파이썬 수치해석

Chapter 1. 보간법

박형묵

명신여자고등학교

Update 20250403

강의 자료 다운로드

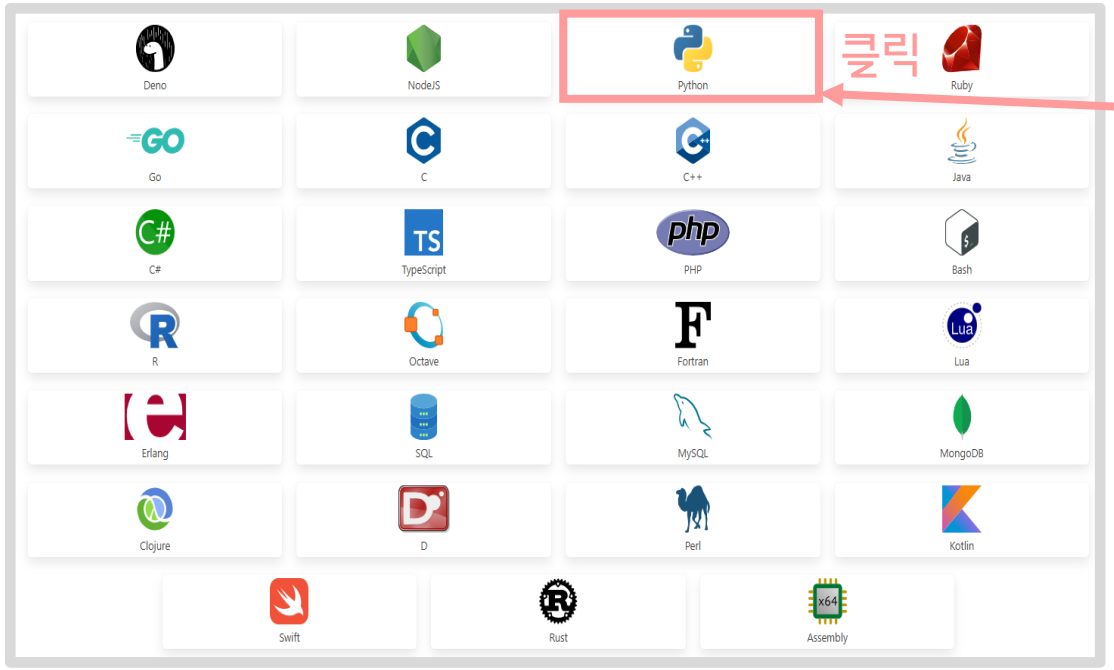
파이썬 수치해석 강의 자료

https://github.com/PigeonDove/PythonNumericalAnalysis

