확률통계론_HW03

2021045796 김도겸

Conception of Program 1

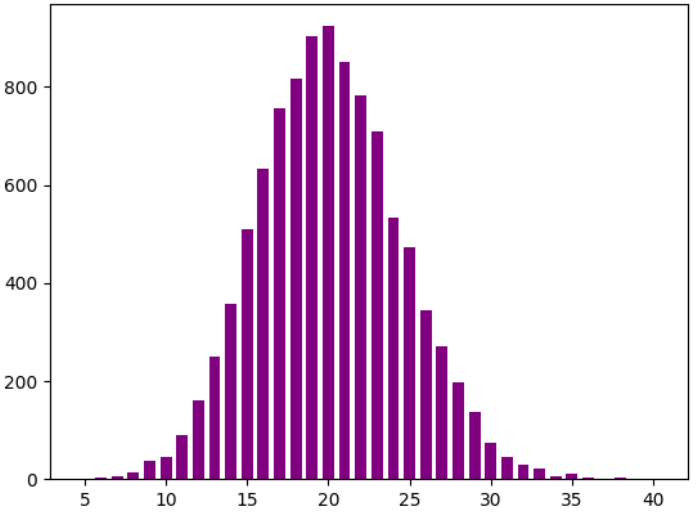
- 랜덤 넘버 생성기로 생성한 1000개의 랜덤한 수를 1000 size 배열에 각각 저장
- For iteration을 이용하여 10000회 반복, 1000*10000 size 배열에 값을 저장. Lambda값과 time interval 값 계산
- Poisson Distribution를 Histogram 형식으로 작성

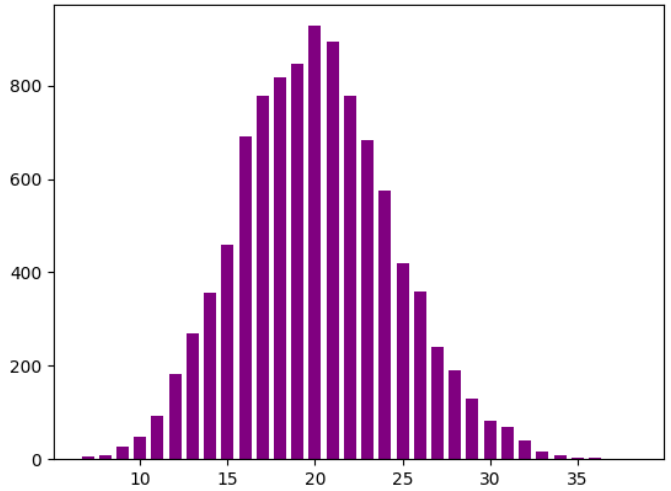
Code Description 1

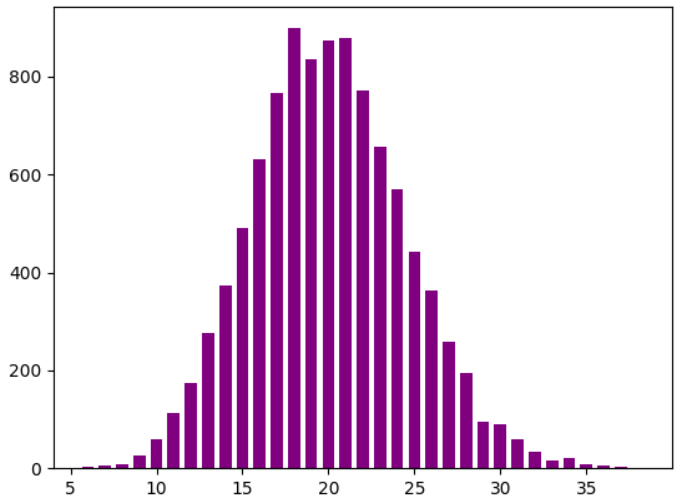
```python
import random
import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson

ffile = open("C:/Users/admin/PoissonDistribution.txt", 'w')

#np.random.seed(seed=10)
array = [[0 for col in range(1000)] for row in range(10000)]

arrForPoisson = []
calcul = 0
timeInterval = 0
sumOfCalcul = np.zeros(10000)
checkTime = 0


for restart in range(10000):
    for repeat in range(1000):
        array[repeat] = np.random.randint(1, 50)
        timeInterval = timeInterval + 1
        if array[repeat] == 1:
            calcul = calcul + 1
            arrForPoisson.insert(checkTime, timeInterval)
            #[checkTime] = timeInterval
            timeInterval = 0
            checkTime = checkTime + 1
    sumOfCalcul[restart] = calcul
    calcul = 0
```

- Python을 사용
- numpy와 mathplotlib를 중심으로 구현
- calcul = Trial 1당 Event 발생 횟수
- timeInterval = Time Interval
- sumOfCalcul = 10000개의 Event 발생 횟수
- checkTime = 총 Time Interval 개수(몇 번째 배열에 insert할지 판단하는데 사용

Execution 1~3







2 37 9 15 19 40 19 22 32 30 1 31 26 41 34 14 15 13 18 14
28 36 25 9 20 10 33 9 45 13 45 21 31 26 36 22 48 33 20 32
7 20 37 3 35 15 43 6 9 34 46 37 25 32 9 32 43 47 30 30
1 3 11 11 25 3 39 22 1 31 17 5 37 40 30 23 29 41 46 11
42 17 13 32 30 24 34 32 20 18 28 12 32 2 4 45 35 10 28 28
30 5 2 24 19 12 13 12 46 30 21 9 40 11 6 17 15 18 45 30
12 15 37 39 42 32 20 26 31 48 16 38 7 1 12 23 5 11 44 5
3 6 31 30 48 30 49 45 22 1 11 36 37 6 44 3 36 45 41 7
30 13 42 7 25 30 3 47 14 18 27 40 12 8 22 6 17 16 49 47
31 48 44 22 41 8 45 48 17 41 28 46 18 19 3 20 13 36 10 5
3 29 6 33 26 32 36 23 31 23 34 47 42 45 39 36 20 9 9 22
12 7 35 28 39 21 29 22 10 38 24 25 7 35 10 21 5 6 47 24
33 42 46 16 22 1 29 17 37 3 8 43 48 44 41 49 37 11 28 29
38 44 20 46 2 39 4 30 5 46 32 25 16 38 20 46 4 2 47 10
28 17 44 45 39 29 43 42 28 39 28 17 44 30 2 31 49 2 11 25
26 31 43 12 12 29 17 14 35 28 49 48 29 49 24 45 39 14 14 19
3 25 11 25 14 15 11 47 37 16 9 25 38 32 25 48 5 31 37 25
48 22 46 10 40 17 35 3 7 5 21 36 13 47 45 39 32 29 49 23
9 46 11 14 20 27 4 29 8 1 11 28 13 21 12 1 16 20 7 49
18 17 48 40 23 11 20 43 24 17 19 3 13 24 23 37 3 6 15 9
1 47 10 41 6 13 5 10 19 13 19 7 29 22 19 25 33 32 32 23
12 34 39 48 14 28 38 24 31 21 49 13 7 36 47 48 19 29 9 34
6 40 36 34 18 31 13 4 30 3 36 13 20 32 39 6 31 6 44 17
17 33 10 15 13 8 44 6 6 21 27 47 41 41 48 5 15 16 23 25
40 27 45 35 10 25 25 2 26 4 45 3 9 45 10 11 22 3 1 3
18 7 48 22 46 11 11 11 38 48 24 46 48 25 3 1 30 37 42 25
23 26 9 4 10 27 15 15 3 13 9 6 32 1 11 11 31 13 33 16
13 6 6 32 12 4 41 38 43 14 12 14 31 2 30 38 45 11 24 44

