МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра математического моделирования и анализа данных

АВСЯННИК
Елизавета Дмитриевна

Лабораторная работа №1

Имитационное и статистическое моделирование

Проверил:
В.П. Кирлица

Минск, 2021

0.0.1 Лабораторная работа 1: №7, стр. 24

Осуществить моделирование $M = 100$ реализаций СВ $\xi \sim (\lambda, c)$. Исследовать точность моделирования с помощью тестов и графического анализа. Рассмотреть случаи $\lambda = 0.5, 1, 2, 4, 6, 10$.

[1]:
```
import numpy as np
import pandas as pd
from math import floor
from scipy.stats import chi2
import time

import typing as tp

import matplotlib.pyplot as plt
from seaborn import distplot, histplot, distplot

import warnings
warnings.filterwarnings('ignore')
```

Мультипликативный конгруэнтный метод моделирования БСВ

[2]:
```
def generate_brv_congruential_sample(M: int=2**31, alpha_star0: int=65539, beta :
 ↪int=65539) -> float:
    alpha_t = alpha_star0
    while True:
        alpha_t = (alpha_t * beta) % M
        yield alpha_t / M

def generate_brv_congruential(n: int=100, M: int=2**31, alpha_star0: int=65539,␣
 ↪beta :int=65539) -> np.ndarray:
    alpha = np.array([])
    generator = generate_brv_congruential_sample(M=M, alpha_star0=alpha_star0,␣
 ↪beta=beta)
    for i in range(n):
        alpha = np.append(alpha, next(generator))
    return alpha
```

Метод Макларена-Марсальи моделирования БСВ

[3]:
```
def generate_brv_mm_sample(k: int=128) -> float:
    v = np.array([])
    brv_cong_b = generate_brv_congruential_sample()
    t = time.perf_counter()
    alpha_star0 = beta =  int(10**9*float((t-int(t))))
    brv_cong_c = generate_brv_congruential_sample(alpha_star0=alpha_star0,␣
 ↪beta=beta)
    for i in range(k):
            v = np.append(v, next(brv_cong_b))
```

```
    while True:
        index = floor(next(brv_cong_c) * k)
        alpha_t = v[index]
        v[index] = next(brv_cong_b)
        yield alpha_t

def generate_brv_mm(n: int=100, k: int=128) -> np.ndarray:
    alpha = np.array([])
    generator = generate_brv_mm_sample(k=k)
    for i in range(n):
        alpha = np.append(alpha, next(generator))
    return alpha
```

Распределение Пуассона Алгоритм моделирования $\xi \sim (\lambda, c)$ включает следующие шаги:

1) Моделирование реализаций $a_t (t = 1, 2, ...)$ БСВ
2) Принятие решения о том, что реализацией ξ является величина x, определяемая соотношениями

$x = \min \left\{ k : \prod_{i=1}^{k+1} a_i < e^{-\lambda}, k = 0, 1, ... \right\}$

[4]:
```
def generate_poisson_sample(lambda_: float=0.5) -> int:
    brv_mm = generate_brv_mm_sample()
    while True:
        exp_lamda_ = np.exp(-lambda_)
        alpha_i = next(brv_mm)
        k = 0
        while(alpha_i >= exp_lamda_):
            k += 1
            alpha_i = alpha_i * next(brv_mm)
        yield k

def generate_poisson(n: int=100, lambda_: float=0.5) -> np.ndarray:
    alpha = np.array([])
    generator = generate_poisson_sample(lambda_=lambda_)
    for i in range(n):
        alpha = np.append(alpha, next(generator))
    return alpha
```

