

# Python 시각화 라이브러리를 이용한 미적분 시각화

김상오 꽈민승 권이준 김준서

## 요약

Python 프로그래밍 언어에는 matplotlib, plotly와 같이 쉽고 강력한 데이터 시각화 기능을 가진 라이브러리가 있다. 본 연구에서는 위의 라이브러리를 이용하여 고등학생들이 수학을 공부하며 가장 많이 접하지만 동시에 가장 어려워하기도 하는 미적분 문제들을 다룬다. 미적분 문제에서 자주 다루어지는 소재인 그래프 개형 파악과정을 코드를 통해 시각화함으로써 미적분에 대한 이해도를 높힐 수 있고 미적분 문제 뿐 아니라 이차곡선, 확률밀도함수의 개형에 대해서도 시각화해보며 고등학교 수학의 중요 개념에 대한 이해도를 증진시킬 수 있었다.

키워드: 시각화, 미적분, matplotlib, python

## 목차

- I. 서론
- II. 이론적 배경
  - 1. matplotlib
  - 2. NumPy
  - 3. Sympy
  - 4. Plotly
- III. 탐구 방법 및 절차
- IV. 연구 결과 및 분석
  - 1. 시각화를 통한 역도함수 분석
  - 2. 테일러 급수의 시각화
  - 3. 시각화를 통한 점화관계 분석
  - 4. Plotly를 이용한 초월함수의 그래프 개형 분석
  - 5. 시각화를 통한 이차곡선 분석
  - 6. 시각화를 통한 수치적분/몬테카를로 적분 분석
- V. 결론 및 논의
  - 1. 탐구 결론
  - 1. 개선점

## I. 서론

고등학교 수학 교육과정에서 배우는 여러 주제들 중 미적분은 학생들이 가장 많이 접하면서도 가장 어려워하는 단원 중 하나로 인식된다. 미적분 문제를 해결하기 위해서는 식을 통한 접근 능력 뿐 아니라 그래프의 개형을 유추하고 그래프의 특징을 통해 문제 해결의 단서를 얻는 능력도 동시에 요구되기 때문이다. 이렇듯 고등학교 시절 접하는 미적분 문제를 원활히 해결하기 위해서는 그 그래프에 대한 감각이 꼭 필요하다. 우리는 이를 가장 효과적으로 키울 수 있는 방법은 다양한 문제에서 다양한 그래프를 경험해보는 것이라고 생각하였고, 파이썬의 시각화 라이브러리 및 symbolic computation 라이브러리를 활용하여 미적분 문제 풀이 및 이해에 도움을 줄 수 있는 시

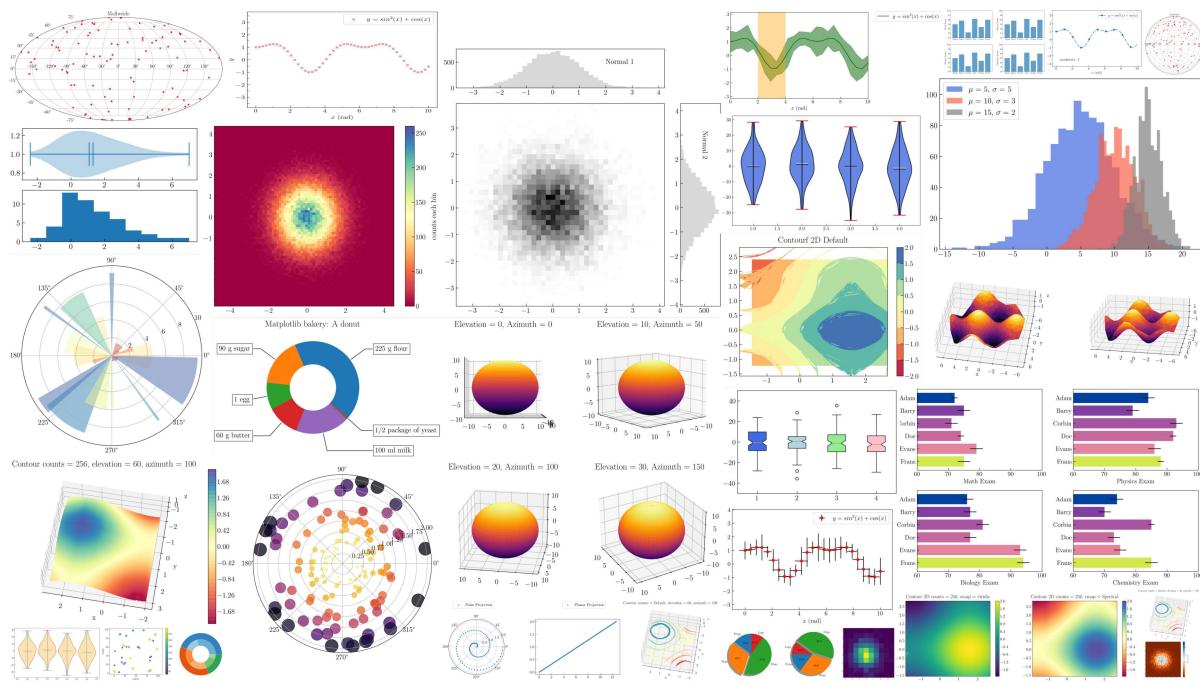
각화 작업을 진행하였다. 또한 표준편차에 따른 확률밀도함수의 개형변화, 수치적분 등 여러 수학 개념들에 대한 시각화로 이해를 쉽게 할 수 있도록 하였다.

## II. 이론적 배경

- 2장에서 어떠한 것을 다루는지에 대한 간단한 소개와 요약 넣기

### 1. matplotlib

matplotlib는 그래프나 2차원 데이터 시각화를 생성하는 유명한 파이썬 라이브러리이다. 존 D. 헌터가 만들었고 지금은 많은 개발 팀이 유지하고 있다. 출판물에 필요한 그래프를 만드는데 맞추어 설계되었다. 현재 파이썬에서 사용할 수 있는 다양한 시각화 라이브러리들 중 matplotlib는 생태계 내 다른 라이브러리들과 가장 잘 연동되어 있다고 할 수 있다. 따라서 가장 널리 사용되고 기본 시각화 도구로는 가장 안전한 선택이 될 수 있어 본 연구에서도 활용하게 되었다.

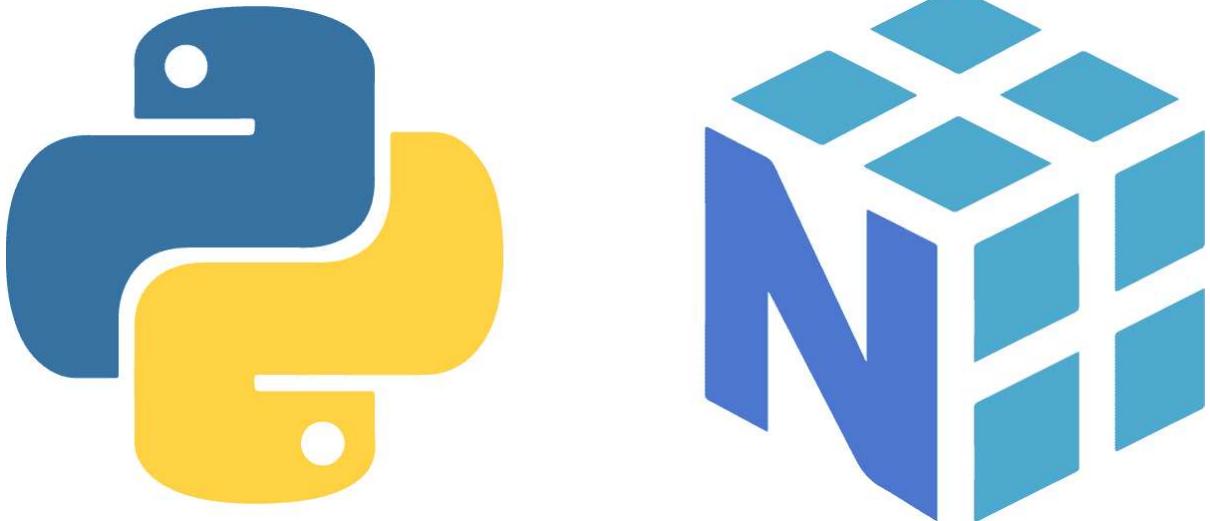


### 2. NumPy

NumPy는 Numerical Python의 줄임말로, 파이썬 산술 계산의 주춧돌과 같은 라이브러리이다. 자료구조, 알고리즘 산술 데이터를 다루는 대부분의 과학 계산 애플리케이션에서 필요한 라이브러리를 제공한다. Numpy가 제공하는 기능은 다음과 같다.

