고급통계프로그래밍 Solution 9

컴퓨터과학부 2017920054 이호준

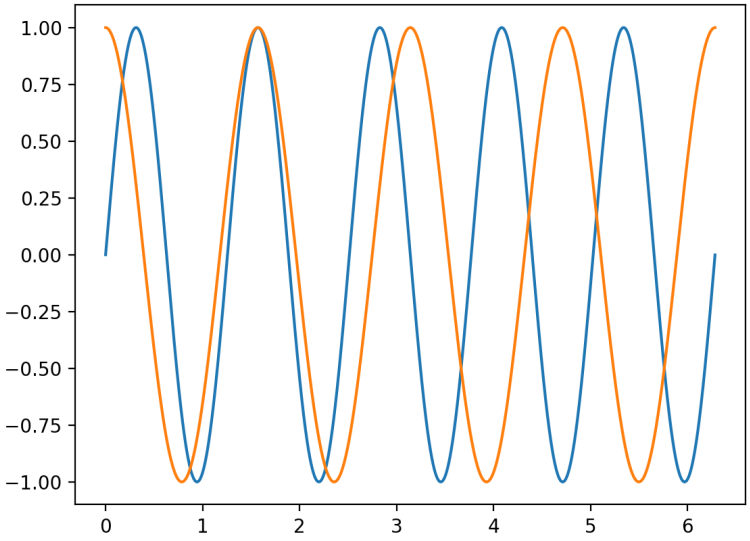
Ex 01

Solution

```
import matplotlib.pyplot as plt
import math
import numpy as np

x = [i for i in np.arange(0, 2*math.pi, 0.001)]

y_1 = [math.sin(5*i) for i in x]
y_2 = [math.cos(4*i) for i in x]

plt.plot(x, y_1)
plt.plot(x, y_2)
plt.show()
```

- 0부터 2π까지 0.001 간격으로 숫자가 담긴 정의역 리스트 `x` 를 생성한다.
- 이 정의역을 순회하면서 $sin(5x), cos(4x)$을 적용한 리스트 `y_1`, `y_2` 를 선언한다.
- matplotlib.pyplot(`plt`)의 `.plot()` 함수를 이용해서 x와 y_1, y_2 사이의 그래프를 생성한다.
- `plt.show()` 를 통해 그래프를 확인한다.

Result



- 파란색 그래프가 $sin(5x), 0 \leq x \leq 2\pi$, 주황색 그래프가 $cos(4x), 0 \leq x \leq 2\pi$ 이다.

Ex 02

Solution

```
from collections import Counter
import numpy as np
import math

class Data:
    def __init__(self, data):
        self.lst = data # data
        self.size = len(data) # data size
        self.sorted_lst = sorted(data) # sorted data

    # 평균 Mean 구하기
    def mean(self):
        return sum(self.lst) / self.size

    # 중위수 Median 구하기
    def median(self):
        if self.size % 2 == 1: # 중위수 있음
            return self.sorted_lst[self.size // 2]
        else: # 중위수 없음 - 중위값 둘의 중간값
            return (self.sorted_lst[self.size // 2]
                    + self.sorted_lst[self.size // 2 - 1]) / 2

    # 최빈값 Mode 구하기
    def mode(self):
```

```
        cnt = Counter(self.lst)
        max_cnt = max(cnt.values())
        return [i for (i , freq) in cnt.items() if freq == max_cnt ]

    # 최댓값 Max 구하기
    def max(self):
        return self.sorted_lst[-1]

    # 최솟값 Min 구하기
    def min(self):
        return self.sorted_lst[0]

    # 범위 Range 구하기
    def range(self):
        return self.max() - self.min()

    # 분산 Variance 구하기
    def variance(self):
        square_of_error = [(i - self.mean()) ** 2 for i in self.lst]
        try:
            return sum(square_of_error) / (self.size - 1)
        except ZeroDivisionError:
            print("Variance를 구하는 과정에서 self.size가 0이 되는 문제가 발생했습니다.")
            return -1

    # 표준편차 Std 구하기
    def std(self):
        return math.sqrt(self.variance())
```

- 데이터값과 이것의 기술통계량을 클래스와 메서드로 해결해보자.
  - Class `Data` - 데이터를 담고 처리하는 클래스
  - **Attribute** - 생성자를 통해 초기화된다.
    - `lst` : Data 값 (list)
    - `size` : Data 값의 개수 (int)
    - `sorted_lst` : 정렬된 Data값 (list)
  - **Method** - 각 이름에 해당하는 값을 반환하는 메서드
    - `mean()` : 평균을 반환하는 메서드
      - lst의 합(sum())을 size로 나눈 후 반한한다.
    - `median()` : 중위수 반환하는 메서드
      - 만약 size가 짝수라면 중위수의 인덱스가 특정되지 않으므로, 인덱스 size // 2와 size // 2 - 1의 중간값을 구한다.
        - ex) size == 8이라면 size // 2 == 4, 인덱스 3과 4의 값의 중간값을 반환한다.

