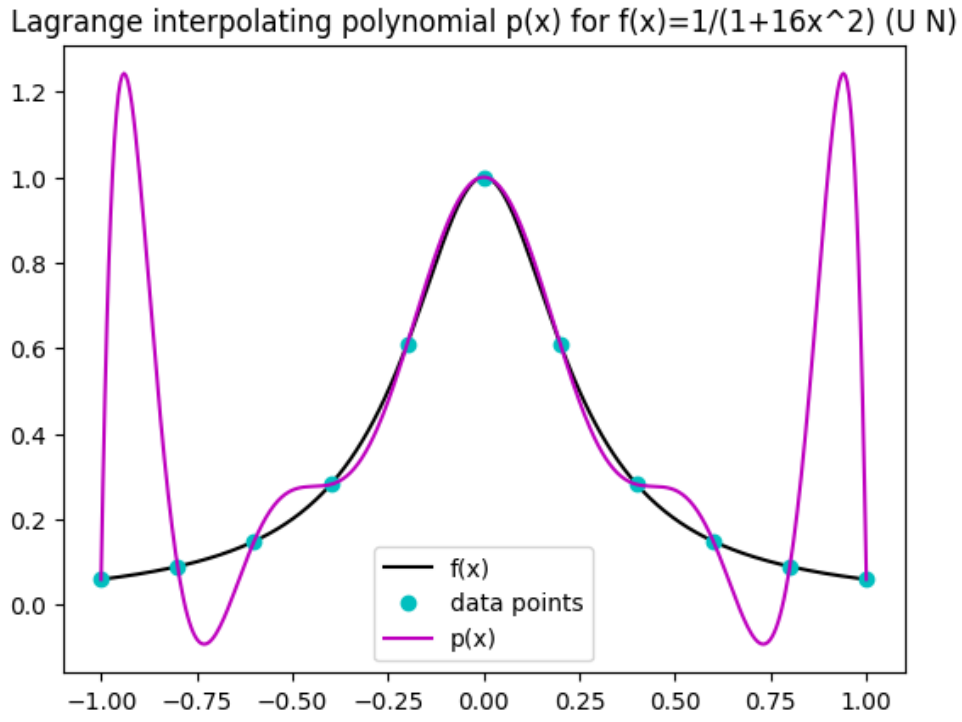
이중 for문을 사용한 함수 p(x)를 이용하여 이를 나타내었으며, 코드에서의 for i으로 각 Lk(x)를 구성한 후, 이것을 for n으로 이 Lk(x)를 모두 더해 p(x)를 완성했다. xlist는 간격 h로 나눈 uniform nodes인 x를 담은 리스트이며, p(x)는 xlist의 모든 x에서 f(x)와 만난다.

```
import numpy as np

y2list=[p(x) for x in xlist]

xor=np.linspace(-1,1,400)
yor=[f(x) for x in xor]
yor2=[p(x) for x in xor]
```

세밀한 node로 (리스트 xor) 찍어 그린 f(x) 그래프와 p(x)를 함께 그렸다. 또한 xlist의 모든 점을 원형으로 찍어 기준이 된 x 값이 어디에 있는지 쉽게 알 수 있게 하였다.

f(x)와 p(x) 그래프 코드와 결과

```
import matplotlib.pyplot as plt

plt.plot(xor, yor, 'k', label='f(x)')
plt.plot(xlist, ylist, 'c', label='data points', linestyle='None', marker='o')
plt.plot(xor, yor2, 'm', label='p(x)')
plt.legend(loc='lower center')
plt.title("Lagrange interpolating polynomial p(x) for f(x)=1/(1+16x^2) (U N)")
plt.show()
```



양 끝에서 error가 크게 나타난 것을 확인할 수 있다. (오버슈팅)

1-2

2. 동일한 과정을 Chebyshev nodes $x_i = cos\left(\frac{2i+1}{2n+2}\pi\right), i = 0, 1, \dots, 10$ 를 사용하여 진행하시오.