- 빠르고 효율적인 다차원 배열 객체 ndarray
- 배열 원소를 다루거나 배열 간의 수학 계산을 수행하는 함수
- 디스크로부터 배열 기반의 데이터를 읽거나 쓸 수 있는 도구
- 선형대수 계산, 푸리에 변환, 난수 생성기
- 파이썬 확장과 C/C++코드에서 NumPy의 자료구조에 접근하고 계산 기능을 사용할 수 있도록 해주는 C API

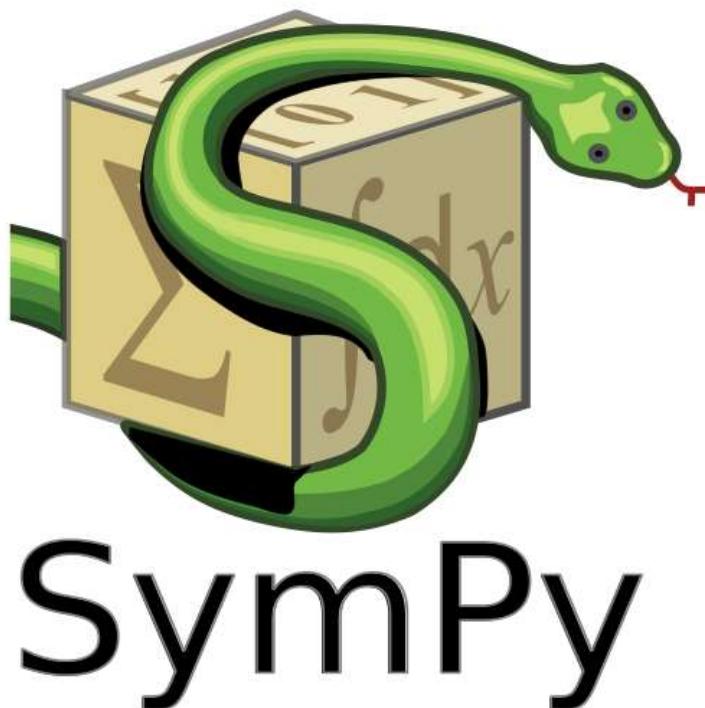
본 연구에서 NumPy는 matplotlib와 연동되어 변수의 구간 설정 등에 사용되었다.



### 3. Sympy

Sympy는 연산을 위한 오픈 소스 Python 라이브러리이다. Sympy는 독립 실행형 응용 프로그램과 다른 응용 프로그램의 라이브러리 또는 Sympy Live 또는 Sympy Gamma로 컴퓨터 대수 기능을 제공한다. Sympy는 기본적인 기호 산술부터 미적분, 대수, 이산 수학, 양자 물리학까지 다양한 특징을 포함하고 있다. 이러한 연산 결과를 LaTeX코드로 포맷할 수 있다. Sympy는 전반적으로 Python으로 작성이 잘되기 때문에 접근하기 쉬운 편이다.

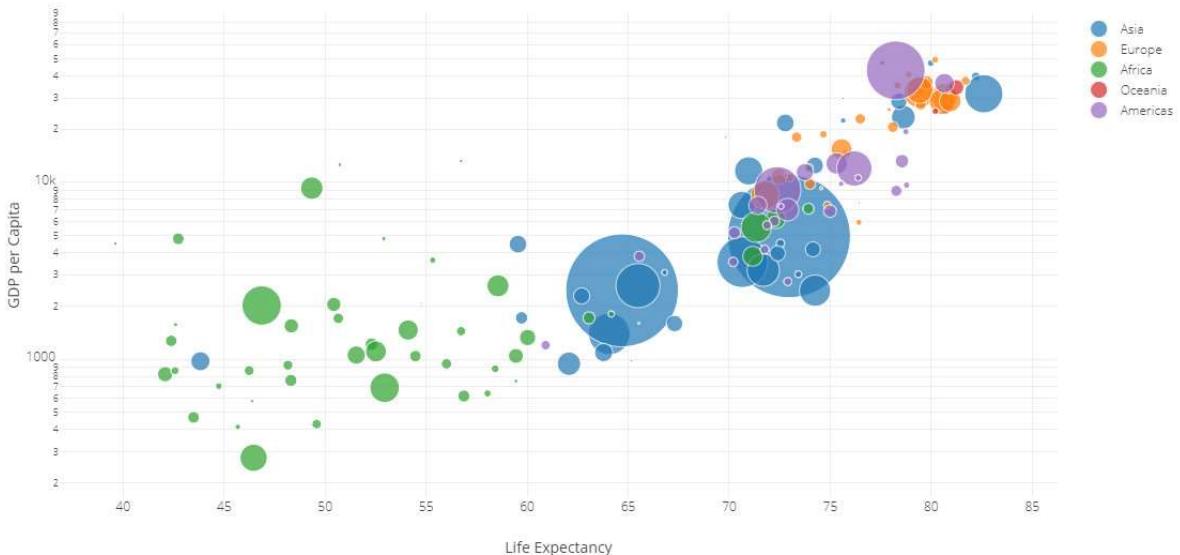
본 연구에서는 미적분 문제에서 사용된 함수를 코드를 통해 미분 또는 적분하는 과정에서 사용되었다.



### 4. Plotly

Plotly는 온라인 데이터 분석 및 시각화 도구를 개발하는 케ベ주 몬트리올에 본사를 둔 기술 컴퓨팅 회사이다. Plotly는 파이썬, R, MATLAB, 펄, 줄리아, 아두이노 및 REST를 위한 과학적 그래프 라이브러리를 비롯하여 개인 및 협업을 위한 온라인 그래프, 분석 및 통계 도구를 제공한다.

본 연구에서는 기본적인 시각화는 matplotlib를 기반으로 하되, 동적 그래픽스의 사용이 필요한 경우에 plotly 라이브러리를 활용하였다.



### III. 탐구 방법 및 절차

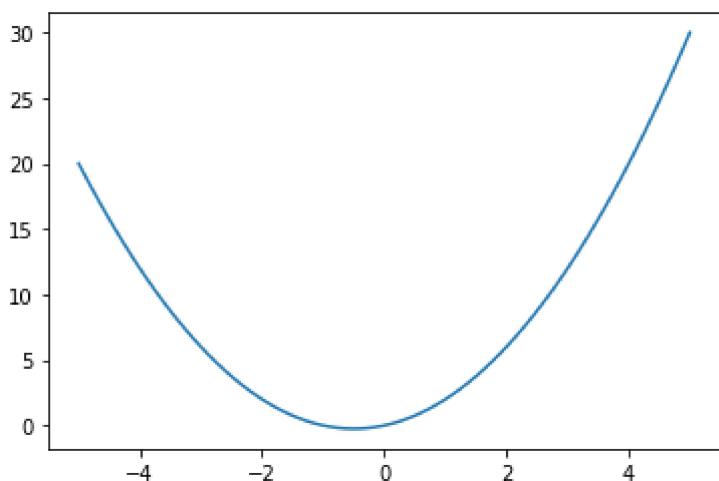
- 탐구 방법 및 절차에서 어떠한 내용들을 다루는지 간단하게 앞으로 나올 내용 소개 넣기 예시  
조금 더 추가하기

#### 1. matplotlib, NumPy

```
In [1]: import matplotlib.pyplot as plt
import numpy as np

X = np.linspace(-5, 5, 1000)
Y = X ** 2 + X

plt.plot(X, Y)
plt.show()
```



## 2. SymPy

```
In [2]: import sympy as sp
x = sp.Symbol('x')
expr = x ** 2
sp.integrate(expr)
```

Out[2]:  $\frac{x^3}{3}$

```
In [6]: import sympy as sp
x = sp.Symbol('x')
expr = sp.exp(x) * (x + 1)
sp.diff(expr)
```

Out[6]:  $(x + 1)e^x + e^x$

```
In [10]: import sympy as sp
x = sp.Symbol('x')
expr = (x - 1) * sp.exp(x ** 2)
sp.integrate(expr, (x, -1, 1))
```

Out[10]:  $-\sqrt{\pi} \operatorname{erfi}(1)$

## IV. 연구 결과 및 분석

### 1. 시각화를 통한 역도함수 분석

#### 1)

함수  $f(x) = \sin \pi x$ 와 이차함수  $g(x) = x(x+1)$ 에 대하여 실수 전체의 집합에서 정의된 함수  $h(x)$ 를