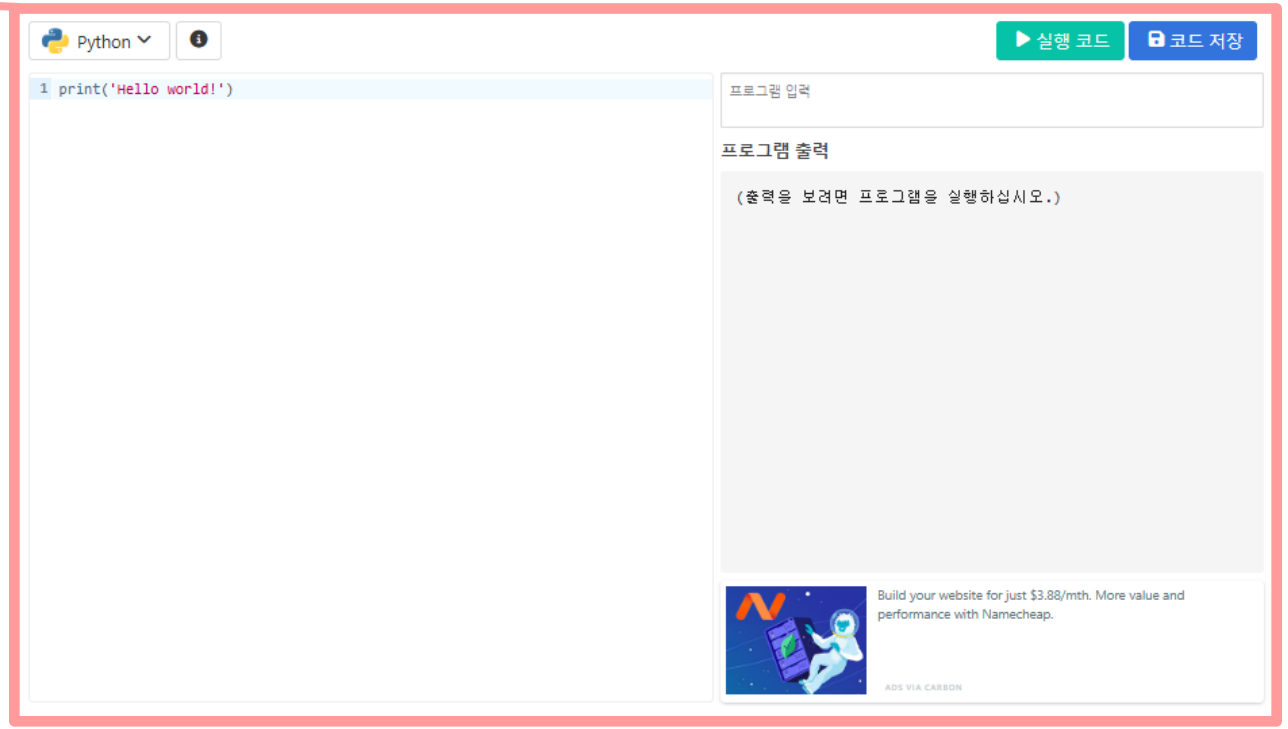
개발 환경

myCompiler 의 Python

파이썬 코딩 웹 사이트



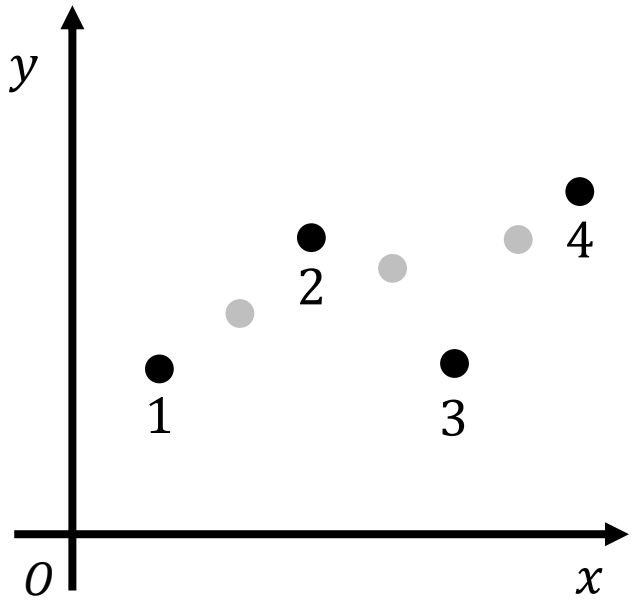
https://www.mycompiler.io/ko/new/python



보간법

보간법이란 무엇인가?