        - 만약 size가 홀수라면 중위수가 인덱스가 특정되므로 이 인덱스에 해당하는 값을 반환한다.
            - ex) size == 7이라면 size // 2 == 3, 인덱스 3의 원소가 실제 중위값이므로 이를 반환한다.
    - `mode()` : 최빈값을 반환하는 메서드
        - Counter를 이용해 개수를 세준 후, max_cnt와 cnt가 같은 원소를 찾아 리스트로 묶어서 반환한다.
    - `max()` : 최댓값을 반환하는 메서드
        - 정렬된 리스트의 가장 마지막 인덱스 값(인덱스 -1)이 주어진 데이터의 최솟값이므로,
          이를 반환한다.
    - `min()` : 최솟값을 반환하는 메서드
        - 정렬된 리스트의 인덱스 0의 값이 주어진 데이터의 최솟값이므로, 이를 반환한다.
    - `range()` : 범위을 반환하는 메서드
        - self.max - self.min을 통해서 구할 수 있다.
    - `variance()` : 분산을 반환하는 메서드
        - 오차의 제곱값들을 구한 후, 이들을 더한 값을 `size-1` 로 나누어준 후 반환한다.
        - size == 1인 경우, ZeroDivisionError 예외처리 → -1을 반환한다.
    - `std()` : 표준편차을 반환하는 메서드
        - math.sqrt()를 이용해 variance의 제곱근을 반환한다.

Result

```
# Test 1
weight = [65, 78, 80, 140, 100, 75, 78]

x = Data(weight)

print("mean :", x.mean())
print("median :", x.median())
print("mode :", x.mode())

print("max :", x.max())
print("min :", x.min())
```

```
print("range :", x.range())

print("variance :", x.variance())
print("standard_deviation :", x.std())

# in Stdout 1:
mean : 88.0
median : 78
mode : [78]
max : 140
min : 65
range : 75
variance : 635.0
standard_deviation : 25.199206336708304
```

```
# Test 2 - 예외처리 Test
weight = [1]

x = Data(weight)

print("mean :", x.mean())
print("median :", x.median())
print("mode :", x.mode())

print("max :", x.max())
print("min :", x.min())
print("range :", x.range())

print("variance :", x.variance())
print("standard_deviation :", x.std())

# in Stdout 2:
mean : 1.0
median : 1
mode : [1]
max : 1
min : 1
range : 0
Variance를 구하는 과정에서 self.size가 0이 되는 문제가 발생했습니다.
variance : -1
Variance를 구하는 과정에서 self.size가 0이 되는 문제가 발생했습니다.
standard_deviation : -1
```

Ex 03

Solution

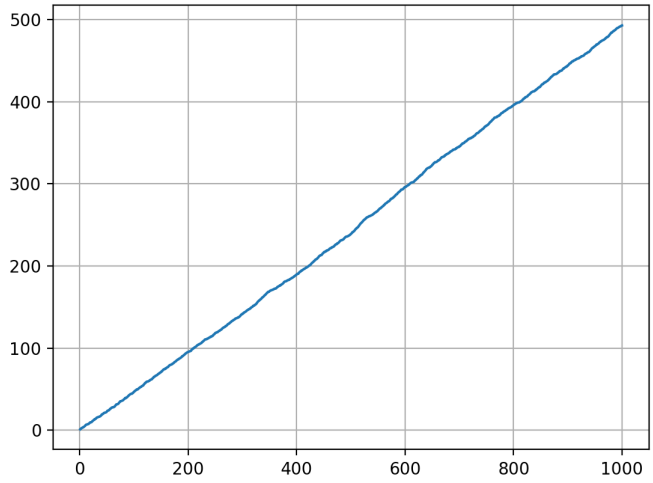
```
import matplotlib.pyplot as plt
import random
```

```
sample_size = 1000

domain = [i + 1 for i in range(sample_size)]
sample = [random.uniform(0, 1) for _ in range(sample_size)]

cumul_sum = 0
cumul_sample = []
for i in sample:
    cumul_sample.append(cumul_sum + i)
    cumul_sum += i

plt.plot(domain, cumul_sample)
plt.grid()
plt.show()
```