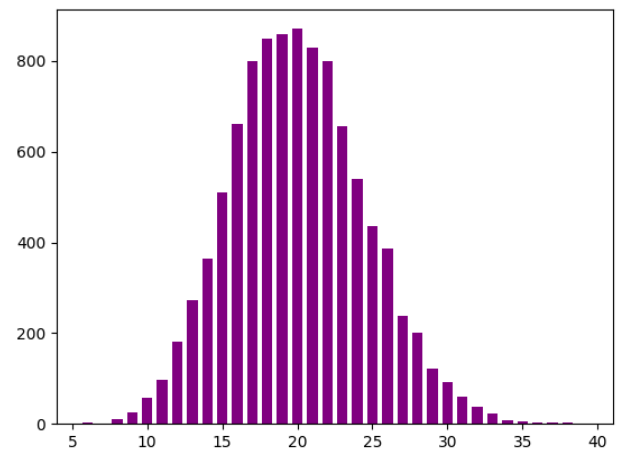
38 30 14 6 18 19 32 19 18 27 17 21 8 29 49 40 41 25 20 46
15 27 9 8 41 12 7 7 20 35 35 21 40 44 48 38 32 29 42 4
48 18 48 48 23 23 37 43 19 42 13 48 3 43 31 22 15 33 25 6
34 48 7 47 6 22 19 45 23 16 4 49 7 36 22 49 23 36 16 31
26 7 27 1 12 47 37 30 19 31 7 20 39 34 28 1 13 15 18 32
3 15 20 49 9 33 17 4 20 22 12 47 8 23 24 24 49 38 13 6
12 43 29 25 23 21 29 16 28 49 7 10 23 14 16 28 41 49 15 29
17 4 14 11 47 33 13 1 36 18 47 5 37 39 14 2 28 2 34 9
10 22 40 11 28 43 46 16 35 46 14 27 10 25 18 36 14 49 20 40
24 7 48 31 43 34 46 21 31 45 16 24 22 28 6 19 20 38 8 41
34 4 49 19 22 35 40 31 4 30 45 16 39 22 30 4 24 3 38 44
38 30 18 42 26 1 7 42 42 20 4 26 37 44 1 46 8 34 22 20
27 21 32 8 8 18 4 15 6 41 41 42 18 27 16 35 21 46 11 24
15 41 1 44 23 48 28 20 48 8 10 41 19 26 27 9 3 46 38 19
8 35 25 10 7 1 25 1 16 49 12 8 28 16 4 35 12 24 19 32
23 42 19 6 31 49 17 22 7 9 15 25 33 49 49 46 7 30 10 23
44 4 7 14 15 44 29 2 9 5 9 39 25 30 21 16 40 6 42 32
30 40 28 16 41 34 32 34 8 14 38 47 10 12 16 16 42 6 38 41
14 34 35 2 12 47 34 19 49 23 26 49 33 33 32 27 24 44 6 30
16 6 23 27 39 2 45 43 49 48 2 39 4 1 6 28 8 7 30 35
13 20 39 40 41 9 29 39 35 38 16 15 33 22 16 38 20 29 41 42
38 29 30 29 20 10 37 48 46 15 4 12 19 14 33 26 43 47 22 27

21 42 8 19 29 22 42 42 24 34 17 43 1 37 7 22 2 12 40 29 42 11 38 42 7 26 2 36 25 33 45 12 24 36 30
36 41 32 14 26 43 28 32 48 44 40 31 33 49 20 11 21 45 27 37 37 22 32 7 34 17 15 47 42 8 7 21 15 17 11
16 49 30 27 34 47 19 40 41 41 44 28 28 32 17 40 27 11 16 1 4 22 17 6 25 24 14 47 38 33 9 21 30 9 28
2 13 30 16 28 12 17 33 14 28 47 33 35 1 8 41 45 23 7 43 34 11 28 28 27 40 17 29 39 9 30 14 37 2 4
18 6 14 3 38 8 9 48 37 8 48 11 41 24 39 37 30 14 11 6 3 1 16 16 36 46 34 28 11 21 30 27 35 4 15
22 33 23 20 30 23 44 8 48 15 8 41 47 23 2 34 19 25 15 12 26 9 2 4 32 2 20 49 16 49 16 38 35 46 25
33 49 5 18 24 4 10 47 23 9 19 37 48 35 35 33 34 13 29 30 31 42 31 15 19 19 4 14 7 42 3 25 20 31 46
40 27 26 21 32 35 3 29 13 26 11 25 31 1 39 27 19 20 48 47 18 8 42 29 13 10 27 21 33 18 32 43 12 44 43
14 9 48 36 26 8 49 26 33 49 30 26 12 7 26 37 12 47 37 21 3 21 47 40 5 2 12 32 39 34 29 46 10 22 16
29 26 45 31 49 13 30 16 44 26 45 40 4 29 31 4 49 48 48 43 7 42 25 2 29 25 12 42 13 12 31 36 34 34 16
2 12 48 45 12 39 4 6 49 17 15 6 21 49 25 40 8 10 15 36 40 34 29 14 8 43 23 28 49 31 25 27 16 49 25
29 22 43 26 22 7 2 31 12 31 6 29 17 48 1 17 40 7 29 45 11 18 27 48 17 24 34 42 40 18 40 49 4 37 3
11 1 13 26 29 33 42 35 15 2 39 31 20 45 25 27 37 47 13 21 20 29 10 43 46 27 29 43 20 2 10 27 29 36 41
39 24 26 47 27 42 41 8 2 25 5 16 21 26 11 36 35 24 6 29 25 2 2 5 4 11 16 38 33 9 46 45 33 20 30
26 42 6 21 42 40 46 23 26 44 23 11 18 2 35 10 46 41 24 5 4 34 8 14 12 4 30 8 7 37 19 17 1 42 15
14 49 17 7 42 48 3 12 46 30 27 40 37 43 20 25 8 46 44 26 49 33 33 45 40 7 26 19 9 30 5 25 45 12 29
35 2 4 45 29 33 38 49 13 4 49 21 2 46 14 14 3 31 46 22 35 12 43 19 4 49 22 36 39 47 11 5 44 7 47
46 46 31 21 21 44 27 17 36 13 11 3 2 7 30 37 38 46 28 26 5 31 13 13 26 34 38 1 36 36 14 16 12 27 4
28 47 6 36 19 19 37 4 23 33 33 17 21 2 36 30 13 28 16 21 18 29 25 24 44 24 15 6 12 8 41 33 49 12 48
24 49 47 34 47 41 16 31 21 11 23 38 21 32 14 15 16 46 33 27 16 25 29 29 41 1 2 8 43 46 33 6 39 18 21
28 26 14 8 8 31 2 3 26 39 37 42 27 23 6 28 31 38 31 5 10 8 45 37 10 23 10 5 47 2 5 8 17 48 5
34 43 26 30 31 25 21 5 14 27 3 10 36 48 16 10 33 34 45 19 5 19 29 14 49 26 33 15 48 32 45 45 29 49 29
9 1 25 2 44 25 4 37 22 27 45 17 39 10 28 23 1 26 6 15 46 5 31 27 23 12 39 22 9 45 30 26 19 41 27
47 38 42 27 22 15 30 18 9 23 21 7 8 41 37 34 22 9 20 28 27 16 27 10 2 46 49 41 19 20 6 1 10 33 13
11 47 21 24 49 45 1 37 25 49 34 23 10 16 20 4 35 42 9 12 27 20 38 13 47 39 45 12 32 40 2 44 26 7 39
46 32 33 40 3 34 7 22 47 22 22 41 33 7 46 40 29 21 43 9 17 2 20 3 39 42 6 16 25 13 10 17 10 22 30
26 20 8 5 38 4 3 7 36 9 6 16 9 5 12 10 36 25 24 11 28 44 13 35 31 29 49 5 27 6 13 44 2 21 10
46 24 40 39 18 25 9 5 43 41 6 20 39 3 5 23 35 7 4 22 3 15 1 23 26 15 45 8 25 14 32 15 10 33 42
6 32 21 14 31 4 18 16 5 44 24 22 27 31 36 23 14 10 24 27