기본 개념 학습



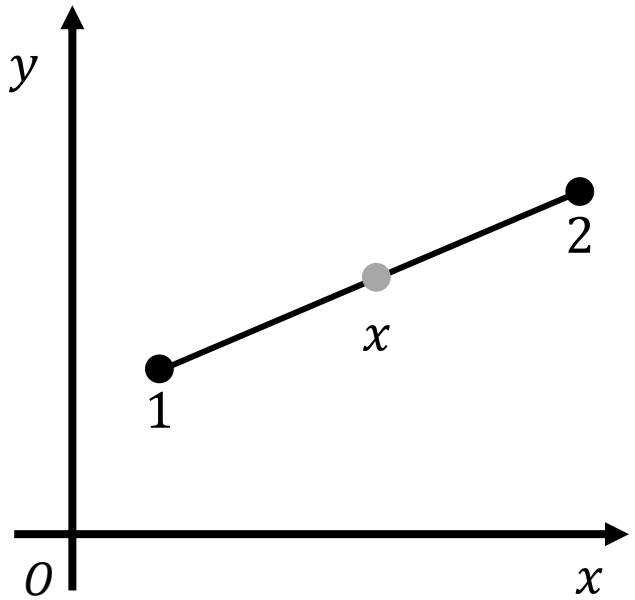
	1	2	3	4
x	1.222	3.535	5.232	7.111
y	3.323	5.555	3.562	6.232

정확한 데이터 점들 사이 있는 중간 값을 추정하는 방법

보간법

선형 보간법

기본 개념 학습



두 점의 기울기를 이용해 사이 값(y_n)을 계산하는 방법

두 점의 기울기 m은 다음과 같이 정의 된다.

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

임의의 점 x에서의 기울기를 구하면

$$\frac{y - y_1}{x - x_1} = m$$

여기에 $m = \frac{y_1 - y_1}{x_2 - x_1}$을 대입하면

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$$

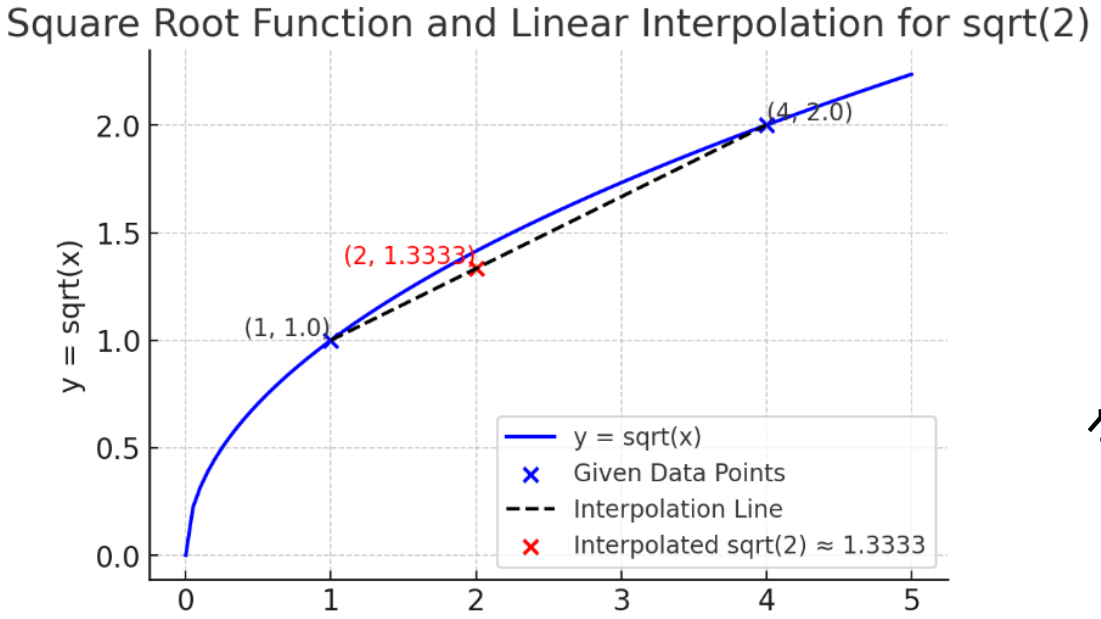
이 식을 y에 대해서 정리

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$$

보간법

선형 보간법

$\sqrt{2}$ 값 계산해보기



기본 개념 적용

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$$

$(x_1, y_1) = (1, 1)$
$(x_2, y_2) = (4, 2)$ 에서 $x_n = 2$ 를 대입하여 y_n 값을 구하면

$$y = \frac{2-1}{4-1}(2-1) + 1 = \frac{1}{3} + 1 = 1.3333$$

실제 $\sqrt{2}$의 값은 1.4142이나, 선형 보간법을 이용한 값은 1.3333 이다.