χ^2 - критерий согласия Пирсена

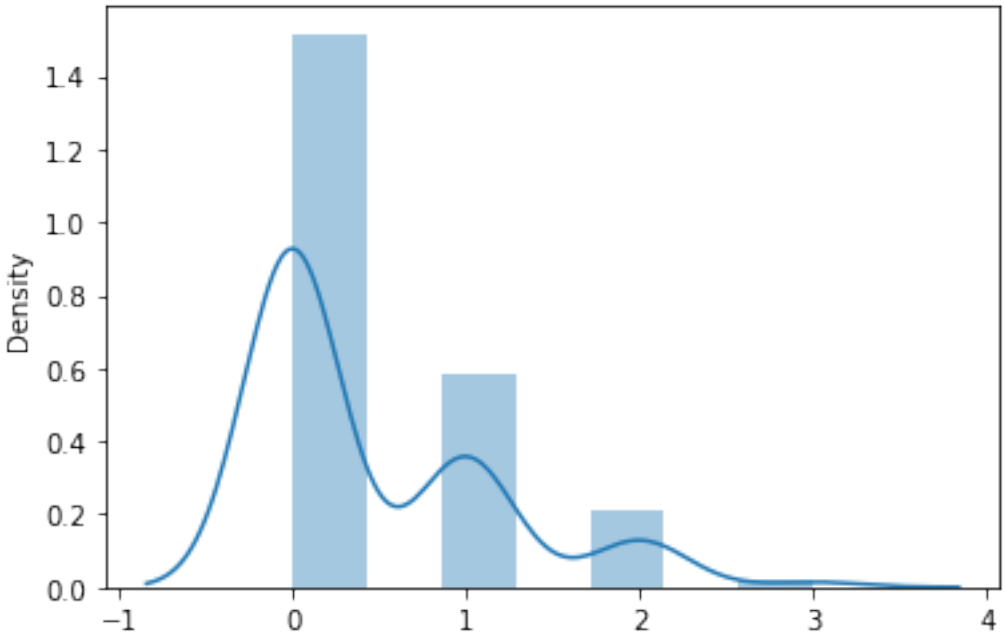
[5]:
```
def poisson_proba(k: int, lambda_: float=0.5):
    return np.exp(-lambda_) * np.power(lambda_, k) / np.math.factorial(k)
```

[6]:
```
def calculate_chi_square(samples: np.ndarray, lambda_: float=0.5, epsilon:␣
↪float=0.05) -> tp.Tuple[float, float, bool]:
    stats = {}
```

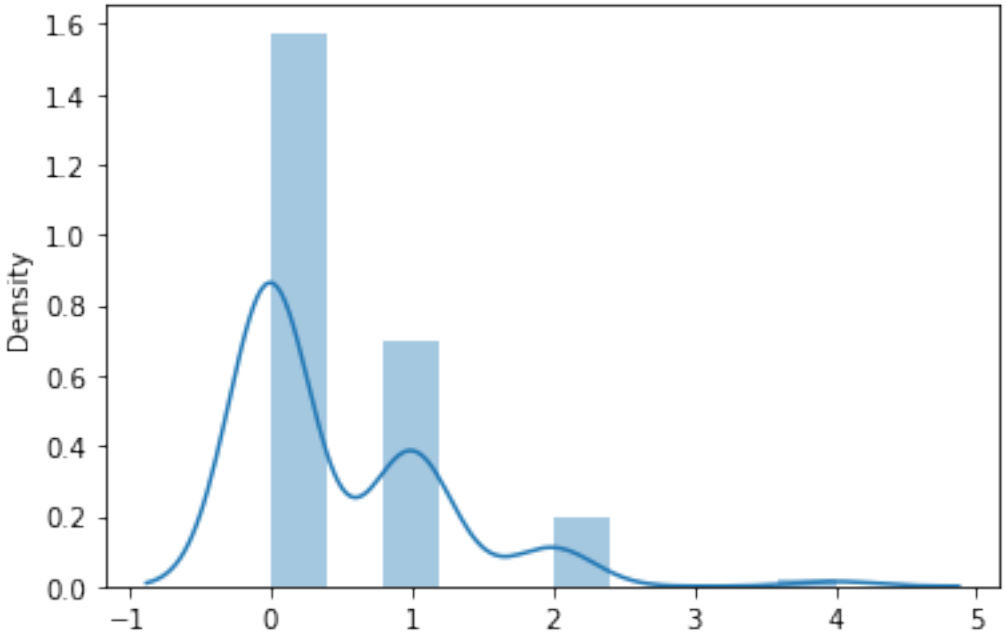
```
    for sample in samples:
        stats.setdefault(sample, [0, poisson_proba(sample, lambda_=lambda_)])
        stats[sample][0] += 1

    samples_length = samples.shape[0]
    chi_square = 0
    for el in stats:
        chi_square += np.square(stats[el][0] - samples_length * stats[el][1]) /␣
↪(samples_length * stats[el][1])

    G =  chi2.ppf(1 - epsilon, len(stats) - 1)
    return chi_square, G, chi_square < G
```