$$h(x) = \int_{g(x)}^{g(x+1)} f(t) dt$$

라 할 때, 닫힌 구간  $[-1, 1]$ 에서 방정식  $h(x) = 0$ 의 서로 다른 실근의 개수는?

```
In [5]: import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(15, 10))
pi = 3.14159265
X = np.linspace(-3, 6, 1024)

Y1 = np.sin(pi*X)
Y2 = X*(X+1)
Y3 = (X+2)*(X+1)
```

```

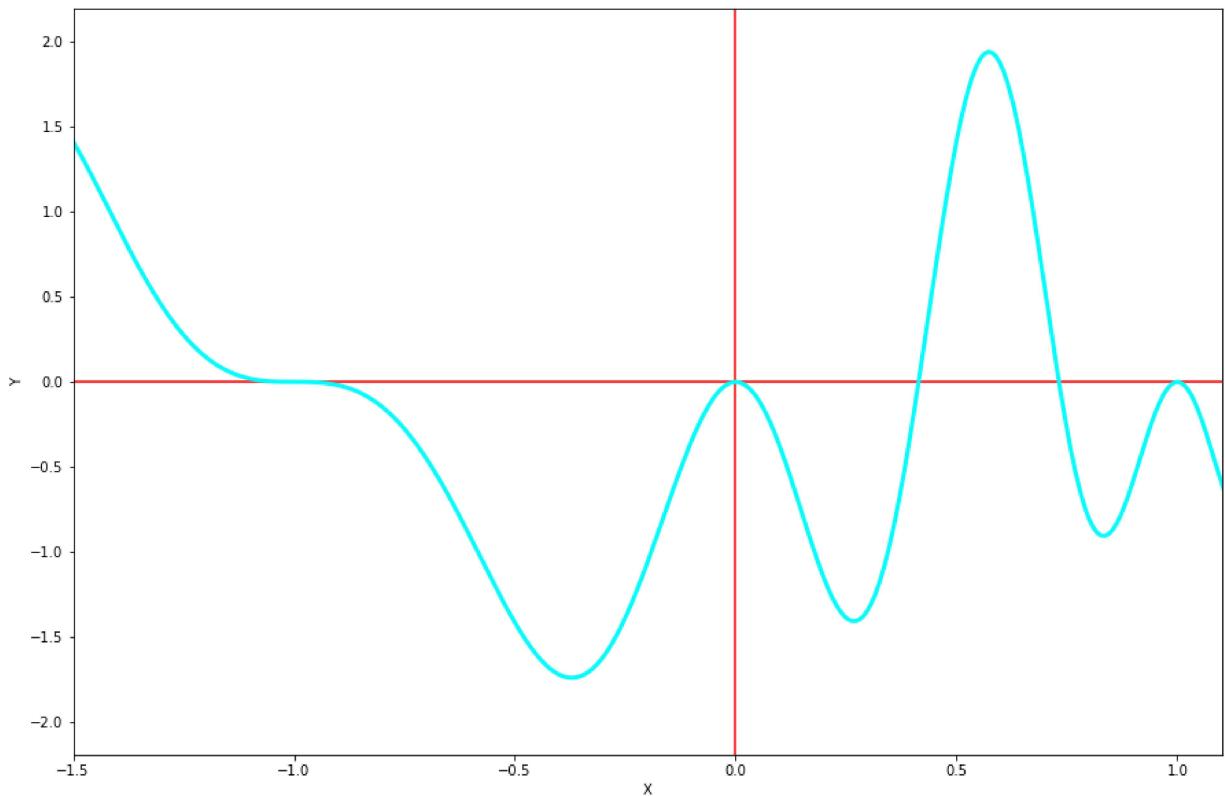
plt.xlim([-1.5, 1.1])
plt.xlabel('X')
plt.ylabel('Y')

plt.axvline(x=0, color='r')
plt.axhline(y=0, color='r')

Y4 = (np.cos(pi*Y3) - np.cos(pi*Y2))
plt.plot(X, Y4, c='cyan', lw = 3.)

plt.show()

```



## 시각화 결과 분석

이 그래프를 직접 그릴려면 현재 교육과정 밖에 있는 삼각함수의 곱을 차를 곱으로 고치는 공식이 필요하다. 그것 외에는 극대점과 극소점을 찾는 것이 쉽지 않으며 일반각을 이용해 찾는다는 아이디어가 있지 않으면 해맬 수 있고 일반각을 이용하더라도 경우를 나누어서 찾아야 하기 때문에 빠뜨리기 쉽다. 이 문제는 단순하게 그래프를 그리는 문제에서 정적분으로 정의된 함수를 그린다는 것에서 의의가 있다.

## 2)

다음 함수의 그래프를 그리고, y가 양수일때의 넓이가 수렴하는가 발산하는가?

$$y = \frac{\ln x}{x^2}$$

문제 선정의 이유 추가하기

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
from sympy import *
# sympy로 방정식 풀어내는 코드 추가하기
```

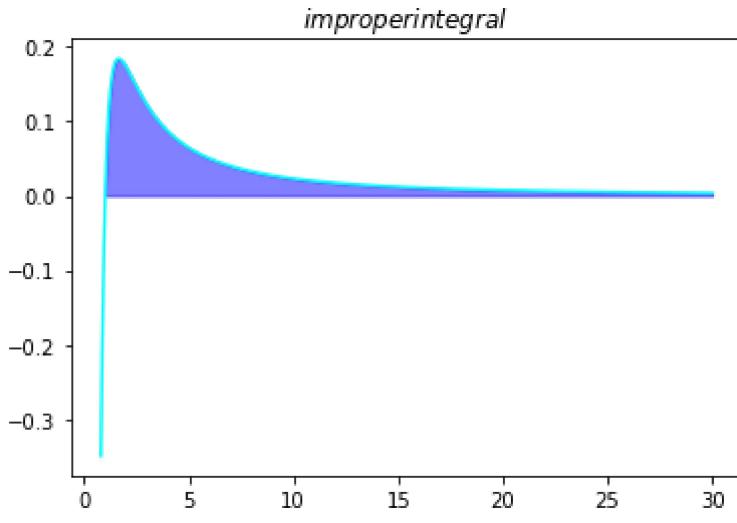
```

x=np.linspace(0.8,30,1500)
Y1= np.log(x)/(x**2)
plt.title('Improper integral')
plt.plot(x, Y1, c='cyan')

plt.fill_between(x, Y1, where = (x>1), color='blue', alpha=0.5)
plt.show()

import sympy as sp
x = sp.symbols('x')
Y = sp.log(x)/(x**2)
print(integrate(Y, (x, 1, oo)))

```



1

## 시각화 결과 분석

$y = \ln x / x^2$ 에 대한 함수는 위에서의  $y = \ln x / x$ 에서 분모의 차수가 증가한 꼴이다. 그래프 개형을 보았을 때는  $y = \ln x / x$ 와 거의 유사한 그래프를 보이기 때문에 적분한 값도 같다(무한대로 발산한다)라고 생각할 수 있을 것이다. 그러나 sympy 함수를 이용해서 적분을 하게 되면 넓이의 값이 1로 수렴하게 된다.  $y = \ln x / x$ ,  $y = \ln x / x^2$ 의 함수의 개형은 비슷하지만 수렴의 여부는 다름을 알 수 있다. 그래프만으로 판단하기 어려운 함수 그래프에서의 양의 넓이의 면적을 sympy를 이용해서 구할 수 있다는 점에서 의의가 있을 것이다.

## 2. 테일러 급수의 시각화

$n=1, 2, 3$  정도만 시각화하고  $x=0$ 에서의 개형에 주목하여 설명하기

- 테일러 급수란?  
→ 어떤 점에서 무한 번 미분 가능한 함수를 그 점에서 미분계수 값으로 계산할 수 있는 무한급수로 표현된 함수로 나타내는 것이다.
- 테일러급수는 여러 함수를 다루기 쉬운 다항함수로 근사해주어 매우 유용하다

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x-a)^n$$

→ 테일러 급수는 원래함수를 다항식으로 근사적으로 표현해주는 것인데 원래함수와 근사한 다항식이 어느 정도로 차이가 있는지 궁금하였습니다. 이를 시각화하여 그래프로 나타내면 쉽게 차이를 실감할 수 있을 것 같아서 다항함수와  $\sin$ 함수를 근사적으로 비교해보았습니다.

## 2.1 $y = \sin(x)$

```
In [16]: # n을 입력받고 n까지泰일러급수 전개 된식을 출력해주는 코드
from sympy import *
from math import factorial

n=int(input())

X = symbols('X')

def f(n):
    expr = (X ** (2 * n + 1)) * (-1) ** n / factorial(2 * n + 1)
    return expr