$$Error = \frac{|\text{실제 값} - \text{계산 값}|}{\text{실제 값}} \cdot 100$$

$$Error = \frac{|1.4142 - 1.3333|}{1.4142} \cdot 100 = 5.72\%$$

보간법

선형 보간법

소스 코드

$\sqrt{2}$ 값 계산해보기

1-1.py

```python
import numpy as np  # numpy 라이브러리 불러오기
import matplotlib.pyplot as plt  # 그래프를 그리기 위한 matplotlib 라이브러리 불러오기

# 실제 sqrt(2)의 값 (numpy의 sqrt 함수 이용)
actual_value = np.sqrt(2)

# 선형 보간법을 위한 두 개의 데이터 포인트 설정
x1, y1 = 1, np.sqrt(1)  # 첫 번째 데이터 포인트 (1, 1)
x2, y2 = 4, np.sqrt(4)  # 두 번째 데이터 포인트 (4, 2)

# 선형 보간법 공식 적용하여 sqrt(2) 값을 근사적으로 계산
x_pred = 2  # 보간을 수행할 x 값 (즉, sqrt(2)의 x 값)
y_pred = ((x_pred - x1) / (x2 - x1)) * (y2 - y1) + y1  # 선형 보간법 공식 적용

# 오차 계산 (상대 오차)
relative_error = (abs(actual_value - y_pred) / actual_value) * 100  # 상대 오차 계산

# 결과 출력 (콘솔 출력)
print(f"Actual sqrt(2) Value: {actual_value:.4f}")  # 실제 sqrt(2) 값 출력
print(f"Interpolated sqrt(2) Value: {y_pred:.4f}")  # 선형 보간법을 이용한 근사 sqrt(2) 값 출력
print(f"Relative Error: {relative_error:.2f}%")  # 상대 오차 출력
```

```
Actual sqrt(2) Value: 1.4142
Interpolated sqrt(2) Value: 1.3333
Relative Error: 5.72%
```

그래프는 1-2.py

보간법

선형 보간법

문제점

1. 두 점 사이의 거리가 멀수록 보간 된 값의 신뢰도가 낮아진다.
   → 두 구간 사이를 좁혀서 해결

2. 두 직선으로 연결하여 중간 값을 예측하기 때문에 비선형 데이터에 대해 부정확 하다.
   → **2차 보간법 사용**

보간법

2차 보간법

라그랑주 1차 보간법 공식

$$y = y_1 \frac{(x - x_2)}{(x_1 - x_2)} + y_2 \frac{(x - x_1)}{(x_2 - x_1)}$$

라그랑주 2차 보간법 공식

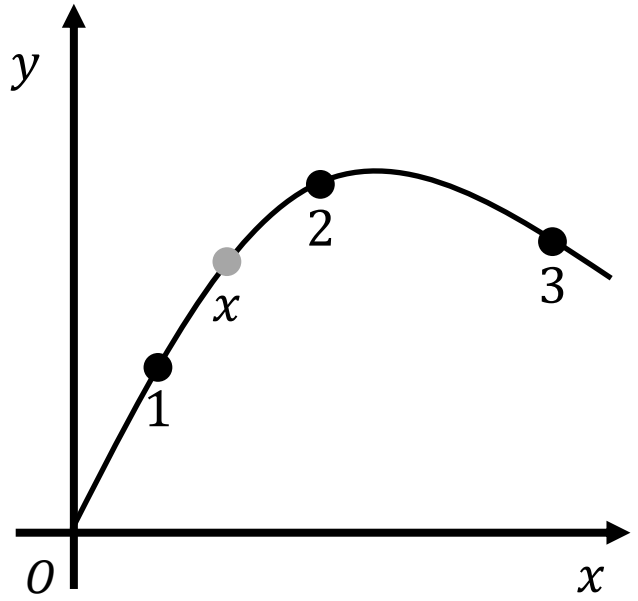
$$y = y_1 \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} + y_2 \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} + y_3 \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}$$