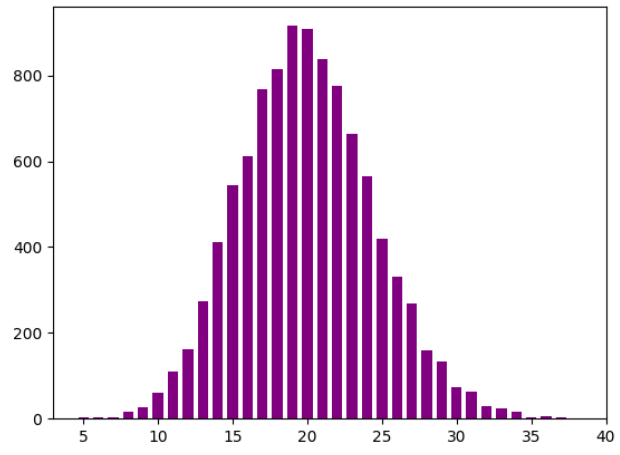
41 42 30 12 48 35 35 27 24 21 10 10 36 33 22 18 17 30 45 13 39 10 36 21 21 45 26 1 44 17 17 12 31 37 7
7 16 12 17 21 43 21 33 16 43 7 22 49 9 21 36 41 25 29 32 22 8 22 38 20 19 4 36 7 29 34 21 43 18 31
9 21 17 32 11 8 30 24 11 19 10 3 40 7 18 12 43 44 35 37 33 26 38 15 35 39 1 16 24 48 11 19 5 37 17
32 6 2 2 3 44 28 44 23 35 7 19 47 15 47 41 40 28 38 12 28 3 29 2 16 2 37 6 34 46 11 15 43 9 10
11 43 35 35 28 27 11 20 10 13 40 15 35 41 7 48 15 37 44 23 24 10 34 9 13 4 21 11 4 49 5 31 23 13 41
3 14 21 3 14 25 10 39 32 4 40 18 32 13 9 19 37 30 44 1 34 10 20 27 36 8 5 46 38 6 13 7 28 18 37
9 32 2 40 34 2 18 35 32 5 10 45 46 49 30 8 4 32 9 23 24 6 38 19 37 41 7 38 7 45 42 5 39 18 24
11 46 43 35 32 2 47 3 26 49 34 33 24 16 48 20 16 2 26 14 5 3 47 12 19 44 48 49 45 5 43 16 44 20 40
27 12 16 23 34 37 5 6 10 20 47 32 25 9 24 42 20 28 30 18 36 38 25 2 15 33 44 19 40 49 27 36 38 31 21
15 27 45 35 13 17 18 13 44 20 44 18 43 27 11 16 46 28 1 21 22 26 41 9 44 28 19 20 32 48 32 33 26 37 29
19 4 36 29 20 6 39 8 45 48 11 40 34 32 29 7 45 42 45 48 41 32 8 40 33 22 32 30 9 36 1 29 46 47 48
36 8 27 5 25 44 39 40 2 49 30 13 39 27 44 4 38 19 2 33 7 34 5 48 41 10 16 36 42 31 16 8 40 26 48
18 45 24 1 36 20 29 31 8 19 14 26 38 38 26 37 21 42 38 9 49 15 11 35 48 13 19 45 4 34 20 15 20 33 47
42 46 45 30 40 38 24 39 19 20 29 26 9 45 29 44 38 6 7 33 20 9 6 16 1 34 18 46 8 32 15 39 22 26 47
6 45 33 21 13 36 40 15 25 27 11 2 36 41 26 20 23 31 16 15 9 31 46 33 6 35 27 1 22 37 10 7 15 41 9
43 13 20 33 38 45 4 36 6 6 39 47 39 34 5 49 26 12 45 35 47 48 8 30 44 16 30 20 2 44 5 33 36 47 12
40 48 28 42 26 46 9 18 13 27 18 10 4 7 8 27 11 17 45 5 44 32 44 7 33 23 7 5 17 6 48 3 19 2 12
5 19 32 17 49 1 46 25 24 3 19 46 19 49 20 11 27 18 25 20 2 34 15 4 47 29 10 18 26 7 4 43 8 47 29
29 49 24 17 14 24 49 44 30 3 19 42 26 31 30 47 45 43 10 34 7 10 16 15 18 34 29 7 41 24 34 35 43 13 22
3 47 39 19 49 40 14 26 5 25 35 30 15 36 6 3 4 5 26 26 41 48 3 36 16 44 43 39 39 24 9 23 8 37 12
13 24 5 35 33 13 2 23 37 21 38 26 44 20 44 11 12 13 4 39 35 1 49 8 3 47 17 41 1 11 14 41 43 19 40
34 6 2 46 38 35 12 5 44 17 5 33 39 44 5 30 26 16 27 37 12 13 28 7 24 25 9 46 15 35 45 32 44 42 38
18 33 34 37 40 9 18 15 26 20 46 17 40 29 33 24 20 3 6 46 36 39 14 31 28 24 10 17 49 24 33 40 49 9 32
39 11 34 16 24 27 45 21 20 35 33 35 24 21 12 6 26 16 13 16 18 43 12 46 19 35 46 40 3 7 36 2 29 39 13
7 47 34 16 42 10 33 48 24 24 25 3 20 40 8 24 22 33 48 49 28 35 35 46 40 13 6 2 16 39 7 26 34 1 12
32 7 19 7 17 32 22 42 33 42 16 8 17 21 41 43 23 19 15 44 10 3 6 13 8 30 36 3 23 18 11 45 40 40 39
44 37 2 10 9 23 21 17 1 13 32 44 40 48 47 46 48 24 39 17 26 43 21 15 33 11 22 40 1 10 44 23 4 1 28
17 8 11 38 33 5 47 34 16 41 32 47 19 32 24 37 28 26 36 41 48 28 24 8 17 7 42 36 8 45 29 40 47 36
22 38 22 11 32 39 46 48 10 1 41 [illegible]

One example of random generation

Execution 4~5




```
[ 6  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 31 32 33 34 35 36 37 38 39]
[  2  11  26  57  98 182 272 363 511 660 799 848 859 871 829 799 655 539
 435 387 238 201 121  93  61  37  22   8   6   4   3   2   1]
```




```
[ 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
 29 30 31 32 33 34 35 36 37 38]
[  2   2   2  15  27  60 109 161 275 411 544 613 769 814 916 910 838 775
 663 566 419 330 269 159 134  74  62  28  25  15   4   6   2   1]
```