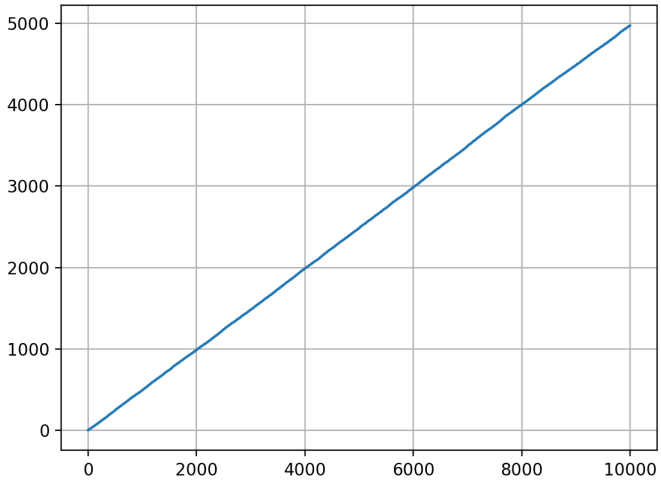
- n을 담을 변수 `sample_size`를 선언한다.
- `sample_size` 만큼의 domain을 생성하고, 이에 해당하는 sample을 `random.uniform(0, 1)`을 이용해 추출한다.
  - 누적합을 담을 변수 cumul_sum과 리스트 cumul_sample을 생성한다.
- 반복을 진행:
  - `cumul_sample`에 지금까지의 누적합 `cumul_sum + i`을 append
  - `cumul_sum`에 원소 i를 더해서 누적합을 갱신
- `plt.plot()`을 이용해 domain과 cumul_sample 사이의 그래프를 생성한다.
- `plt.grid()`를 통해 격자를 생성한다.
- `plt.show()`로 생성한 그래프를 확인한다.

Result

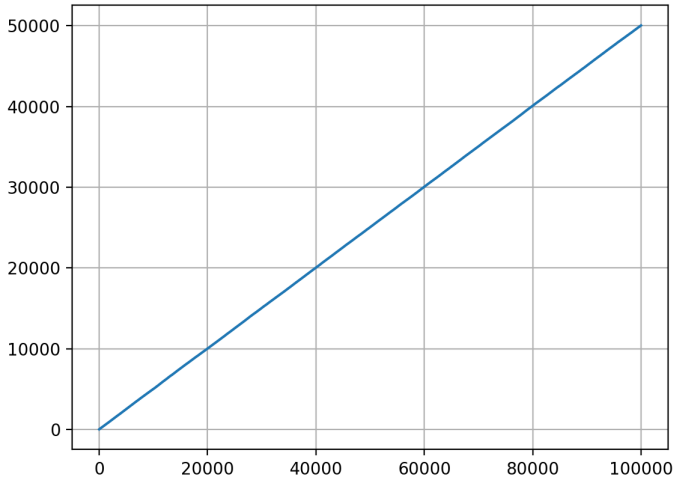
sample_size = 1000



sample_size = 10000



sample_size = 100000



Conclusion

- n의 사이즈가 클 수록 그래프의 경향이 $y = \frac{1}{2}x$를 따른다. 따라서 이는 $\frac{n}{2}$에 수렴한다.

Ex 04

Solution

```
from random import randint

def gen_matrix():
    mat_1 = [[randint(0, 100) for _ in range(3)] for _ in range(3)]
    mat_2 = [[randint(0, 100) for _ in range(3)] for _ in range(3)]

    mat_mul = [[mat_1[i][j]*mat_2[i][j] for j in range(3)] for i in range(3)]

    print("matrix 1 :", mat_1)
    print("matrix 2 :", mat_2)
    print("multiply matrix elementwise :", mat_mul)
```

- 이중 list comprehension 방법을 이용해 0~100 사이의 3*3 행렬 `mat_1`, `mat_2`를 생성한다.
- 이중 list comprehension 방법을 이용해 인덱스 i, j를 0, 1, 2에 대해 반복을 진행하면서 동일한 인덱스의 원소를 곱한 값을 3*3형태로 담는 행렬 `mat_mul`을 생성한다.
- 생성한 행렬들을 출력한다.

Result

```
# Test 1
gen_matrix()

# in Stdout 1:
matrix 1 : [[52, 58, 16], [30, 45, 0], [1, 48, 36]]
matrix 2 : [[69, 84, 80], [50, 45, 66], [86, 54, 53]]
multiply matrix elementwise : [[3588, 4872, 1280],
                               [1500, 2025, 0],
                               [86, 2592, 1908]]
```

```
# Test 2 - with tuple
gen_matrix()

# in Stdout 2:
matrix 1 : [[90, 71, 85], [91, 28, 97], [82, 82, 24]]
matrix 2 : [[69, 79, 38], [98, 53, 3], [71, 7, 82]]
multiply matrix elementwise : [[6210, 5609, 3230],
                               [8918, 1484, 291],
                               [5822, 574, 1968]]
```