라그랑주 n차 보간법 공식

$$y = y_1 \frac{(x - x_2)(x - x_3)\cdots(x - x_{n+1})}{(x_1 - x_2)(x_1 - x_3)\cdots(x_1 - x_{n+1})} + y_2 \frac{(x - x_1)(x - x_3)\cdots(x - x_{n+1})}{(x_2 - x_1)(x_2 - x_3)\cdots(x_2 - x_{n+1})} + \cdots + y_{n+1} \frac{(x - x_1)(x - x_2)\cdots(x - x_n)}{(x_{n+1} - x_1)(x_{n+1} - x_2)\cdots(x_{n+1} - x_n)}$$

$$= \sum_{j=1}^{n+1} y_j \prod_{\substack{i=1 \\ i \neq j}}^{n+1} \frac{(x - x_i)}{(x_j - x_i)}$$

라그랑주 보간법



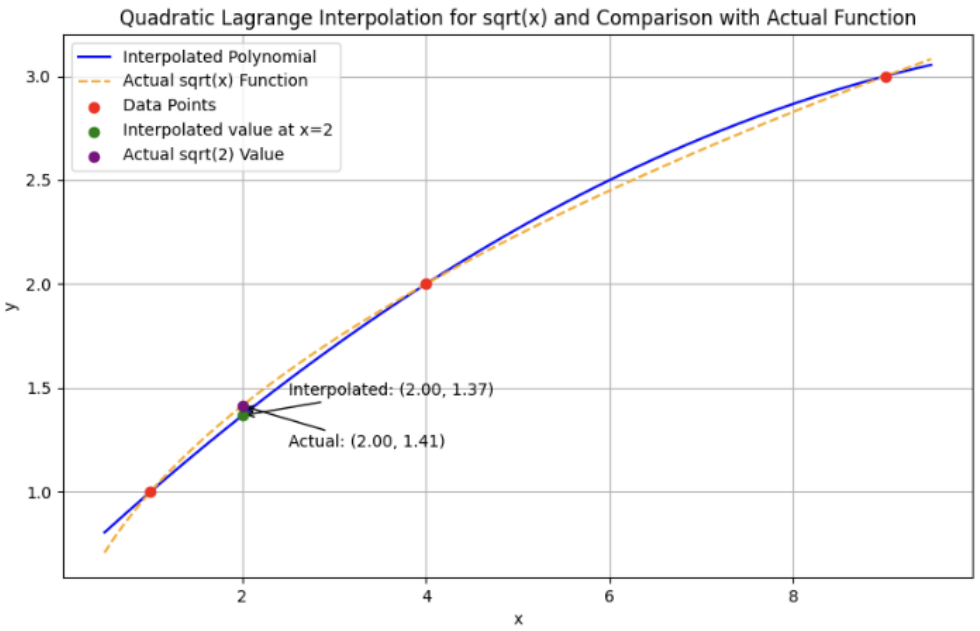
2차 식 보간법의 경우 그림 예제

n차 식의 보간법을 위해서는 n+1개의 점이 필요.
각 포인트에서 x_n의 경우 y_n이 나오도록 하는 식 정의

보간법

2차 보간법

$\sqrt{2}$ 값 계산해보기



기본 개념 적용

$\sqrt{2}$ 값을 계산하기 위해 세 개의 점을 선택한다

$$(x_1, y_1) = (1, \sqrt{1}) = (1, 1)$$
$$(x_2, y_2) = (4, \sqrt{4}) = (4, 2)$$
$$(x_3, y_3) = (9, \sqrt{3}) = (9, 3)$$