Unique and Counts of Poisson Distribution

Code Description 2

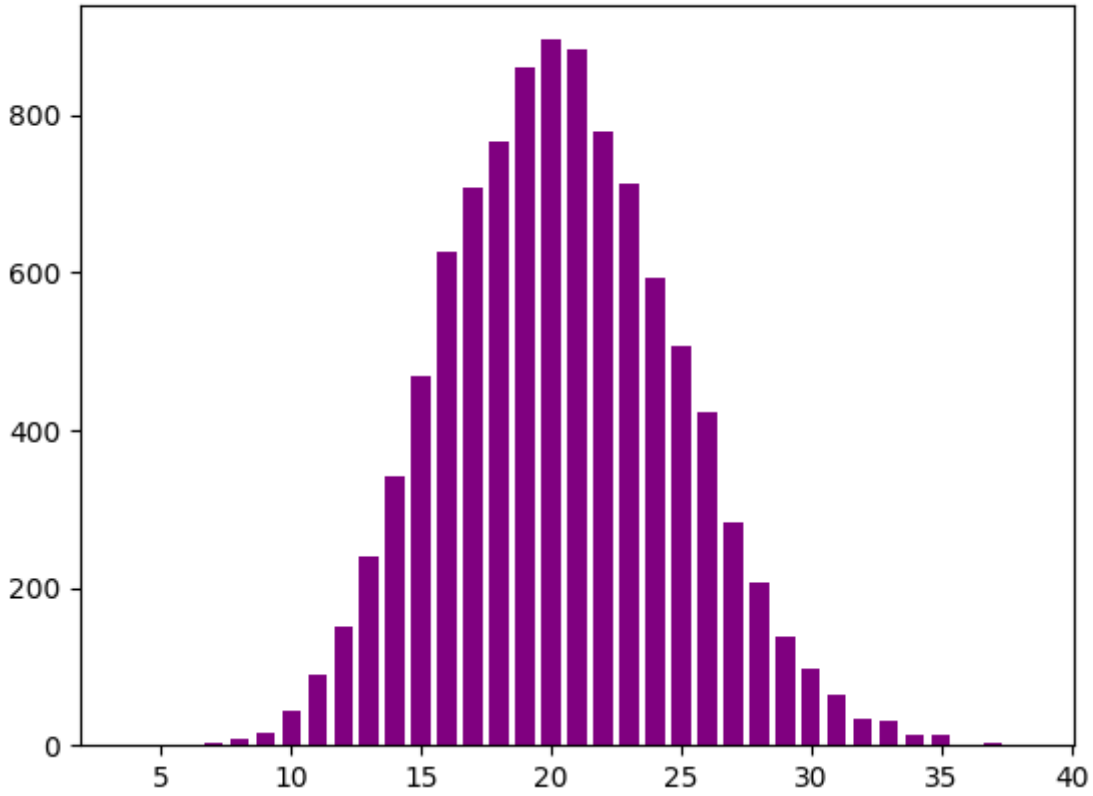
```
unique, counts = np.unique(sumOfCalcul, return_counts=True)

np.asarray((unique, counts)).T

#print(unique)
#print(counts)

plt.bar(unique, counts, width=0.7, color="purple", align='center')
plt.show()
```

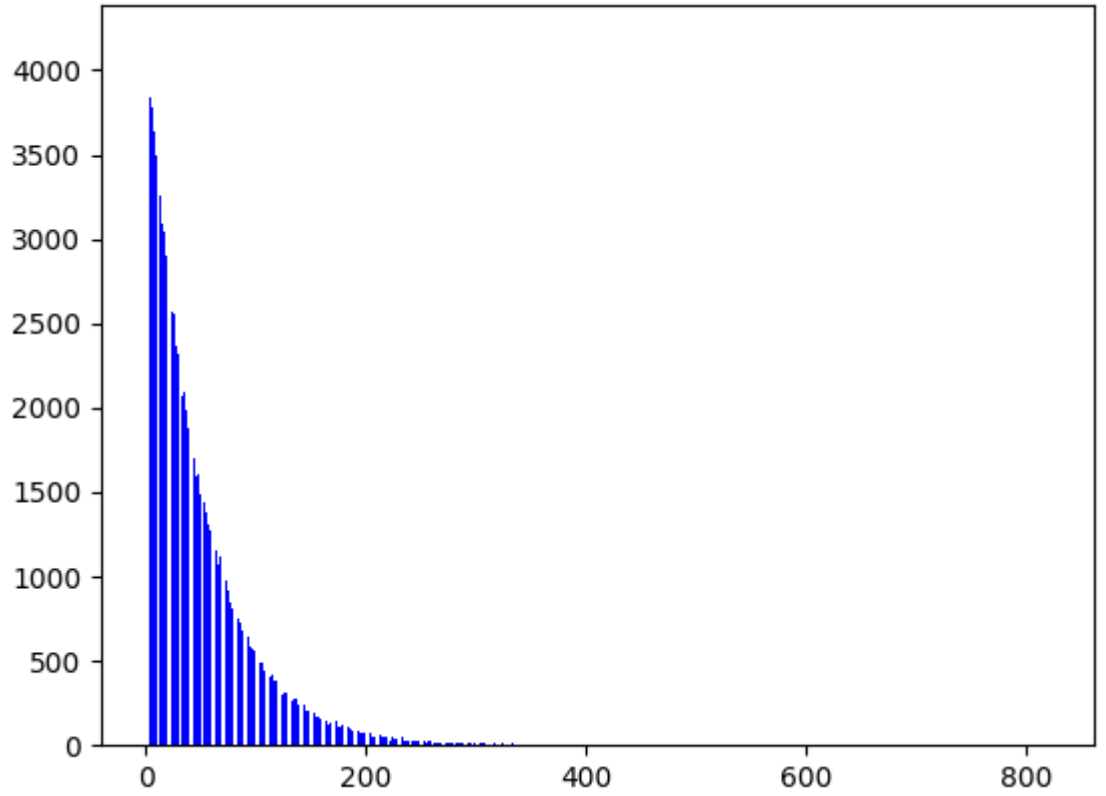
- Numpy의 unique 함수를 이용
- Event 발생 횟수를 담고있는 sumOfCalcul 배열을 unique와 counts로 분해.
- unique를 x축, counts를 y축으로 하는 Histogram 작성
- arrForPoisson 배열에는 Time Interval을 삽입

Conception of Program 2

- Time Interval은 굳이 이중 배열로 만들 필요가 없다는 것을 알게 되었음
- 빈 배열을 생성하여 Time Interval을 따로 저장
- Time Interval을 Histogram, Line Graph로 작성