[7]:
```
lambdas = [0.5, 1, 2, 4, 6, 10]
n = 100
success_probas = {}

for l in lambdas:
    for i in range(10):
        generated = generate_poisson(n=n, lambda_=l)
        chi_square, G, pass_ = calculate_chi_square(generated, lambda_=l)
        success_probas.setdefault(l, [])
        success_probas[l].append(pass_)
        print(f'Lambda: {l}')
        print(f'Expriment № {i + 1}')
        print(f'Chi-square: {chi_square}, G: {G}, Pass: {pass_}')
        distplot(generated)
        plt.show()
```

```
Lambda: 0.5
Expriment № 1
Chi-square: 1.5674273592758254, G: 7.814727903251179, Pass: True
```

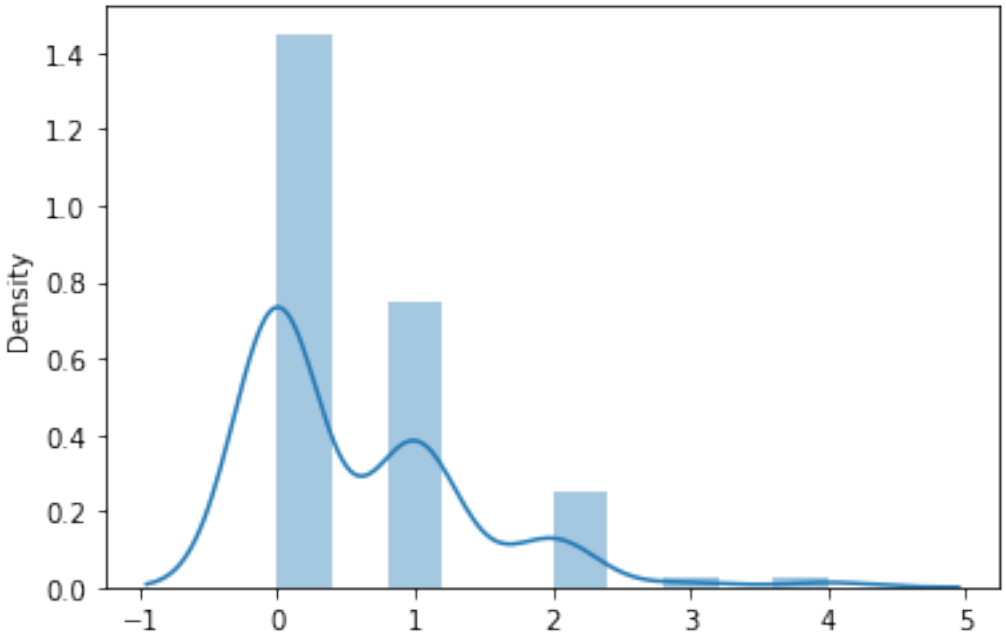



```
Lambda: 0.5
Expriment № 2
Chi-square: 4.781422240075673, G: 7.814727903251179, Pass: True
```