2차 식의 라그랑주 보간법 공식은 아래와 같으며, 각 자리에 맞는 값을 대입한다.

$$y = y_1 \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} + y_2 \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} + y_3 \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}$$

$$y = 1 \frac{(x - 4)(x - 9)}{(1 - 4)(1 - 9)} + 2 \frac{(x - 1)(x - 9)}{(4 - 1)(4 - 9)} + 3 \frac{(x - 1)(x - 4)}{(9 - 1)(9 - 4)}$$

$x = 2$를 대입하여 y값을 계산한다.

$$y = 1.366667$$

$$Error = \frac{|1.4142 - 1.4366667|}{1.4142} \cdot 100 = 3.3621\%$$

보간법

2차 보간법

소스코드

1-3.py

```python
import numpy as np                # numpy 모듈을 np라는 이름으로 불러옴
import matplotlib.pyplot as plt   # matplotlib의 pyplot 모듈을 plt라는 이름으로 불러옴

# f(x) = sqrt(x) 함수에 따른 데이터 포인트 정의 (np.sqrt 사용)
x1, y1 = 1, np.sqrt(1)   # 첫 번째 데이터 포인트: x=1, y=sqrt(1)=1
x2, y2 = 4, np.sqrt(4)   # 두 번째 데이터 포인트: x=4, y=sqrt(4)=2
x3, y3 = 9, np.sqrt(9)   # 세 번째 데이터 포인트: x=9, y=sqrt(9)=3

# 보간할 x 값 설정 (여기서는 sqrt(2)를 근사하기 위해 x=2 사용)
x_interp = 2

# x=2에서의 첫 번째 라그랑주 기저 다항식 계산
L1 = ((x_interp - x2) * (x_interp - x3)) / ((x1 - x2) * (x1 - x3))
# x=2에서의 두 번째 라그랑주 기저 다항식 계산
L2 = ((x_interp - x1) * (x_interp - x3)) / ((x2 - x1) * (x2 - x3))
# x=2에서의 세 번째 라그랑주 기저 다항식 계산
L3 = ((x_interp - x1) * (x_interp - x2)) / ((x3 - x1) * (x3 - x2))

# x=2에서의 보간값 P_x 계산 (각 데이터 포인트의 y값과 기저 다항식의 곱의 합)
P_x = y1 * L1 + y2 * L2 + y3 * L3

# 실제 sqrt(2) 값 계산 (np.sqrt 사용)
actual_value = np.sqrt(2)

# 실제 값과 보간값의 상대 오차 계산
relative_error = abs((actual_value - P_x) / actual_value) * 100

# 결과 출력: 실제 sqrt(2), 보간된 sqrt(2), 상대 오차
print("Actual sqrt(2) Value: {:.6f}".format(actual_value))
print("Interpolated sqrt(2) Value: {:.6f}".format(P_x))
print("Relative Error: {:.4f}%".format(relative_error))
```

```
Actual sqrt(2) Value: 1.414214
Interpolated sqrt(2) Value: 1.366667
Relative Error: 3.3621%
```

그래프는 1-4.py

보간법

2차 보간법

문제점

1. 데이터 포인트 3개만을 사용하므로, 실제 함수의 형태가 2차 이상일 경우 전체적인 변화를 충분히 반영하지 못해 근사 오차가 커질 수 있다.

2. 2차 보간은 주어진 3개 점 근처에서만 유효한 국소적 근사법이므로, 전체 함수의 복잡한 변화나 극값, 변곡점을 반영하기 어렵다.
   → **n차 보간법 사용**

감사합니다

박형묵

명신여자고등학교