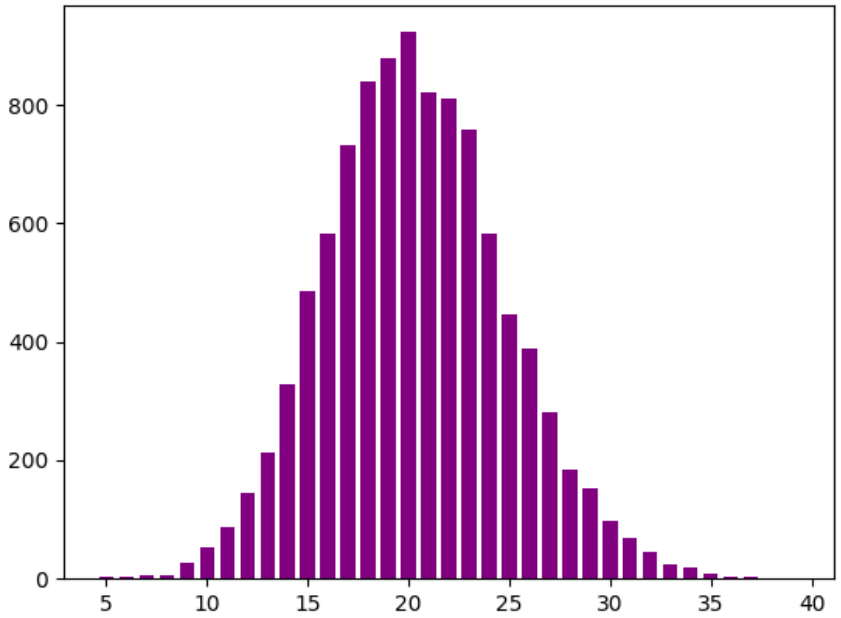
Execution 6

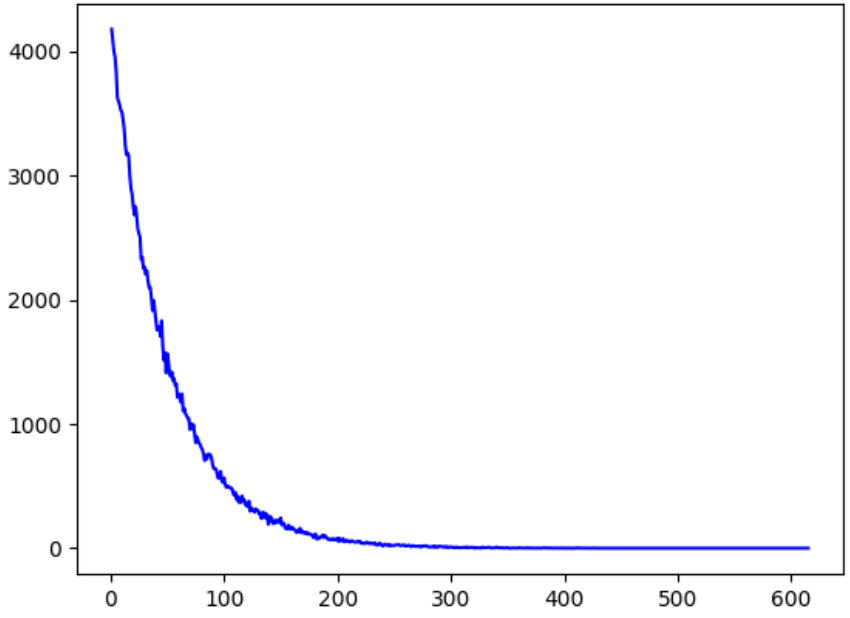




Version that made with Histogram

Execution 7

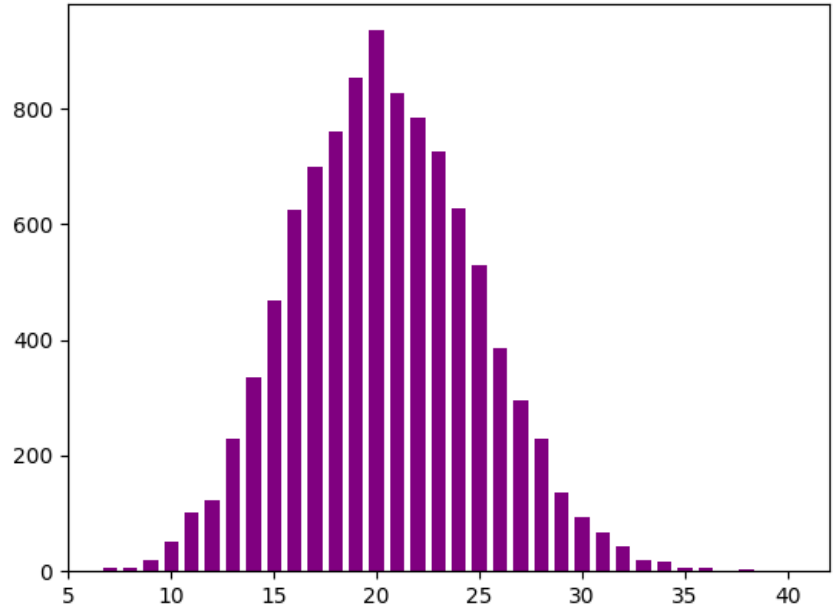


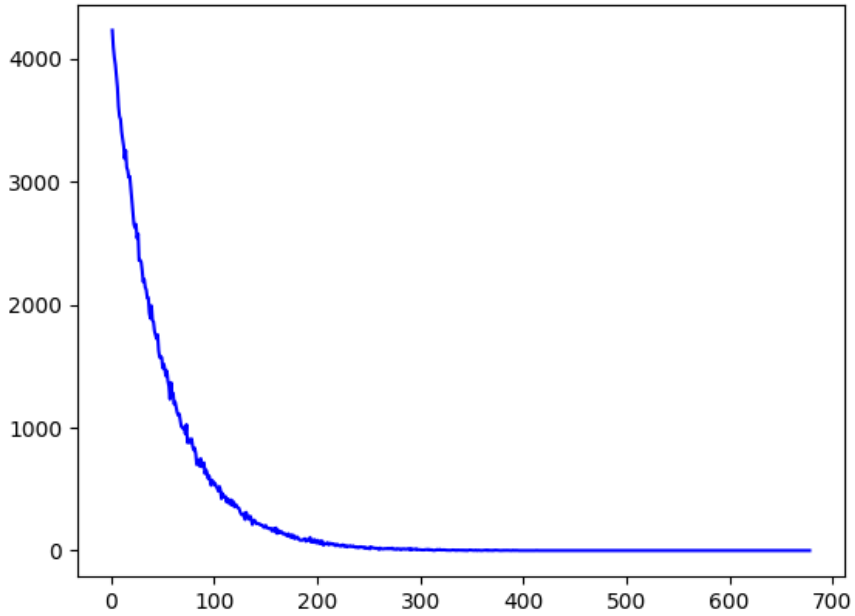


4176 4085 3994 3951 3829 3628 3595 3568 3525 3509 3451 3377 3241 3167 3180 3158 2991 2894 2847 2751 2683 2748 2675 2569 2531 2505 2325 2338 2250 2263 2204 2234 2149 2098 2098
2013 1913 1994 1918 1828 1756 1782 1774 1705 1830 1630 1513 1573 1410 1566 1502 1406 1381 1419 1348 1366 1310 1323 1214 1226 1240 1176 1242 1104 1126 1084 1067 1047 1044 956
1003 961 995 933 849 896 873 846 833 809 792 755 706 748 718 758 757 751 728 669 641 642 635 594 563 583 619 543 531 567 518 490 502 498 487
486 482 444 426 450 396 421 370 406 368 422 383 387 348 355 336 379 298 327 308 308 290 318 304 304 277 281 239 291 250 285 241 272 192 224
257 227 202 199 229 207 213 231 207 245 187 195 199 160 163 155 187 164 152 170 161 152 142 126 140 136 162 125 140 124 123 129 115 126 113
112 112 100 85 113 113 73 85 79 94 92 106 87 102 90 79 71 74 69 75 70 76 69 69 81 54 80 62 61 78 53 55 65 57 59
49 60 59 55 50 49 50 58 50 56 40 42 43 36 43 50 42 36 45 38 38 36 45 31 25 31 43 36 24 19 26 34 26 25 21
31 21 21 23 23 26 27 29 30 24 22 23 27 27 17 19 23 25 20 12 22 21 15 17 17 15 19 11 12 21 16 17 22 17 10
12 10 13 15 9 18 11 7 15 17 15 15 11 8 14 14 4 11 11 8 11 8 4 10 7 9 4 5 7 6 8 8 4 12 4
5 9 7 5 9 1 5 6 3 4 5 1 9 6 4 6 7 9 3 2 4 3 3 10 6 6 2 5 4 5 2 3 5 6 1
1 1 3 3 3 1 4 5 3 2 3 2 3 2 3 5 1 2 3 2 2 1 2 3 2 2 4 4 1 3 3 1 2 2 2
2 2 1 1 1 1 2 1 3 1 4 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 2 1 1 1 1 1 1 1