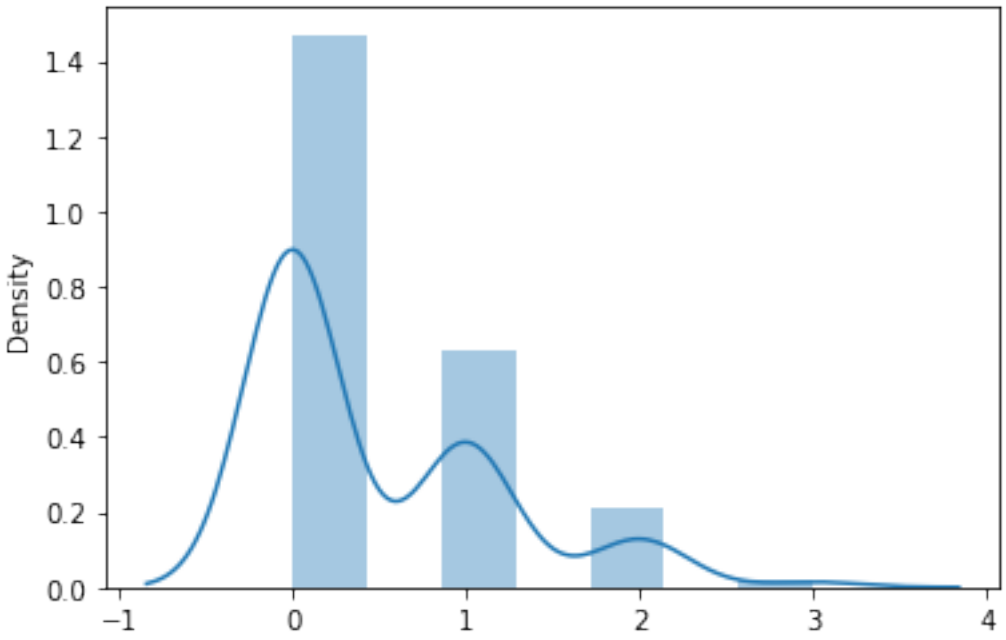



```
Lambda: 0.5
```

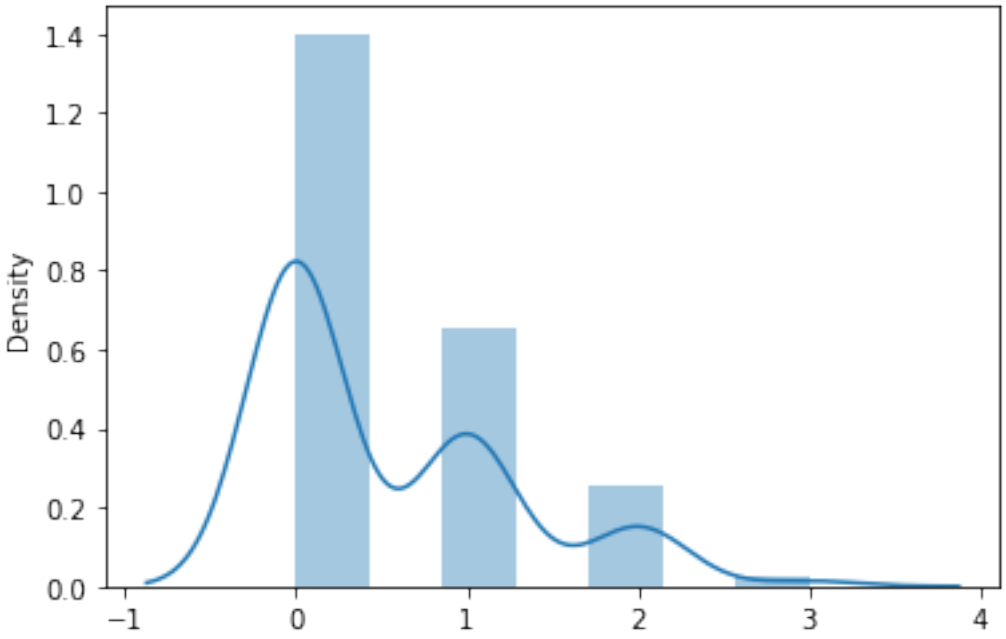
```
Expriment № 3
Chi-square: 5.435000910984613, G: 9.487729036781154, Pass: True
```



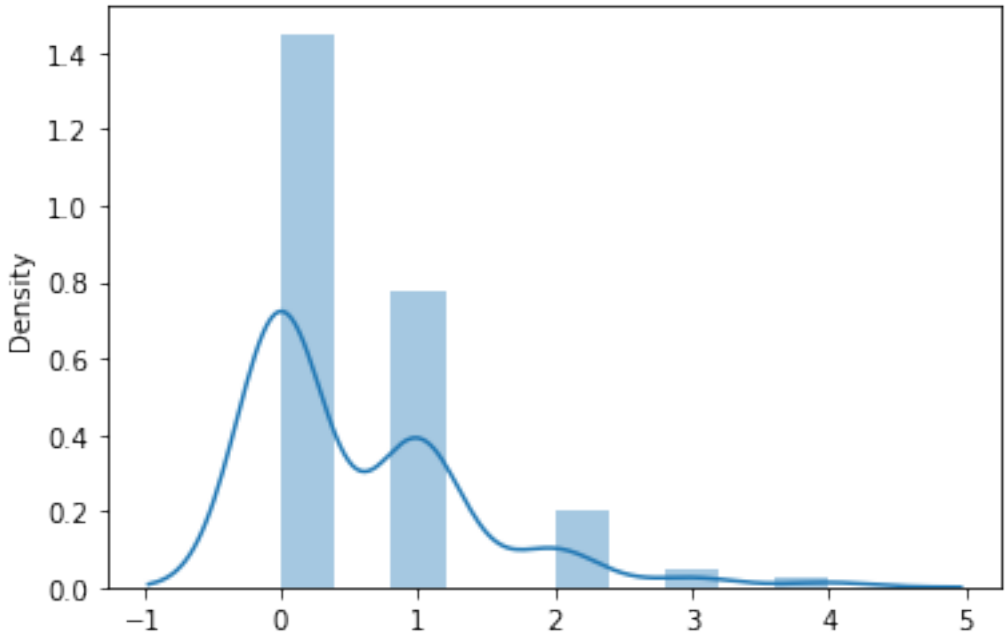

```
Lambda: 0.5
Expriment № 4
Chi-square: 0.7760411493397641, G: 7.814727903251179, Pass: True
```