def g(n):
    s=0
    for i in range(n):
        s+=f(i)
    return s

print(g(n))
```

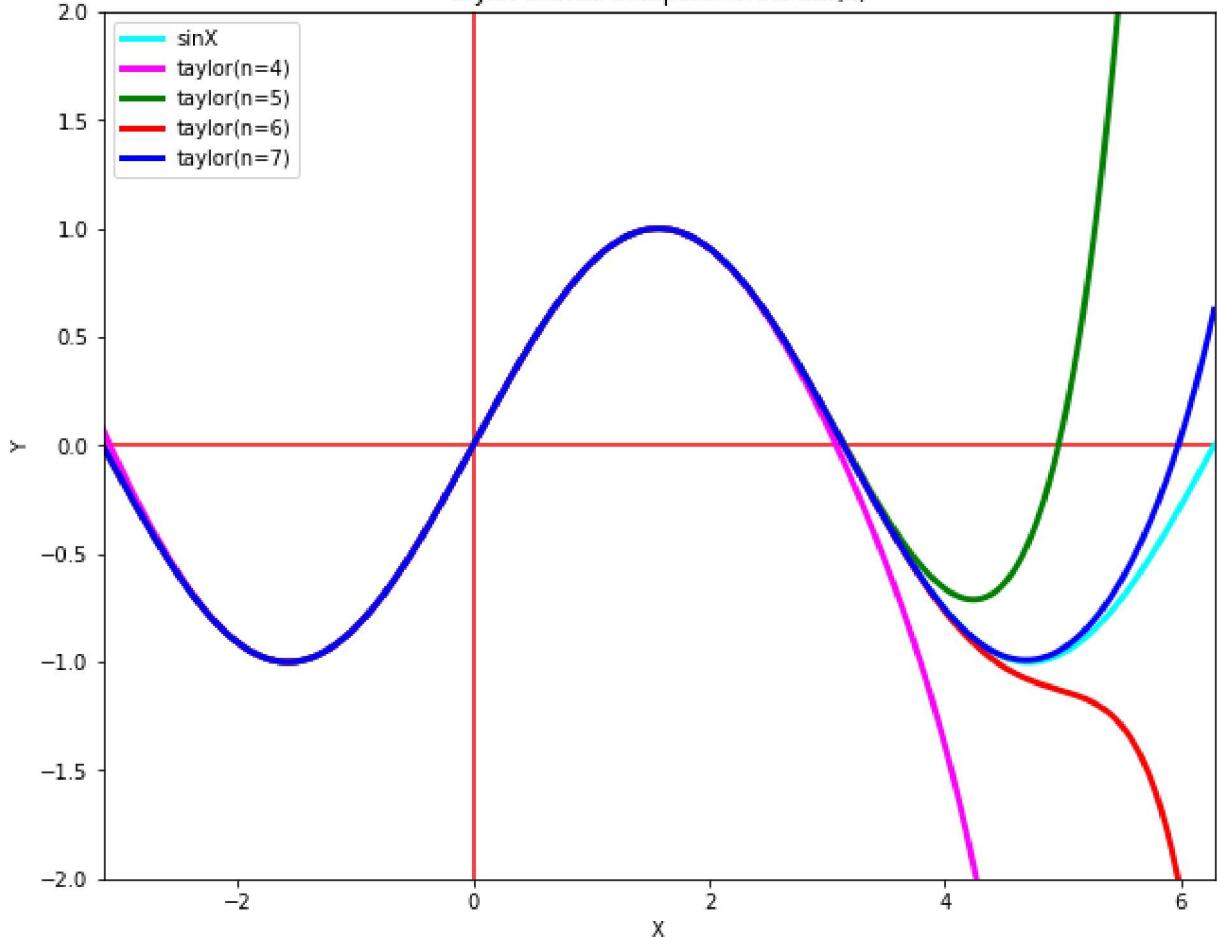
```
4
-X**7/5040 + X**5/120 - X**3/6 + X
```

```
In [9]: import numpy as np
import matplotlib.pyplot as plt
from numpy import pi

X = np.linspace(-6, 2*pi, 1024)
Y1 = np.sin(X)

plt.figure(figsize=(10,8))

plt.xlim([-1*pi, 2*pi])
plt.ylim([-2, 2])
plt.title('Taylor Series Comparison for sin(x)')
plt.xlabel('X')
plt.ylabel('Y')
plt.axvline(x=0,color='r')
plt.axhline(y=0,color='r')
plt.plot(X, Y1, c = 'cyan', lw = 3., label = 'sinX')
plt.plot(X, -X**7/5040 + X**5/120 - X**3/6 + X, c = 'magenta', lw = 3., label = 'tay')
plt.plot(X, X**9/362880 - X**7/5040 + X**5/120 - X**3/6 + X, c = 'g', lw = 3., labe
plt.plot(X, -X**11/39916800 + X**9/362880 - X**7/5040 + X**5/120 - X**3/6 + X, c =
plt.plot(X,X**13/6227020800 - X**11/39916800 + X**9/362880 - X**7/5040 + X**5/120 -
plt.legend()
plt.show()
```

Taylor Series Comparison for  $\sin(x)$ 

$$2.2 \quad y = \ln(x + 1)$$

```
In [3]: from sympy import *
from math import factorial

n=int(input())

X = symbols('X')

def f(n):
    if(n==0): return 0
    else: return ( X**n*(-1)**(n+1) )/factorial(n)

def g(n):
    s=0
    for i in range(n):
        s+=f(i)
    return s

print(g(n))
print(g(4))
print(g(5))
print(g(6))
print(g(15))
```

5  
 $-X^{15}/362880 + X^{14}/479001600 + X^{13}/6227020800 - X^{12}/479001600 + X^{11}/39916800 - X^{10}/36288$   
 $-X^9/36288 + X^8/47900160 - X^7/62270208 + X^6/4790016 - X^5/36288 + X^4/4790016 - X^3/62270208 + X^2/4790016 + X/36288$   
 $X/36288 - 1$

$$00 + X^{*}9/362880 - X^{*}8/40320 + X^{*}7/5040 - X^{*}6/720 + X^{*}5/120 - X^{*}4/24 + X^{*}3/6 - X^{*}2/2 + X$$

```
In [11]: import numpy as np
import matplotlib.pyplot as plt
from numpy import pi

plt.figure(figsize=(10,8))

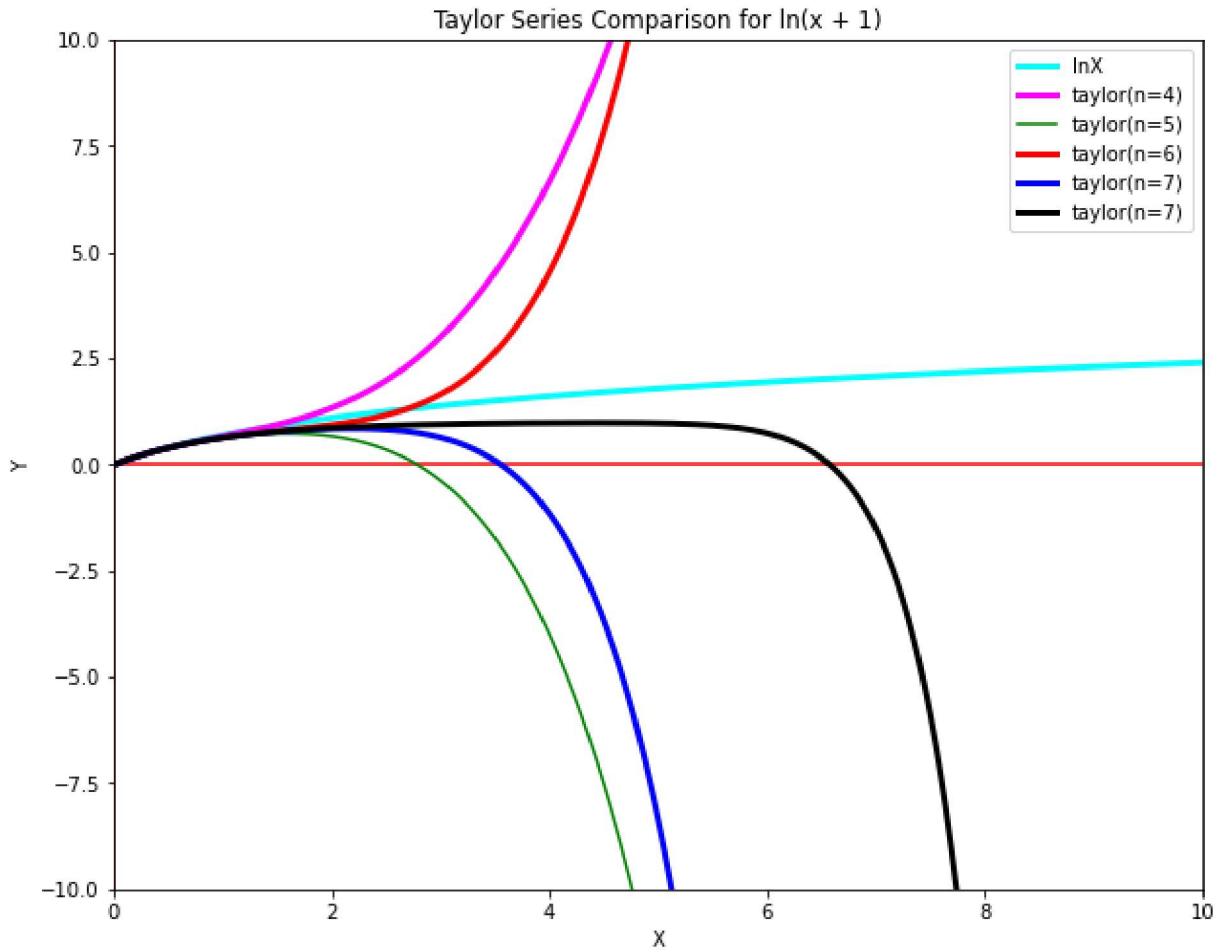
X = np.linspace(0, 10, 1024)
Y1 = np.log(X+1)

plt.title('Taylor Series Comparison for ln(x + 1)')
plt.xlim([0, 10])
plt.ylim([-10, 10])
plt.xlabel('X')
plt.ylabel('Y')
plt.axvline(x=0,color='r')
plt.axhline(y=0,color='r')

plt.plot(X, Y1, c = 'cyan', lw = 3., label = 'lnX')

plt.plot(X, X**3/6 - X**2/2 + X , c = 'magenta', lw = 3, label = 'taylor(n=4)')