Chebyshev nodes는 오버슈팅과 오차를 최소화하기 위하여 사용되는 노드이며, [a, b]에서 n+1개의 Chebyshev nodes를 생성할 경우 노드 xk는 다음과 같이 정의된다.

$$x_k = \frac{a+b}{2} + \frac{b-a}{2}\cos\left(\frac{2k+1}{2(n+1)}\pi\right)$$

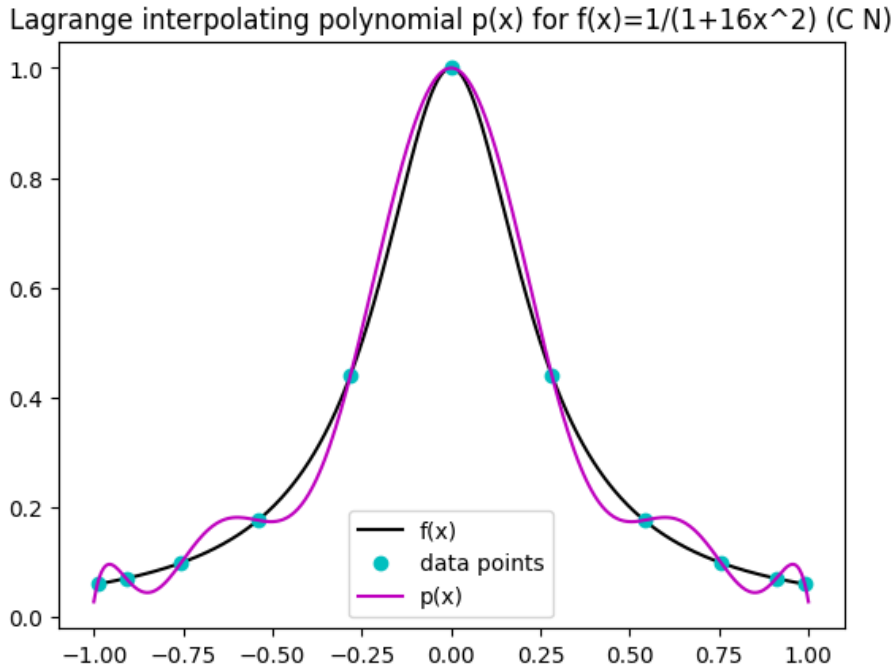
과제에서는 구간이 [-1, 1]이므로 문제와 같은 형태로 나타난다.

```
def f(x):
    return 1/(1+16*x**2)

import numpy as np

N=10
xlist=[np.cos((2*i+1)*np.pi/(2*N+2)) for i in range(N+1)]
xlist.sort()
ylist=[f(x) for x in xlist]

def p(x):
    ll=0
    for n in range(len(xlist)):
        lll=1
        for i in range(len(xlist)):
            if i!=n:
                lll*=(x-xlist[i])/(xlist[n]-xlist[i])
        ll+=lll*ylist[n]
    return ll


y2list=[p(x) for x in xlist]

xor=np.linspace(-1,1,400)
yor=[f(x) for x in xor]
yor2=[p(x) for x in xor]
```

xlist의 정의를 제외한 부분은 1-1과 같다. xi의 값이 i가 증가함에 따라 감소하는 형태이므로 (cos(0)에서 cos(π)까지) 순서를 오름차순으로 맞추어 주기 위해 sort를 진행하였다.

f(x)와 p(x) 그래프 코드와 결과

```
plt.plot(xor, yor, 'k', label='f(x)')
plt.plot(xlist, ylist, 'c', label='data points', linestyle='None', marker='o')
plt.plot(xor, yor2, 'm', label='p(x)')
plt.legend(loc='lower center')
plt.title("Lagrange interpolating polynomial p(x) for f(x)=1/(1+16x^2) (C N)")
plt.show()
```



노드가 양 끝에 몰려 있어 uniform nodes에서 나타났던 양 끝의 오버슈팅이 크게 해결된 것을 육안으로 확인할 수 있다.

1-3

3. Chebyshev nodes를 사용하였을 때가, 동일한 간격으로 나누어진 격자 점을 사용한 경우보다 좋은 결과를 보이는지를 서술하시오. [HINT : uniform and Chebyshev nodes를 사용하여 $\prod_{i=0}^{n}|x-x_i|$를 그려보시오.]

위와 같은 오차 계산 공식은 특정 x 값과 모든 노드와의 거리의 총 곱을 뜻한다.

Uniform nodes와 Chebyshev nodes를 각각 생성한 후 각각에 오차 계산 공식을 적용한 오차 함수를 만들어 비교하였다.