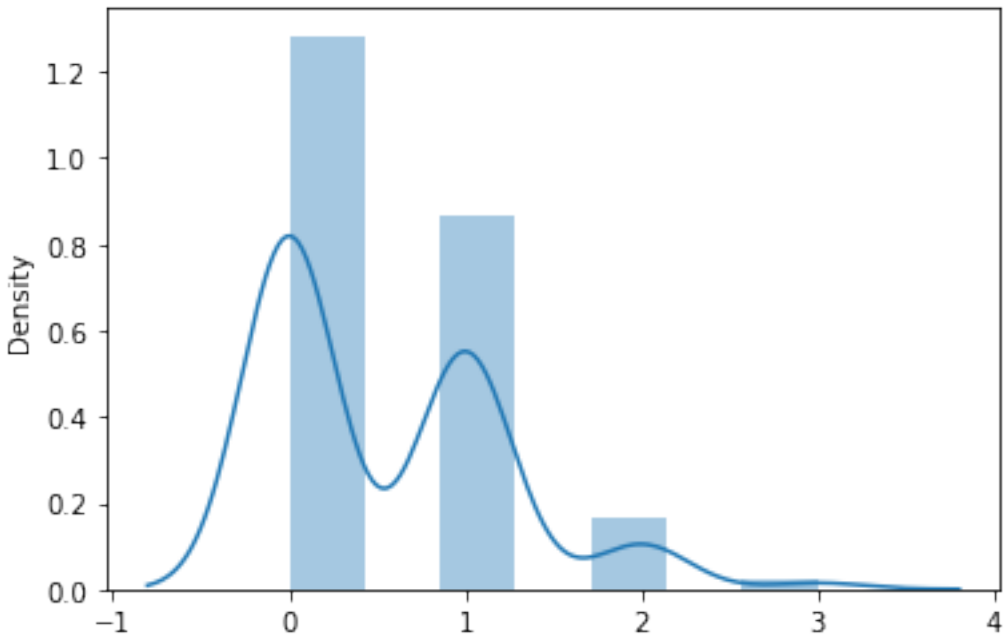
```
Lambda: 0.5
Expriment № 5
Chi-square: 1.781761124466842, G: 7.814727903251179, Pass: True
```




```
Lambda: 0.5
Expriment № 6
Chi-square: 5.072282231430584, G: 9.487729036781154, Pass: True
```

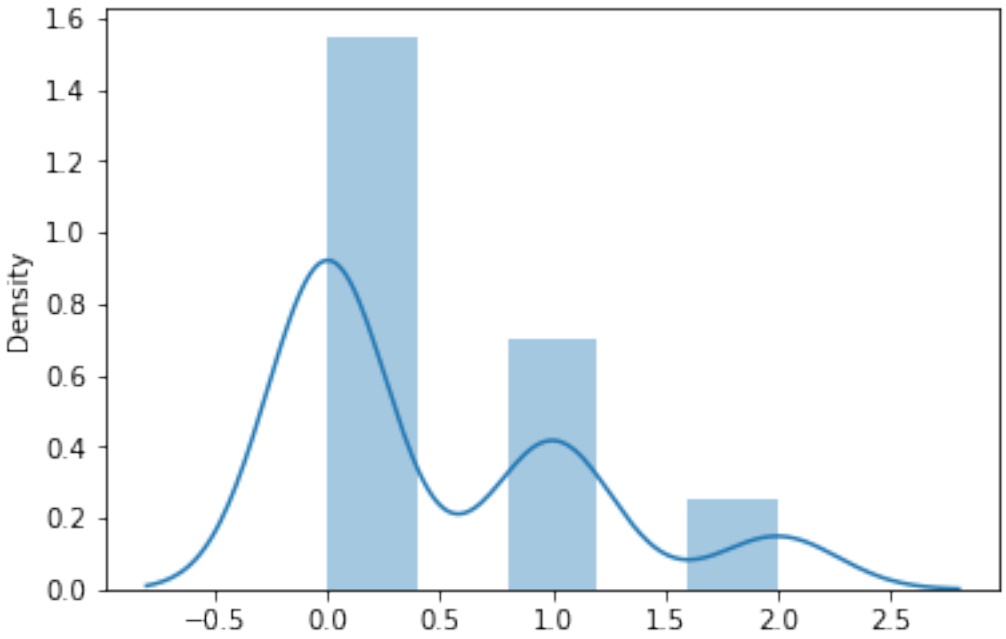



```
Lambda: 0.5
Expriment № 7
Chi-square: 2.0950181658998663, G: 7.814727903251179, Pass: True
```