plt.plot(X, -X**4/24 + X**3/6 - X**2/2 + X , c='g', label = 'taylor(n=5)')
plt.plot(X, X**5/120 - X**4/24 + X**3/6 - X**2/2 + X , c = 'r', lw = 3, label = 'ta
plt.plot(X,-X**6/720 + X**5/120 - X**4/24 + X**3/6 - X**2/2 + X , c = 'b', lw = 3,
plt.plot(X,-X**14/87178291200 + X**13/6227020800 - X**12/479001600 + X**11/39916800
plt.legend()
plt.show()
```



## 시각화 결과 분석

$n$ 값이 커질수록 원래함수와 개형이 비슷해지고,  $n$ 이 증가할 때마다  $\sin(x)$ 를 기준으로 증감을 반복하며 채찍 모양처럼 움직이는 것을 확인할 수 있었습니다.  $x$ 가 커지면서 일정 구간부터 오차가 눈에 띄게 증가하는 것을 확인할 수 있었습니다. 이와 같이 시각화 결과를 통해 근사적인 다항식과 원래 함수간의 오차의 정도에 대해서 보다 직관적으로 이해할 수 있게 되었습니다.

## 3. 시각화를 통한 점화관계 분석

문제 선정의 이유 추가하기

## 2011학년도 수능 수학 가형 25번

- 자연수  $m$ 에 대하여 크기가 같은 정육면체 모양의 블록이 1열에 1개, 2열에 2개, 3열에 3개, ...,  $m$ 열에  $m$ 개 쌓여 있다. 블록의 개수가 짹수인 열이 남아있지 않을 때까지 다음 시행을 반복한다.

블록의 개수가 짹수인 각 열에 대하여 그 열에 있는 블록의 개수의  $\frac{1}{2}$ 만큼의 블록을 그 열에서 들어낸다.

블록을 들어내는 시행을 모두 마쳤을 때, 1열부터  $m$ 열까지 남아있는 블록의 개수의 합을  $f(m)$ 이라 하자. 예를 들어  $f(2) = 2, f(3) = 5, f(4) = 6$ 이다.

$$\lim_{n \rightarrow \infty} \frac{f(2^{n+1}) - f(2^n)}{f(2^{n+2})} = \frac{q}{p}$$

일 때,  $p + q$  (단,  $p$ 와  $q$ 는 서로소인 자연수)를 구하시오.



```
In [12]: import matplotlib.pyplot as plt
import numpy as np

#n=int(input())
#temp=n
def f(n):                      #한 열에서 블록의 개수 함수
    for i in range(n):
        if (n%2==0): n=n/2
        else: break
    return n

def g(n):                      #어떤 열 까지의 합 함수
    s=0
    for i in range(n+1):
        s+=f(i)
    return s

plt.plot([1, 2, 3, 4], [g(2), g(4), g(8), g(16)])
plt.plot([1,2,3],[g(4)-g(2),g(8)-g(4),g(16)-g(8)])
plt.show()

plt.figure(figsize=(6,8))

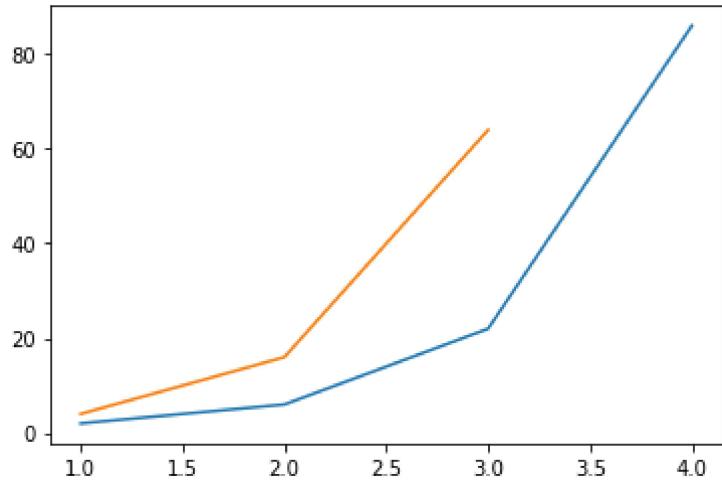
#파란색과 노란색의 이름을 붙이고 그래프의 크기를 조절할 수 있었으면 좋겠다.
#파란색은  $g(2^{**}n)$ , 노란색은  $g(2^{**}n+1)-g(2^n)$ 이다.

#구체적인 값을 구하기 위해 표로 정리한 것

print("n")
for i in range(4):
    print(i+1, end=' ')
print('Wn')
print("g(n)")

for i in range(4):
    print(int(g(2**i)), end=' ')
print('Wn')
print("g(n+1)-g(n)")

for i in range(4):
    print(int(g(2**i+1)-g(2**i)), end=' ')
plt.show()
```



n  
1 2 3 4

g(n)  
1 2 6 22

g(n+1)-g(n)  
1 4 16 64

<Figure size 432x576 with 0 Axes>

## 시각화 결과 분석

수열 문제에서는 이웃한 항들간의 점화 관계를 찾는 것이 매우 중요하다. 점화 관계를 찾는 과정에서 규칙이 잘 파악되지 않는 경우가 많은데 이때는  $n$ 에 1, 2, 3, 4와 같은 간단한 값을 대입하여 규칙성을 유추하는 것이 일반적이다. 그러나 위 문제에서는 간단한 수를 대입하여도 실수를 할 확률이 높고 함수 $g$ 에 들어가는 값이 2의 거듭제곱 형태로 기하급수적인 증가를 하므로 직접 세는 것에는 한계가 있다. 따라서 이들 함수의 그래프를 이용하면 쉽게 경향성을 파악하고 지수함수의 형태로 점화관계가 나타나는 것을 확신할 수 있다.

## 4. Plotly를 이용한 초월함수의 그래프 개형 분석

### 1) 다음 함수의 그래프 개형에 대해 탐구해보자

$$y = \frac{\ln x}{x^n}$$

In [149]:

위에서  $\ln x/x$   $\ln x/x^2$ 에 대해 탐구해보았다.  $\ln x/x^n$ 의 그래프에서  $n$ 이 변화할 때 그래프적분했을 때 발산하는 그래프와 수렴하는 그래프에 대해 알아보자~! 그리고 또한  $\ln x/x^n$ 에서  $1/(n-1)^2$  꼴로 나오는데, 이것 또한 구현하려 했으나 약간의 어려움이 있었다. 슬라이더가 보고 싶다. 이것을 Araboza

'''

```
import plotly.graph_objects as go
import numpy as np
from sympy import *
import sympy as sp

fig = go.Figure()

for step in np.arange(1, 5.1, 0.1): # 변하는 값이 가지는 범위
    fig.add_trace(
```

```

        go.Scatter(
            visible=False,
            name=str(step),
            line=dict(color='red', width=3),
            x=np.arange(0.8, 10, 0.01), # 정의역
            y=np.log(np.arange(0.8, 10, 0.01))/(np.arange(0.8, 10, 0.01)**(step))
        )
    )

def Int(i):
    j = 1 + 0.1*i
    x = sp.symbols('x')
    y = sp.log(x)/x**j
    return str(integrate(y, (x, 1, oo)))

steps = []
for i in range(len(fig.data)):
    step = dict(
        method='update',
        args=[
            {'visible': [False] * len(fig.data)},
            {'title': 'n = ' + str(round(1+0.1*i, 2)) + ': int(ln(x)/x^n, 1, oo) = ' -}
        ]
    )
    step['args'][0]['visible'][i] = True
    steps.append(step)

sliders = [
    dict(
        active=0,
        currentvalue={'prefix': '분모차수 = 1+ 0.1*step-n: '},
        pad={'t': 50},
        steps=steps
    )
]

fig.update_layout(
    sliders=sliders
)

fig.show()

```

## 시각화 결과 분석

위에서 시각화해본 그래프는 수능 수학 시험이나 모의고사에 자주 사용되는 함수 형태이다. 출제 빈도가 높은 함수인만큼 그 개형에 대해 익숙해져 있어야 한다고 생각하여서 시각화하였고 n에 따른 변화를 잘 알 수 있었다.

```
In [42]: import sympy as sp  
x = sp.symbols('x')  
y = sp.log(x)/(x**0)  
print(integrate(y, (x, 1, oo)))
```

oo

```
In [43]: import sympy as sp  
x = sp.symbols('x')  
y = sp.log(x)/(x)  
print(integrate(y, (x, 1, oo)))
```

oo

```
In [11]: import sympy as sp  
x = sp.symbols('x')  
y = sp.log(x)/(x**1.0000000001)  
print(integrate(y, (x, 1, oo)))
```

9.99999834519279e+19

```
In [12]: import sympy as sp  
x = sp.symbols('x')  
y = sp.log(x)/(x**1.0000001)  
print(integrate(y, (x, 1, oo)))
```

9999999983226.6

```
In [13]: import sympy as sp  
x = sp.symbols('x')  
y = sp.log(x)/(x**1.0001)  
print(integrate(y, (x, 1, oo)))
```

100000000.000022

```
In [14]: import sympy as sp  
x = sp.symbols('x')  
y = sp.log(x)/(x**1.1)  
print(integrate(y, (x, 1, oo)))
```

99.999999999998

```
In [15]: import sympy as sp
x = sp.symbols('x')
y = sp.log(x)/(x**1.5)
print(integrate(y, (x, 1, oo)))
```

4.000000000000000

```
In [18]: import sympy as sp
x = sp.symbols('x')
y = sp.log(x)/(x**2)
print(integrate(y, (x, 1, oo)))
```

1

```
In [19]: import sympy as sp
x = sp.symbols('x')
y = sp.log(x)/(x**2.5)
print(integrate(y, (x, 1, oo)))
```

0.444444444444444