Uniform nodes의 경우

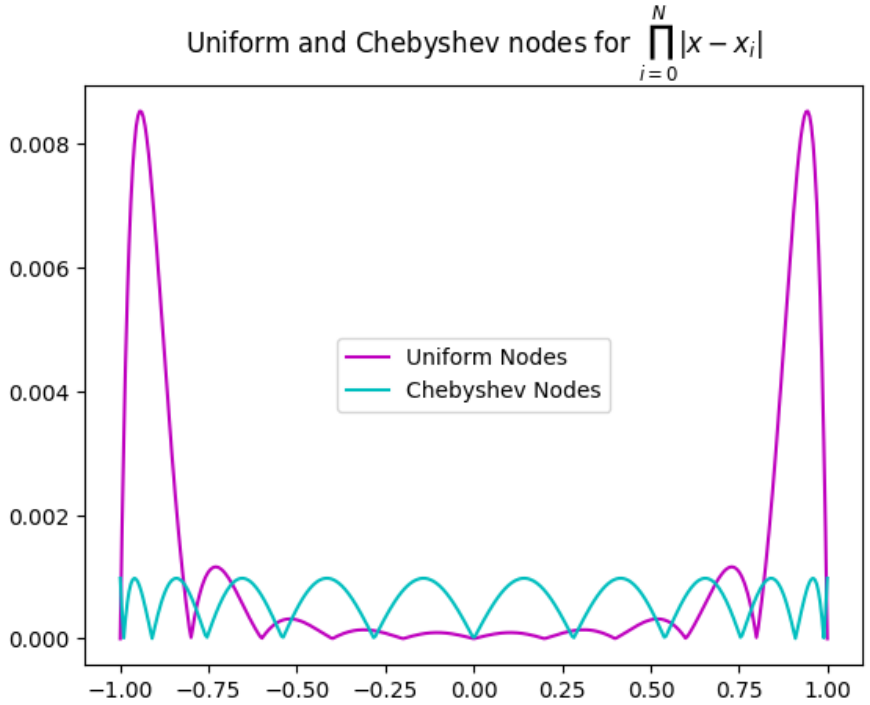
```
h=0.2
N=int((1+1)/h)
x1list=[-1+h*i for i in range(N+1)]
x1list=[round(i,2) for i in x1list]

def g1(x):
    gg1=1
    for i in range(len(x1list)):
        gg1*=abs(x-x1list[i])
    return gg1
```

Chebyshev nodes의 경우

```
N=10
x2list=[np.cos((2*i+1)*np.pi/(2*N+2)) for i in range(N+1)]

def g2(x):
    gg2=1
    for i in range(len(x2list)):
        gg2*=abs(x-x2list[i])
    return gg2

xor=np.linspace(-1,1,400)
yor=[g1(x) for x in xor]
yor2=[g2(x) for x in xor]
```

두 오차 그래프 코드와 결과

```
plt.plot(xor, yor, 'm', label='Uniform Nodes')
plt.plot(xor, yor2, 'c', label='Chebyshev Nodes')
plt.legend(loc='center')
plt.title("Uniform and Chebyshev nodes for $\prod_{i=0}^{N} |x - x_i|$")
plt.show()
```



Uniform nodes의 경우 구간 양 끝에서 오차가 매우 커졌다. 반면 구간 양 끝에서 촘촘한 Chebyshev nodes은 오차가 비교적 일정하게 유지되는 것을 확인할 수 있다. 이것이 1-2에서 언급되었던 '오버슈팅과 오차를 최소화하기 위하여 사용되는 노드'라는 Chebyshev nodes의 설명의 이유이다. Chebyshev nodes은 이러한 특성 때문에 '최적의 노드 배치'라고도 불린다.

1-4

4. Cubic spline interpolation 을 사용하여 1,2에 대해 진행하시오.

**** 양 끝 경계에 대해서는 각 점에서 두 번 미분한 값이 0 이라는 조건을 추가한다.**

Cubic spline interpolation은 두 점을 잇는 구간마다 별도의 3차 다항식을 세워 전체 데이터를 하나의 매끄러운 곡선으로 근사하는 방법이다.