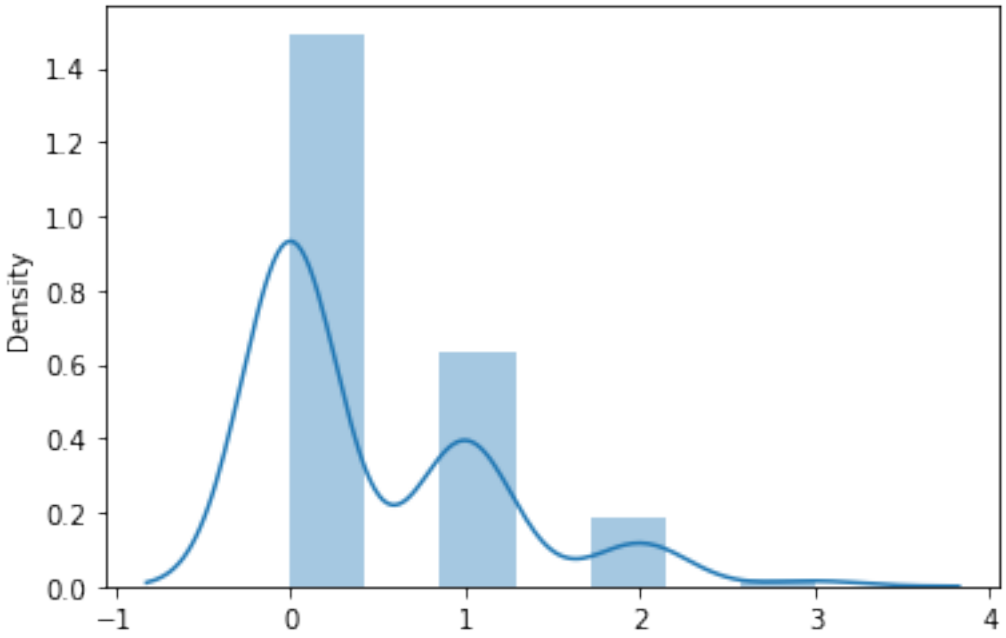



```
Lambda: 0.5
```

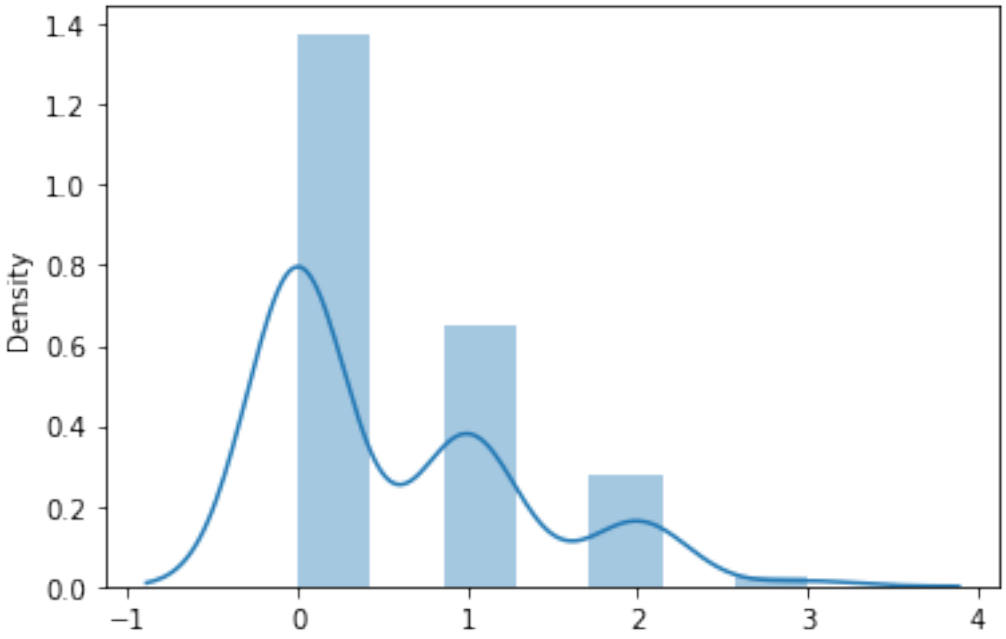
```
Expriment № 8
Chi-square: 0.9797975391948918, G: 5.991464547107979, Pass: True
```