```
In [20]: import sympy as sp
x = sp.symbols('x')
y = sp.log(x)/(x**3)
print(integrate(y, (x, 1, oo)))
```

1/4

```
In [21]: import sympy as sp
x = sp.symbols('x')
y = sp.log(x)/(x**4)
print(integrate(y, (x, 1, oo)))
```

1/9

```
In [22]: import sympy as sp
x = sp.symbols('x')
y = sp.log(x)/(x**5)
print(integrate(y, (x, 1, oo)))
```

1/16

```
In [23]: import sympy as sp
x = sp.symbols('x')
y = sp.log(x)/(x**1000)
print(integrate(y, (x, 1, oo)))
```

1/998001

## 2) 표준정규분포의 확률밀도함수

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

```
In [152]: import plotly.graph_objects as go
import numpy as np

fig = go.Figure()

for step in np.arange(1, 5.1, 0.1): # 변하는 값이 가지는 범위
    fig.add_trace(
```

```
go.Scatter(  
    visible=False,  
    name=str(step),  
    line=dict(color='cyan', width=3),  
    x=np.arange(-5, 5, 0.01), # 정의역  
    y=(1 / (np.sqrt(2 * np.pi) * step) * np.exp(-np.arange(-5, 5, 0.01)**2 /  
        )  
    )  
  
# fig.data[10].visible=True  
  
steps = []  
for i in range(len(fig.data)):  
    step = dict(  
        method='update',  
        args=[  
            {'visible': [False] * len(fig.data)},  
            {'title': 'sigma = ' + str(round(1+0.1*i, 2))}  
        ]  
    )  
    step['args'][0]['visible'][i] = True  
    steps.append(step)  
  
sliders = [  
    dict(  
        active=0,  
        currentvalue={'prefix': 'sigma = 1+ 0.1*step-n: '},  
        pad={'t': 50},  
        steps=steps  
    )  
]  
  
fig.update_layout(  
    sliders=sliders  
)  
  
fig.show()
```

시각화 결과 분석: 표준편차에 따른 확률밀도함수의 개형에 대해 더 잘 알게 되었다.

## 5. 시각화를 통한 이차곡선 분석

이차곡선이  $x^2 - 4x + 9y^2 - 5 = 0$ 과 중심이 점 (2, 0)이고 반지름의 길이가 a인 원이 서로 다른 네 점에서 만날 때, 실수 a의 값의 범위를 구하여라!

```
In [3]: colors = ['red', 'orange', 'yellow', 'green', 'blue']

import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(6, 9)) # 이미지 크기 설정
plt.title('WWfrac{(x-2)^2}{9} + y^2 = 1') # 이미지 제목 설정

plt.axhline(0, 0, 1, c='black') # x축 표시
plt.axvline(0, 0, 1, c='black') # y축 표시

"""
좌표축에 점 표시하기
"""

a = np.arange(-1, 6, 1)

plt.plot(a, a - a, 'bo', color='black', markersize=5)
plt.plot(a - a, a, 'bo', color='black', markersize=5)

"""
타원 그래프 표시하기
"""

X = np.linspace(-1, 5, 1024)
Ye_plus = (1 - (X - 2) ** 2 / 9) ** 0.5
Ye_minus = -(1 - (X - 2) ** 2 / 9) ** 0.5

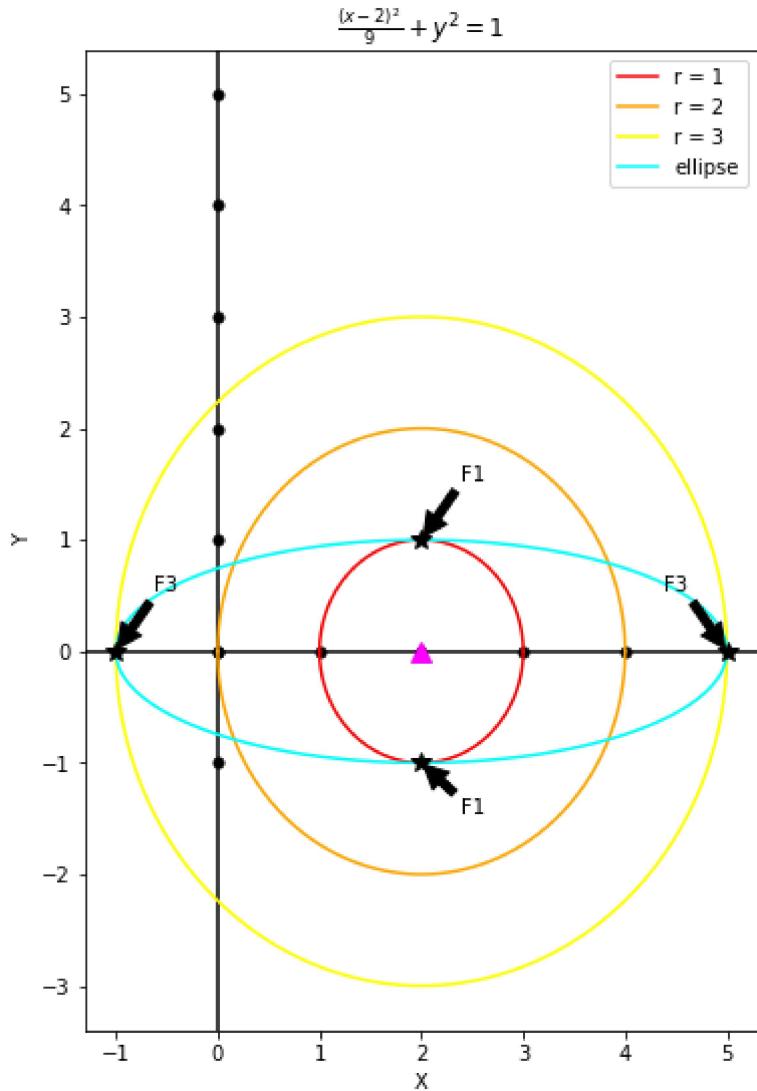
"""
반지름 a값에 따른 원 그래프 표시하기
"""

for a in range(1, 4):
    X_temp = np.linspace(-a + 2, a + 2, 1024)
    Yc_plus = (a**2 - (X_temp - 2) ** 2) ** 0.5
    Yc_minus = -(a**2 - (X_temp - 2) ** 2) ** 0.5
    plt.plot(X_temp, Yc_plus, c=colors[a - 1], label='r = {}'.format(a))
    plt.plot(X_temp, Yc_minus, c=colors[a - 1])

"""
점 표시하기
"""

plt.annotate(
    'F1',
    ha = 'center', va = 'bottom',
    xytext = (2.5, 1.5),
    xy = (2, 1),
    arrowprops = {'facecolor': 'black', 'shrink': 0.05}
)
```

```
plt.annotate(  
    'F1',  
    ha = 'center', va = 'bottom',  
    xytext = (2.5, -1.5),  
    xy = (2, -1),  
    arrowprops = {'facecolor': 'black', 'shrink': 0.05}  
)  
  
plt.annotate(  
    'F3',  
    ha = 'center', va = 'bottom',  
    xytext = (-0.5, 0.5),  
    xy = (-1, 0),  
    arrowprops = {'facecolor': 'black', 'shrink': 0.05}  
)  
  
plt.annotate(  
    'F3',  
    ha = 'center', va = 'bottom',  
    xytext = (4.5, 0.5),  
    xy = (5, 0),  
    arrowprops = {'facecolor': 'black', 'shrink': 0.05}  
)  
  
plt.plot(X, Ye_plus, c='cyan', label='ellipse')  
plt.plot(X, Ye_minus, c='cyan')  
plt.plot([2], [0], marker='^', c='magenta', markersize=10)  
plt.plot([2], [1], marker='*', c='black', markersize=10)  
plt.plot([2], [-1], marker='*', c='black', markersize=10)  
plt.plot([-1], [0], marker='*', c='black', markersize=10)  
plt.plot([5], [0], marker='*', c='black', markersize=10)  
  
plt.xlabel('X')  
plt.ylabel('Y')  
  