Counts of Time Interval

Execution 8





4233 4096 4010 3953 3853 3768 3614 3532 3508 3410 3350 3292 3189 3255 3116 3093 3037 3043 2948 2855 2743 2656 2627 2655 2542 2577 2361 2364 2348 2279 2187 2206 2142 2121 2055
2057 1934 1891 1992 1874 1862 1784 1755 1724 1757 1626 1572 1581 1570 1485 1518 1477 1421 1461 1380 1337 1232 1368 1265 1281 1193 1209 1164 1124 1101 1112 1059 1009 1008 990
995 949 1026 879 911 879 890 910 850 819 835 788 700 721 741 686 748 682 704 715 626 656 634 587 608 570 531 572 561 541 544 518 508 483 522
493 423 465 441 433 440 394 423 397 380 414 370 404 363 356 376 357 350 346 317 300 289 295 303 253 312 284 275 254 280 254 212 244 248 236
219 221 219 218 207 208 205 190 190 205 186 184 179 181 182 160 168 181 153 185 140 167 145 152 156 146 135 137 137 125 118 137 117 110 129
106 125 97 100 112 110 92 86 86 84 88 94 98 76 82 91 87 108 69 78 68 91 70 63 85 64 58 80 58 77 44 56 55 60 66
47 52 61 52 55 56 42 40 52 49 40 39 37 46 47 34 39 41 30 36 37 40 33 34 38 33 27 32 26 24 32 28 32 27 28
20 27 19 19 24 14 32 25 24 21 19 20 19 15 9 22 16 14 14 17 15 17 19 14 18 19 14 21 13 11 14 10 16 14 14
7 12 15 12 15 11 12 16 15 5 17 8 10 12 11 15 7 9 8 6 6 6 6 6 9 9 8 11 5 6 8 4 2 4 4
9 7 6 10 5 6 7 5 7 4 4 4 3 3 3 6 6 5 2 5 2 5 4 2 3 5 2 4 1 4 1 3 6 5 4
2 2 3 5 1 3 4 1 2 3 4 5 5 1 5 2 3 1 4 2 5 2 1 2 1 3 2 3 2 2 2 1 4 1 3
1 1 1 1 1 2 2 2 1 1 1 3 1 1 1 1 2 2 1 1 2 1 1 1 2 1 1 2 1 1 1 1 1 1 1
1 1 1 1

Code Description 3

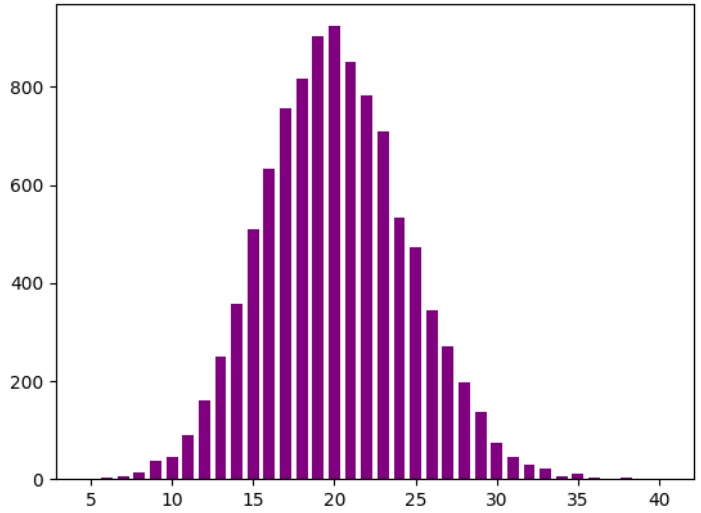
```
unique2, counts2 = np.unique(arrForPoisson, return_counts=True)

np.asarray((unique2, counts2)).T

#print(unique2)
#print(counts2)

numToFind = 0

for printRepeat in range(len(counts2)):
    print(counts2[printRepeat])
    printarray = str(counts2[printRepeat])
    ffile.write(printarray)
    ffile.write(" ")
    numToFind = numToFind + 1
    if numToFind == 35 :
        ffile.write("\n")
        numToFind = 0

ffile.close
plt.plot(unique2, counts2, color="blue")
plt.show()
```

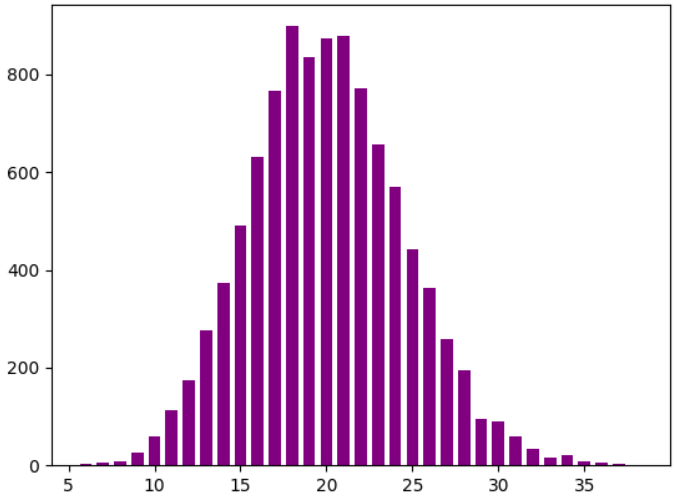
- Numpy의 unique 함수를 이용
- Time Interval을 갖고 있는 arrForPoisson 배열을 unique와 counts 로 분해.
- unique를 x축, counts를 y축으로 하는 Line Graph 작성

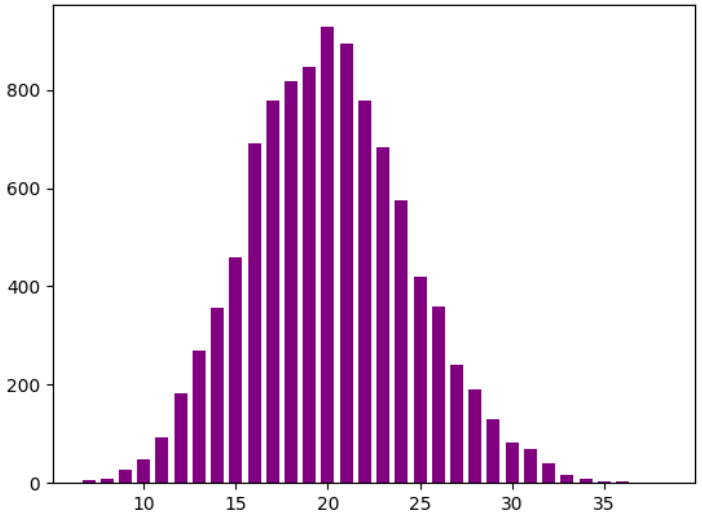
Explanation

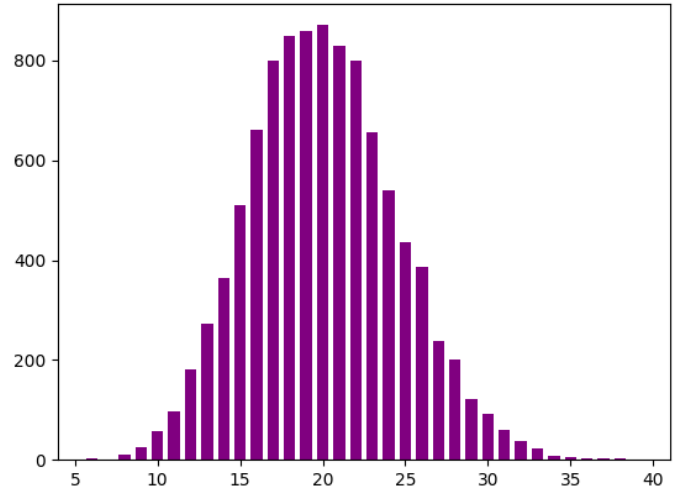
- Random Generator을 이용하여 1~50까지의 변수 생성
- 1이 나오는 경우를 체크
- Repeat It for 10000 Experiment
- $\lambda = 50$
- 기댓값 = 20
- $f(t) = \lambda e^{-\lambda t}$
- Express Poisson Distribution with PDF