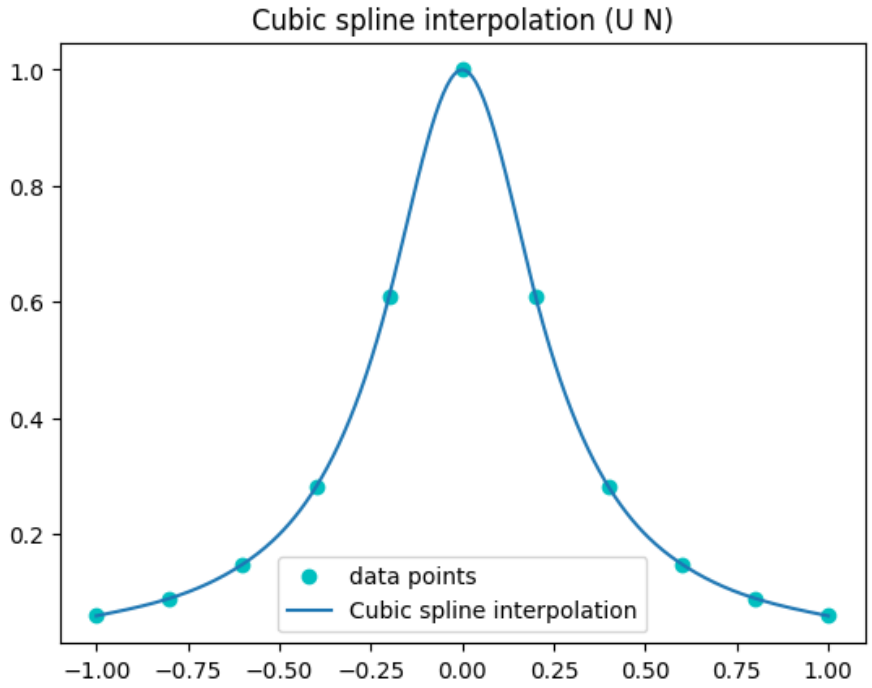
구간마다 3차 다항식을 세운 후, 함수값의 연속성과 미분 가능성, 2차 미분의 값을 이용해 3차 다항식의 계수를 채워 나간다. 문제에서는 두 번 미분한 값이 0이라는 조건을 주었으므로 bc_type='natural'을 통해 이 조건을 만족시키도록 하였다. Python의 내장함수인 CubicSpline을 이용하면 문제를 비교적 간단히 풀 수 있다. Node list를 생성하는 부분은 1-1, 1-2와 같다.

Uniform Nodes의 경우

```
from scipy.interpolate import CubicSpline

# Uniform Nodes
def f(x):
    return 1/(1+16*x**2)

h=0.2
N=int((1+1)/h)
x1list=[-1+h*i for i in range(N+1)]
x1list=[round(i,2) for i in x1list]
y1list=[f(x) for x in x1list]

xor=np.linspace(-1,1,400)

cs1=CubicSpline(x1list, y1list, bc_type='natural')
```

Uniform Node의 그래프 코드와 결과

```
cs1=CubicSpline(x1list, y1list, bc_type='natural')
plt.plot(x1list, y1list, 'c', label='data points', linestyle='None', marker='o')
plt.plot(xor, cs1(xor), label='Cubic spline interpolation')
plt.legend(loc='lower center')
plt.title("Cubic spline interpolation (U N)")
plt.show()
```

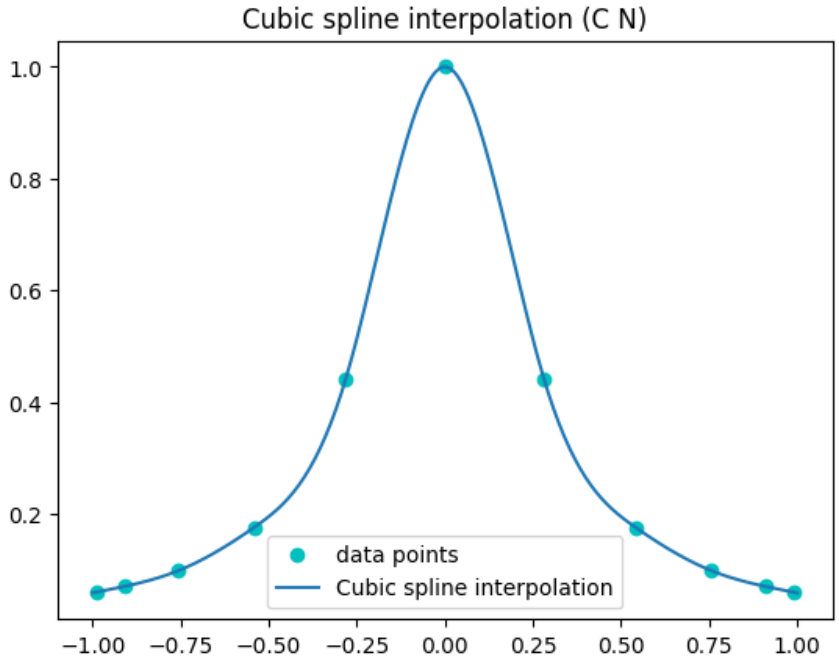


Chebyshev Nodes의 경우

```
def f(x):
    return 1/(1+16*x**2)

N=10
x2list=[np.cos((2*i+1)*np.pi/(2*N+2)) for i in range(N+1)]
x2list.sort()
y2list=[f(x) for x in x2list]

xor=np.linspace(-1,1,400)

cs2=CubicSpline(x2list, y2list, bc_type='natural')
```