plt.legend(loc='best', fancybox=True)  
plt.show()
```



## 6. 시각화를 통한 수치적분/몬테카를로 적분 분석

### 6.1 수치적분 관련 개념

수치 적분(Numerical Integral)이란 임의의 구간에서 피적분함수를 포함한 적당한 급수함으로 근사하여 수치적으로 구하는 것을 말한다. 수치적분의 장점은 역도함수를 구하기 어렵거나 구할 수 없는 함수들은 그 그래프의 밑넓이를 구하는데 용이하다는 것이다. 그렇기에 대부분 수치적분은 해석적으로 적분이 안되는 경우에 사용한다. 수치적분은 주어진 부분구간(suinterval)의 급수들의 합을 이용하여 밑넓이를 근사적으로 구할 수 있다.

수치적분은 수치해석의 한 종류로, 정확한 값이 아니라 근사적으로 값을 구하는 것에 중점을 두고 있다. 다른 근사적인 방법으로는 보간법(알려진 자료(로그표, 정규분포표, 함숫값)를 이용하여 알려지지 않은 것들을 구하는 방법), 보외법(원래의 관찰 범위를 넘어서서 다른 변수와의 관계에 기초하여 변수의 값을 추정하는 과정이다. 관찰된 값들 사이의 추정치를 만들어내는 보간법과 비슷 하지만 보외법은 더 큰 불확실성과 무의미한 결과 생성에 대한 더 높은 위험에 종속된다.)이 있다.

수치적분하는 방법들로는 사다리꼴방법(단순하지만 오차가 크다.) ->합성사다리꼴 공식 (구분구적법과 유사하게 시행) :주어진 다항식을 1차다항식으로 근사하여 적분값을 구한다. ->사다리꼴 공식에 대한 리차드슨 보외법44(보외법 답게 근사값을 이용하여 좀 더 나은 적분값을 구해보자는 것이다.) 직사각형 중간점 심슨 1/3 공식이용: 함수를 이차함수로 근사시켜 좀 더 정확한 적분값을 얻는 방법을 말한다.(적분구간이 작을때는 실제 적분값과 크게 차이가 없으나 적분 구간이 커지면 오차가 커진다) -> 합성 심슨 공식-> 심슨 3/8 공식 피적분함수 3차다항식 근사 (적분 구간 세구간

이라는데 블로그 그림보기) 뉴튼-코츠 적분(구간 크게, 다항식 차수 증가-> 정확도 증가) -> 롬베  
르크 적분(리처드슨 보외법을 이용하여 정적분값을 구하는 방법, 뉴튼 코츠에 한 종류) 2점 가우스  
구적법-> 2개말고 n개 n점 가우스 구적법 몬테 카를로

수치적분 해볼 함수들

1. 최고항 오차인 함수 정도의 다항함수
2. 정규분포
3.  $\sin(x^2)$ ,  $1/\ln x$ ,  $\sqrt{1+x^4}$  같은 함수들

## 6.2 여러가지 수치적분

- 이 파트에서는 여러 가지 수치적분 방법에 대한 간략한 소개와 함께 여러 함수들을 수치적분 방법에 따라 계산해볼 예정이다. 각 수치적분 방식별로 오차가 어느 정도 되는지 확인해보며, 이론적으로 나와있는 오차 정도와 비교해볼 예정이다.

```
In [4]: # P.S) Scipy의 integrate 서브패키지의 quad 명령으로 수치적분을 해볼 수도 있다
# 첫 번째 숫자는 수치적분 값이며, 두 번째 숫자는 오차의 상한값을 의미한다.
from scipy import integrate
import scipy as sp

def f(x):
    return x**4 + 3*x + 5 #함수 f(x) 지정

sp.integrate.quad(f, 0 ,3) #구간[0,3]에 대한 정적분
```

Out[4]: (77.1, 8.559819519859956e-13)

### 1) 직사각형 방법

직사각형 방법은 함수의 밑넓이를 밑변의 길이는 동일하며 높이는 함숫값에 의하여 결정되는 직사각형들의 합으로 나타내는 방법이다.

```
In [6]: import random
def f(x):
    return x**2

a, b, n, k_1, k_2, k_3, k_4 = 0, 3, 20, 0, 0, 0, 0

#왼쪽 리만합
for i in range(n):
    k_1 = k_1 + f(a + i*(b - a) / n)*(b - a) / n
print("Left Riemann Sum:", k_1)

#오른쪽 리만합
for i in range(1, n+1):
    k_2 = k_2 + f(a + i*(b - a) / n)*(b - a) / n
print("Right Riemann Sum:", k_2)

#중간점 이용
# 0.5인 경우
for i in range(0, n):
    k_3 = k_3 + f( a + (i+1/2)*(b - a) / n)*(b - a) / n
print("Midpoint(0.5) Riemann Sum:", k_3)

# 난수로 해본 경우
for i in range(0, n):
```

```
k_4 = k_4 + f( a + (i+random.random())*(b - a) / n )*(b - a) / n
print("Midpoint(random) Riemann Sum:", k_4)
```

Left Riemann Sum: 8.33625  
 Right Riemann Sum: 9.68625  
 Midpoint(0.5) Riemann Sum: 8.994374999999998  
 Midpoint(random) Riemann Sum: 9.078085686215903

```
In [7]: import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

f = lambda x : x**2
a = 0; b = 2; N = 20
n = 10 # Use n*N+1 points to plot the function smoothly

x = np.linspace(a,b,N+1)
y = f(x)

X = np.linspace(a,b,n*N+1)
Y = f(X)

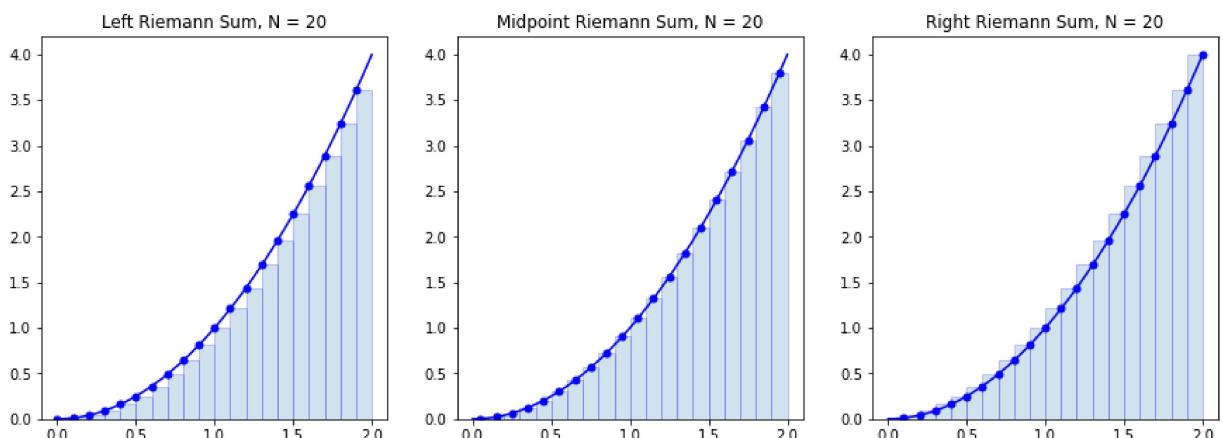
plt.figure(figsize=(15,5))

plt.subplot(1,3,1)
plt.plot(X,Y,'b')
x_left = x[:-1] # Left endpoints
y_left = y[:-1]
plt.plot(x_left,y_left,'b.',markersize=10)
plt.bar(x_left,y_left,width=(b-a)/N,alpha=0.2,align='edge',edgecolor='b')
plt.title('Left Riemann Sum, N = {}'.format(N))

plt.subplot(1,3,2)
plt.plot(X,Y,'b')
x_mid = (x[:-1] + x[1:])/2 # Midpoints
y_mid = f(x_mid)
plt.plot(x_mid,y_mid,'b.',markersize=10)
plt.bar(x_mid,y_mid,width=(b-a)/N,alpha=0.2,edgecolor='b')
plt.title('Midpoint Riemann Sum, N = {}'.format(N))

plt.subplot(1,3,3)
plt.plot(X,Y,'b')
x_right = x[1:] # Left endpoints
y_right = y[1:]
plt.plot(x_right,y_right,'b.',markersize=10)
plt.bar(x_right,y_right,width=(b-a)/N,alpha=0.2,align='edge',edgecolor='b')
plt.title('Right Riemann Sum, N = {}'.format(N))

plt.show()
```



## 2) 사다리꼴 방법

5.1.1에서의 직사각형 대신 사다리꼴을 이용한 방법이다. 사다리꼴이 직사각형 보다는 함수 사이의 면적이 좁으므로 비교적 오차가 적다고 말할 수 있을 것이다. 사다리꼴 방식의 근사적인 수식은 다음과 같다.

$$\int_a^b f(x)dx \approx \sum_{k=1}^{n-1} \left[ \frac{f(x_{i-1}) + f(x_i)}{2} \right] \left( \frac{b-a}{n} \right) = \frac{b-a}{2n} [f(x_0) + 2f(x_1) + 2f(x_2) + \dots + 2f(x_{n-1}) + f(x_n)]$$

다음 수식을 이용하여 5.1.1에서와 동일한 구간, 동일한 분할 횟수에서 넓이가 어떻게 되는지 확인해보자.

```
In [8]: def f(x):
    return x**2
a, b, n, k = 0, 3, 20, 0
for i in range(1, n+1):
    k = k + (f(a + (i-1)*(b - a) / n) + f(a + i*(b - a) / n)) * (b - a) / (2*n)
print("Trapezoidal rule:", k)
```

Trapezoidal rule: 9.01125

## 3) 리처드슨 보외법을 이용한 사다리꼴 방법

두 개의 서로 다른 구간의 크기를 가진 적분 근삿값을 이용하여 좀 더 정확한 적분 근삿값을 얻을 수 있다. 사다리꼴 방법에 대한 리처드슨 보외법은 다음과 같다.

$$T.V. \approx \frac{4}{3}(A.V.)_{2n} - \frac{1}{3}(A.V.)_n$$

시행횟수를 바꿔가며 좀 더 정확한 정적분 값을 추정해보자.