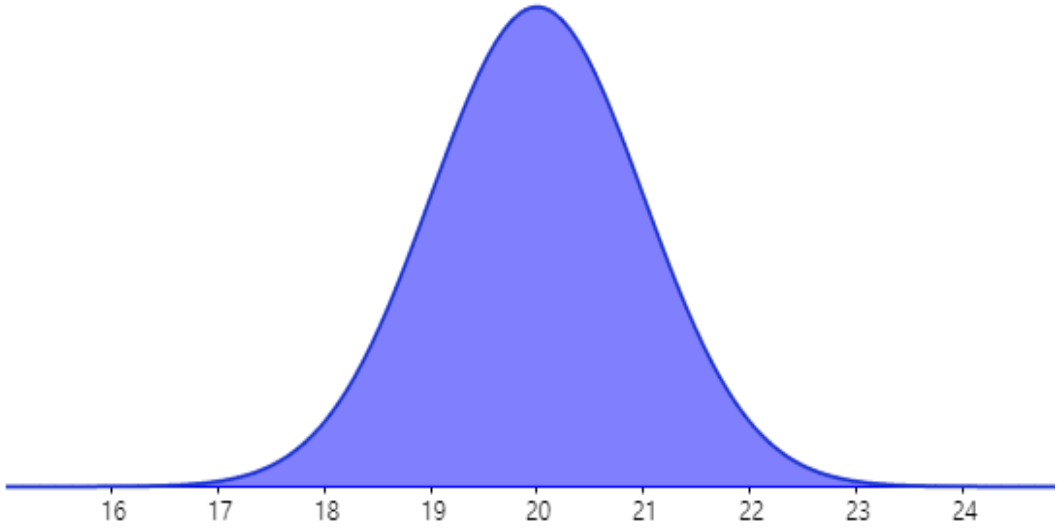
Compare Histogram1





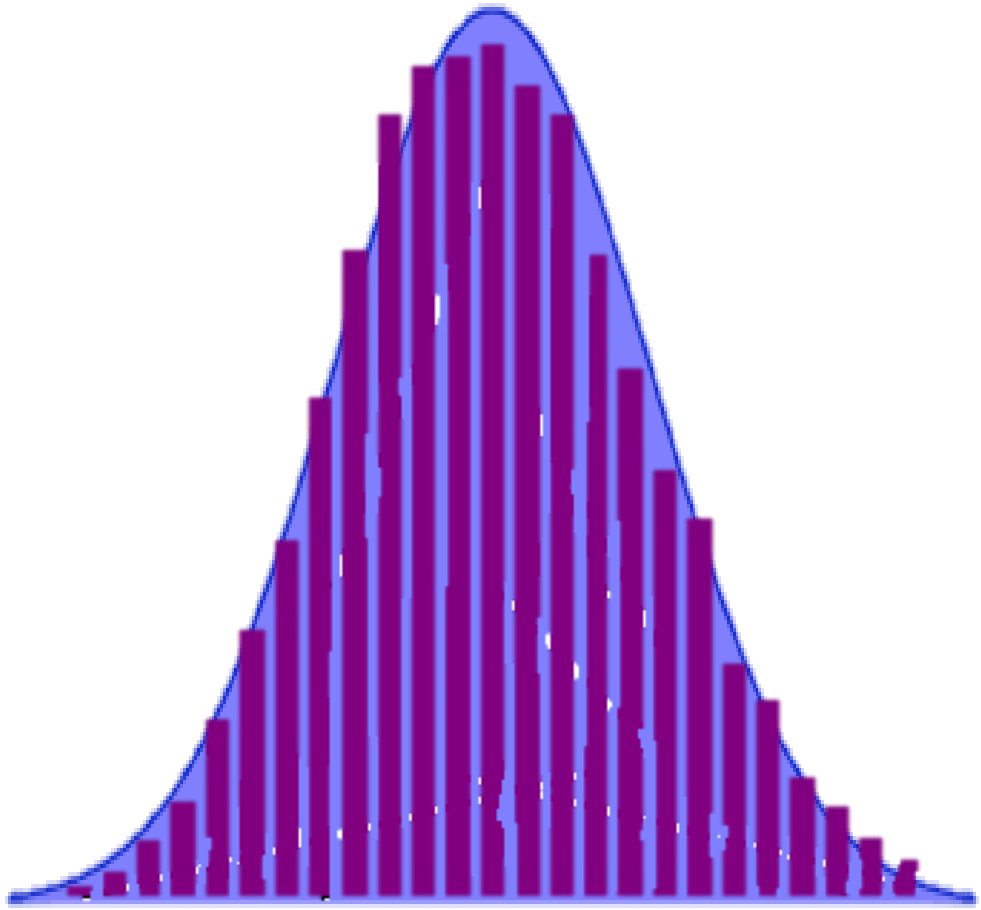






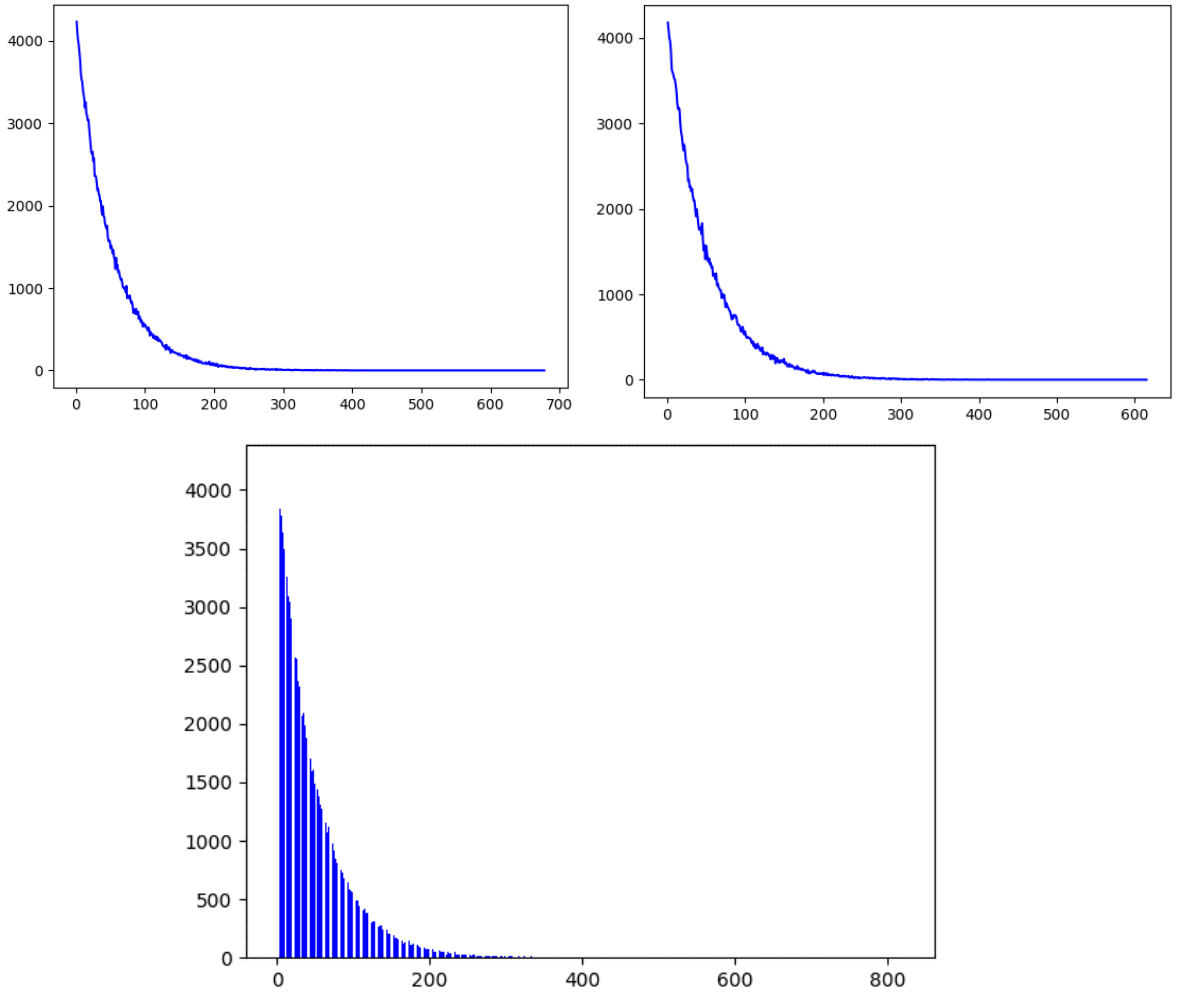
Source: GeoGebra

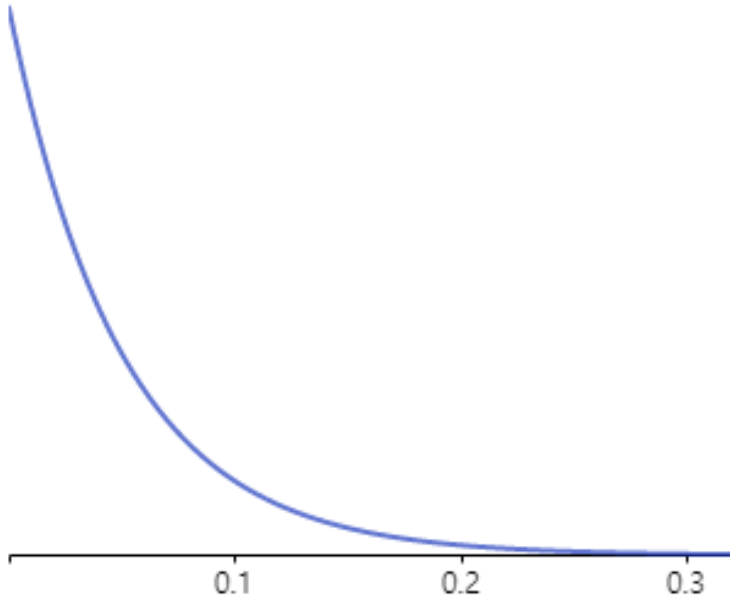
Compare Histogram2



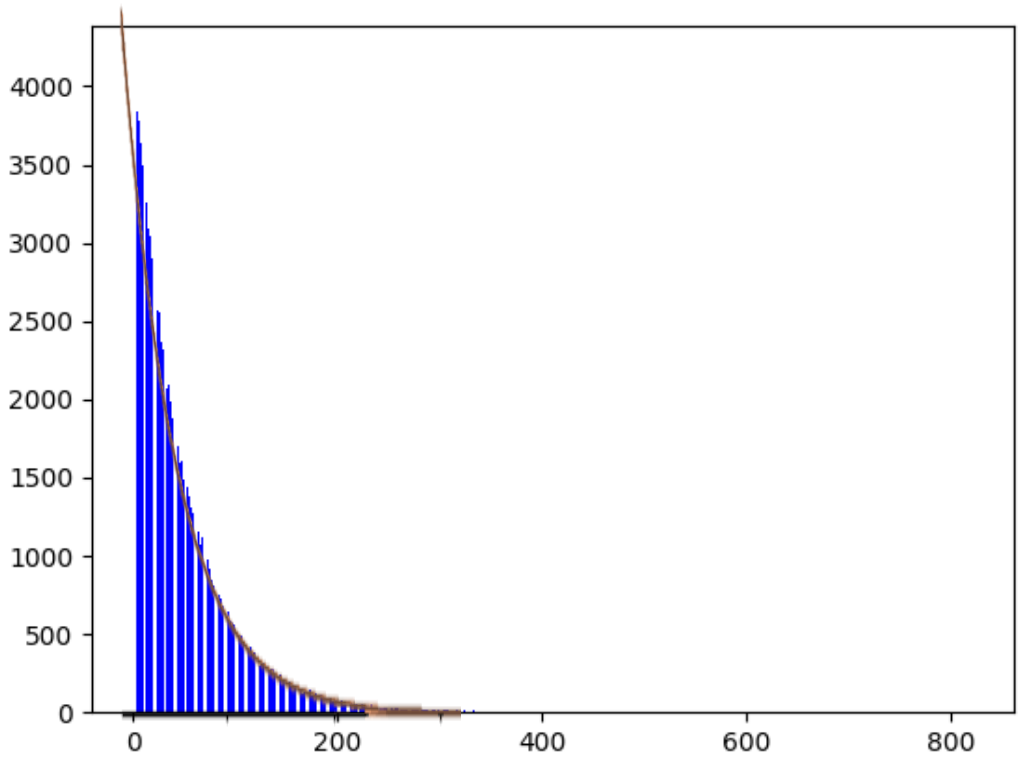
- Poisson Distribution(Ex 4)을 Normalization한 Histogram과 Compare Histogram을 PDF 형태로 작성하여 비교
- Normalization한 Histogram과 Compare Histogram 사이 상당한 유사도를 보임

Compare Histogram3





Compare Histogram4



- Time interval Histogram Lambda값의 지수 함수와 비교
- Time Interval Histogram과 Lambda값의 지수 함수와의 형태가 매우 유사함을 알 수 있음

Analysis of the Simulation

- 총 10000000번의 시행에서 Event가 발생한 사건은 [204168, 204073, 203710, 203808, 203835 …]로 Event가 발생한 확률은 기댓값인 1/λ 보다 조금 크게 나타났다.
- Time Interval은 0~4까지 4000회 이상, ~91까지 1000회 이상, ~298까지 100회 이상 나타났으며, 667이라는 특수한 Time Interval이 나타나기도 했다.
- Event Occurrence가 20인 경우의 Normalization은 평균적으로 0.09였다.
- Time Interval을 1로 설정한 경우, 확률 밀도는 4.1*10^-8

Your Conclusions

- Modeled Poisson Distribution Histogram과 Compare Histogram은 상당히 유사한 모습을 보인다.
- Modeled Poisson Distribution Histogram은 정규 분포 형상을 띄고있다
- Compare Histogram과 다르게 λ값이 최고점이 되지 않는 경우가 있다
- 10000번의 실행을 거치면 1번 정도는 특수한 특잇값이 나오는 경우가 있다.
- Time Interval이 0일 확률은 (1/50)^2가 아닐까 생각했지만, 실제로는 더 복잡한 계산이 필요하다는 것을 알 수 있었다.

Full Code

```python
import random
import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson

ffile = open("C:/Users/admin/PoissonDistribution.txt", 'w')