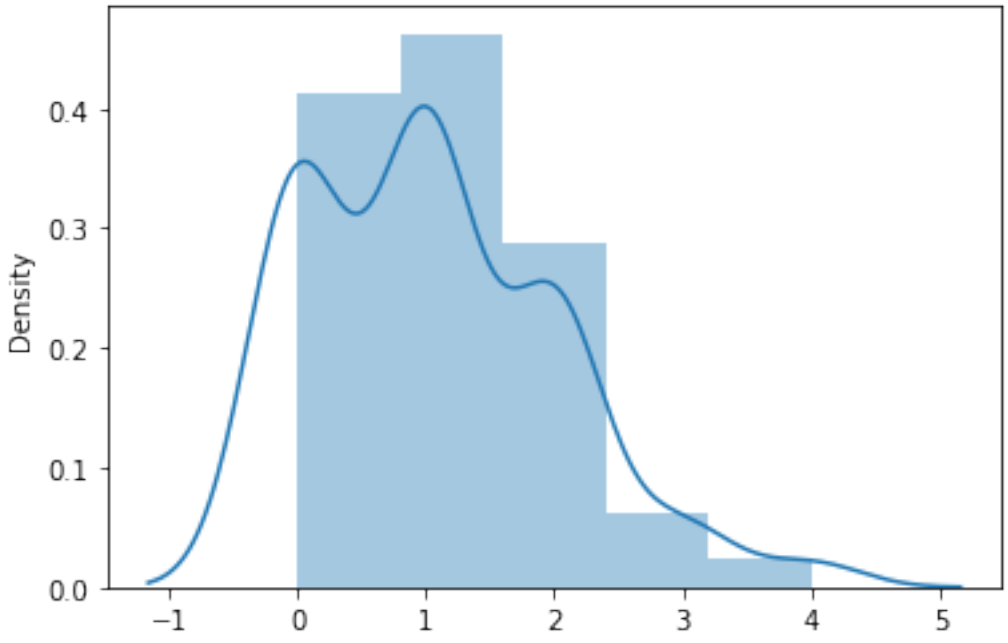

```
Lambda: 0.5
Expriment № 9
Chi-square: 0.6276562349767525, G: 7.814727903251179, Pass: True
```



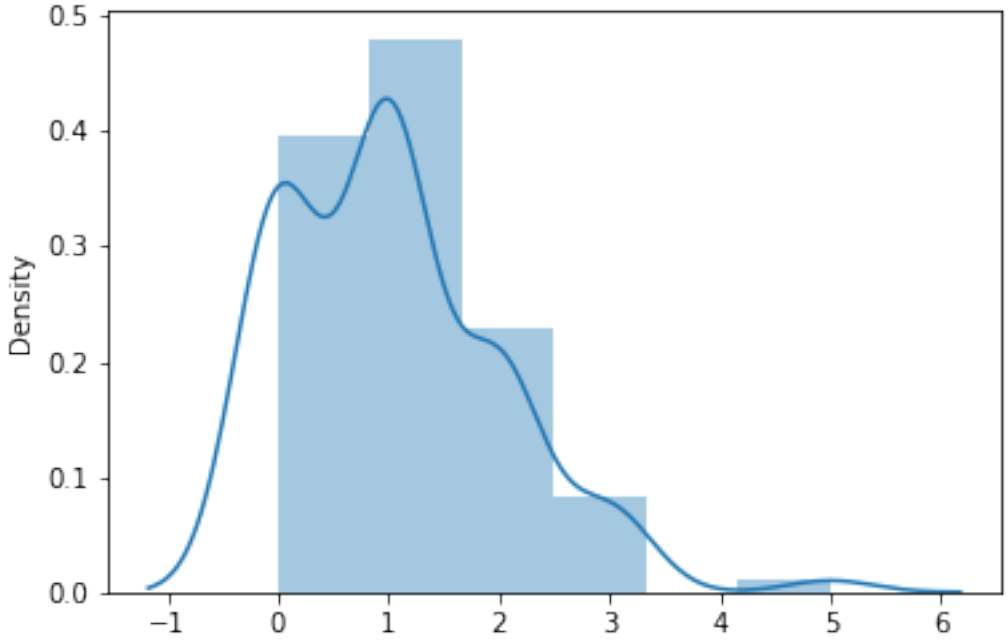
```
Lambda: 0.5
Expriment № 10
Chi-square: 2.8534299504219254, G: 7.814727903251179, Pass: True
```




```
Lambda: 1
Expriment № 1
Chi-square: 1.8957777038949104, G: 9.487729036781154, Pass: True
```