```
In [9]: def f(x):
    return x**2
a, b, n, k, l = 0, 3, 20, 0, 0

for i in range(1, n+1):
    k = k + (f(a + (i-1)*(b - a) / n) + f(a + i*(b - a) / n)) * (b - a) / (2*n)

for j in range(1, 2*n+1):
    l = l + (f(a + (j-1)*(b - a) / (2*n)) + f(a + j*(b - a) / (2*n))) * (b - a) / (2*n)

m = (4*l - k)/3
print("Trapezoidal rule with :", m)
```

Trapezoidal rule with : 9.000000000000004

## 4) 심슨 1/3 공식을 이용한 수치적분

직사각형 방법과 사다리꼴 방법이 피적분함수  $f(x)$ 를 일차함수로 근사시켜 그 합을 구하는 방법이었다면 심슨 1/3 공식은 피적분함수  $f(x)$ 를 이차 다항식에 근사시켜 좀 더 정확한 적분 값을 얻는 방법이다. 이 방법도 단일 이차함수 하나의 넓이가 아니라 여러 구간을 나눠 각 부분의 이차함수 넓이의 합을 구할 시 오차가 줄어든다고 말할 수 있다. 심슨의 법칙은 다음과 같다.

$$\int_a^b f(x)dx \approx S_n \approx \frac{h}{3} [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \dots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]$$

여기서  $n$ 은 짝수이고,  $h = \frac{b-a}{n}$  이다.

```
In [ ]: def f(x):
    return x**2
```

```

a, b, n = 0, 3, 20
h = (b - a) / n
k = (f(a) + f(b))*h/3
for i in range(1, n):
    if i%2 == 1:
        k = k + 4*f(a + i*(b - a) / n )*h / 3
    if i%2 == 0:
        k = k + 2*f(a + i*(b - a) / n )*h / 3
k

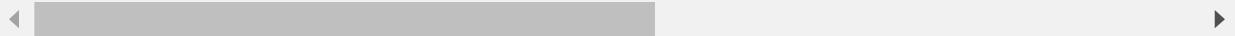
```

## 5) 심슨 3/8 공식을 이용한 수치적분

심슨 1/3 공식과 달리 피적분함수  $f(x)$ 를 3차 다항식으로 근사하여 적분값을 얻는 방법을 심슨 3/8 공식이라 한다. 심슨 1/3 공식을 이용할 경우 근사 구간  $[a,b]$ 에 대하여  $a, b, \frac{a+b}{2}$  두 구간으로 나누었다면, 심슨 3/8공식은 적분 구간을  $a, \frac{2a+b}{3}, \frac{a+2b}{3}, b$ 로 3등분해야 한다는 차이가 있다. 미정계수법, 라그랑주 보간법 등을 이용하여 심슨 3/8 공식을 유도해보면 다음과 같다.

$$\int_a^b f(x)dx \approx \frac{3h}{8}[f(x_0) + 3f(x_1) + 3f(x_2) + 2f(x_3) + 3f(x_4) + 3f(x_5) + 2f(x_6) + \dots + 3f(x_{n-1})]$$

for  $k \in \mathbb{N}$



## 6) 뉴튼-코츠 공식을 이용한 수치적분

뉴튼-코츠 공식은 아이작 뉴튼(Sir Isaac Newton)과 로처 코츠(Roger Cotes)의 이름을 딴 수치적분의 공식군이다. 피적분함수  $f(x)$ 를  $n$ 차 다항식으로 근사한 뒤 적분하는 방법을 말한다. 이때 사다리꼴 공식 또는 심슨 공식은 뉴튼 - 코츠 공식의 일부인 것으로 생각하면 된다. 뉴튼-코츠 적분법을 일반화하면 다음과 같다.

\$\$

분할 구간이 많아질수록, 근사하는 다항식의 차수가 증가할수록 오차가 줄어든다고 할 수 있다.

## 7) $n$ 점 가우스 구적법을 이용한 수치적분

사다리꼴 방법을 이용하여 적분을 할 때 적분 구간의 양 끝점을 피적분함수의 함숫값을 이용하는 것이 아니라 적분 구간 내의 다른 두 점의 위치를 적당히 잘 정한다면 좀 더 정확한 적분값을 얻을 수 있지 않을까?라는 생각으로 부터 나오게 되었다. 가우스 구적법을 일반화하면 다음과 같다.

## 8) 몽테카를로 적분을 이용한 수치적분

```

In [10]: import numpy as np
         from matplotlib import pyplot as plt

def function(x): ### 함수 정의
    return x**2 + np.sin(np.pi*x)

if __name__ == '__main__':
    N = int(1e+4) ### 랜덤 샘플링 시행 횟수 정의
    W, H = 1, 1.4 ### 랜덤 샘플링을 할 사각형 R의 가로 세로 정의

    X = np.random.random(N) ### 각 점의 x 좌표 랜덤 샘플링
    Y = H*np.random.rand(N) ### 각 점의 y 좌표 랜덤 샘플링

    F = function(X) ### 각 랜덤 샘플링에 대한 f(x)를 계산
    in_out = Y < F ### y와 f(x) 값 비교

```

```
A = H * W * np.sum(in_out) / N ### 영역 S의 넓이
print("A = ", A)
```

A = 0.9786

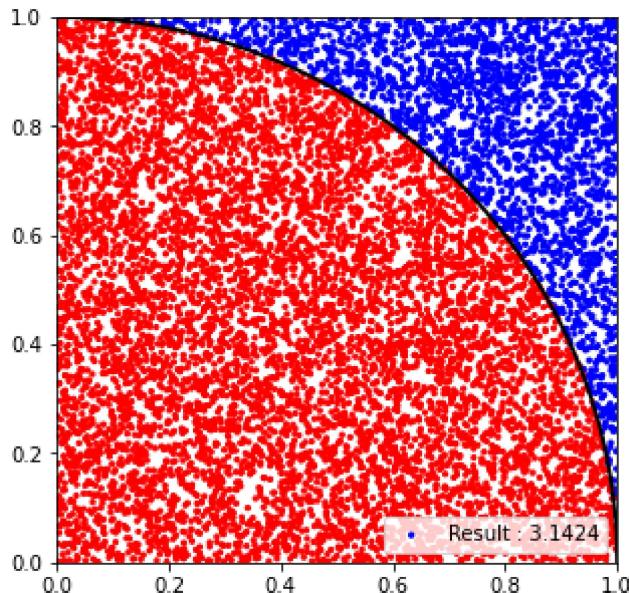
```
In [11]: #파이썬을 이용한 몬테카를로 적분 구현
import numpy as np
from multiprocessing import Pool
from matplotlib import pyplot as plt

if __name__ == '__main__':
    N = 10000 ### 무작위시행 횟수 정의
    x = np.random.random([N, 2])
    distance = np.sum(x ** 2.0, axis=1)
    in_out = distance <= 1.0
    pi = np.sum(in_out)*4/N ### Pi 값은 천제 시행에서 원 안에 있는 점의 갯수로 정해짐
    color = list(map(lambda x: 'red' if x else 'blue', in_out)) ### 원의 안, 밖에 따

    plt.figure(figsize=(5, 5)) ### 그림 사이즈
    plt.scatter(x[:,0], x[:,1], color = color, s=5, label ='Result : {}'.format(np.ro

    cx = np.cos(np.linspace(0, np.pi/2, 1000))
    cy = np.sin(np.linspace(0, np.pi/2, 1000))
    plt.plot(cx, cy, color = 'black', lw =2) ### 원의 경계를 그려주는 부분
    plt.legend(loc = 'lower right')

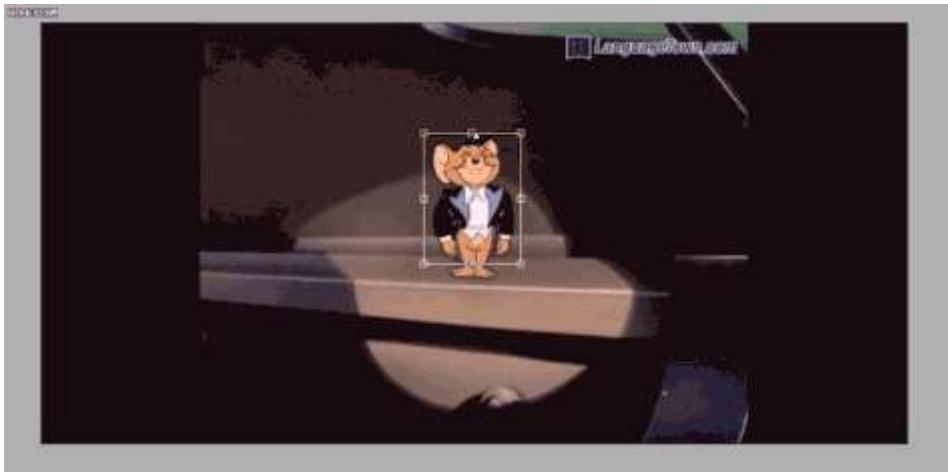
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.show()
```



## V. 결론 및 논의

### 참고 문헌

- [1] 웨스 맥키니, 파이썬 라이브러리를 활용한 데이터 분석, 김영근, 한빛미디어, 2019, p.28-p.32
- [2] <https://docs.sympy.org/latest/index.html>



봐주셔서 감사합니다!