#np.random.seed(seed=10)
array = [[0 for col in range(1000)] for row in range(10000)]

arrForPoisson = []
calcul = 0
timeInterval = 0
sumOfCalcul = np.zeros(10000)
checkTime = 0


for restart in range(10000):
    for repeat in range(1000):
        array[repeat] = np.random.randint(1, 50)
        timeInterval = timeInterval + 1
        if array[repeat] == 1:
            calcul = calcul + 1
            arrForPoisson.insert(checkTime, timeInterval)
            #[checkTime] = timeInterval
            timeInterval = 0
            checkTime = checkTime + 1
    sumOfCalcul[restart] = calcul
    calcul = 0


unique, counts = np.unique(sumOfCalcul, return_counts=True)

np.asarray((unique, counts)).T

#print(unique)
#print(counts)



print(counts)

plt.bar(unique, counts, width=0.7, color="purple", align='center')
plt.show()



unique2, counts2 = np.unique(arrForPoisson, return_counts=True)

np.asarray((unique2, counts2)).T

print(unique2)
print(counts2)

numToFind = 0

for printRepeat in range(len(counts2)):
    print(counts2[printRepeat])
    printarray = str(counts2[printRepeat])
    ffile.write(printarray)
    ffile.write(" ")
    numToFind = numToFind + 1
    if numToFind == 35:
        ffile.write("\n")
        numToFind = 0

ffile.close
print(sum(counts2))

plt.plot(unique2, counts2, color="blue")
plt.show()
```

Thank You

Use Python, Geogebra

random, math, numpy, matplotlib