Chebyshev Nodes의 그래프 코드와 결과

```
plt.plot(x2list, y2list, 'c', label='data points', linestyle='None', marker='o')
plt.plot(xor, cs2(xor), label='Cubic spline interpolation')
plt.legend(loc='lower center')
plt.title("Cubic spline interpolation (C N)")
plt.show()
```



1-1, 1-2에서와 달리, Uniform Nodes와 Chebyshev Nodes 모두에서 매우 균일하고 매끄러운 곡선을 얻었음을 확인할 수 있다.

1-5

5. Chebyshev node를 사용하여 Lagrangian interpolation과 Cubic spline method를 적용한 결과와 Exact solution 과의 비교를 통한 Error 분석을 하고, 이에 대한 본인의 의견을 서술하시오.

위 문제들에서 풀이한 대로 Chebyshev node 리스트인 xlist, 반드시 지나는 점이자 f(x)와 만나는 점들의 리스트인 ylist, [-1, 1]을 촘촘히 자른 xor 리스트를 생성한 후, 각각 함수 p(x)와 CubicSpline을 이용하여 Lagrangian interpolation과 Cubic spline을 하였다. 그리고 xor에 적용하여 각각의 점에서의 오차를 구한 후 이를 종합하여 Error값을 구하였다.

기준이 되는 Error은 Mean Squared Error, 이하 RMSE Error을 사용하였다. RMSE Error은 예측값과 실제값의 차이를 제곱한 후 이를 평균하고, 이 평균값을 루트 씌워 얻는 값으로 아래와 같다.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(e_i)^2}$$

Lagrangian interpolation의 경우

```
def f(x):
    return 1/(1+16*x**2)

N=10
xlist=[np.cos((2*i+1)*np.pi/(2*N+2)) for i in range(N+1)]
xlist.sort()
ylist=[f(x) for x in xlist]
xor=np.linspace(-1,1,400)

# Chebyshev node, Lagrangian interpolation
def p(x):
    ll=0
    for n in range(len(xlist)):
        lll=1
        for i in range(len(xlist)):
            if i!=n:
                lll*=(x-xlist[i])/(xlist[n]-xlist[i])
        ll+=lll*ylist[n]
    return ll

error_LI=[(f(x)-p(x))**2 for x in xor]
Error_LI=math.sqrt(sum(error_LI)/len(xor))
print(f"Lagrangian interpolation를 이용한 경우 RMSE Error은 {Error_LI:.4f}입니다.")
```

Cubic spline method의 경우

```
# Chebyshev node, Cubic spline method
cs=CubicSpline(xlist, ylist, bc_type='natural')

error_CS=[(f(x)-cs(x))**2 for x in xor]
Error_CS=math.sqrt(sum(error_CS)/len(xor))
print(f"Cubic spline method를 이용한 경우 RMSE Error은 {Error_CS:.4f}입니다.")
```

Lagrangian interpolation와 Cubic spline method의 RMSE Error

```
Lagrangian interpolation를 이용한 경우 RMSE Error은 0.0349입니다.
Cubic spline method를 이용한 경우 RMSE Error은 0.0139입니다.
```

Lagrangian interpolation에 비해 Cubic spline method의 Error가 약 2.5배 작게 나온 것을 확인하였다. 이는 1-2와 1-4에서 결과로 얻은 그래프를 육안으로만 비교해 보아도 예상할 수 있는 결과이다. 전체에 대해 하나의 고차 다항식을 얻는 Lagrangian interpolation과 달리 Cubic spline method는 두 노드 사이의 구간에 알맞은 3차 다항식을 모두 각각 구한 후 이를 연결해 완성한다. 따라서 각 구간에 최적화된 근사 함수를 얻을 수 있어 Error값을 획기적으로 줄일 수 있는 것으로 보인다.