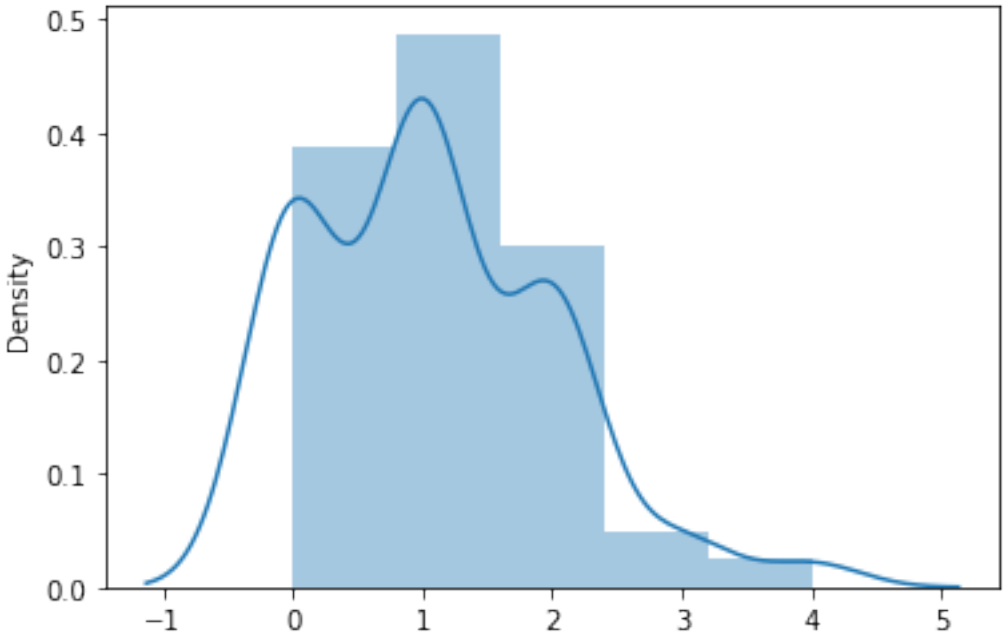



```
Lambda: 1
Expriment № 2
Chi-square: 2.3820304519193005, G: 9.487729036781154, Pass: True
```

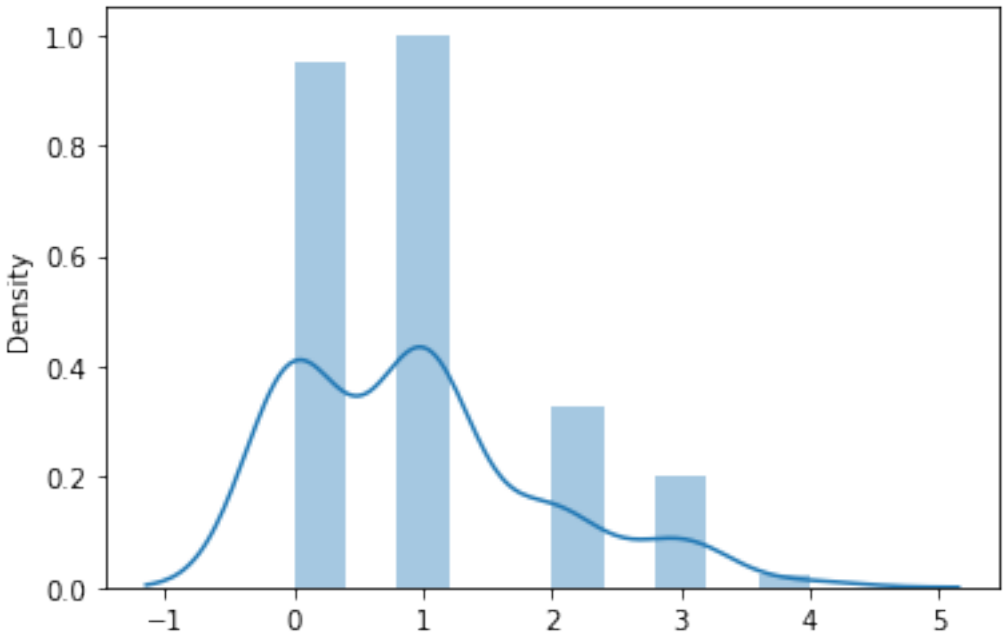



```
Lambda: 1
```

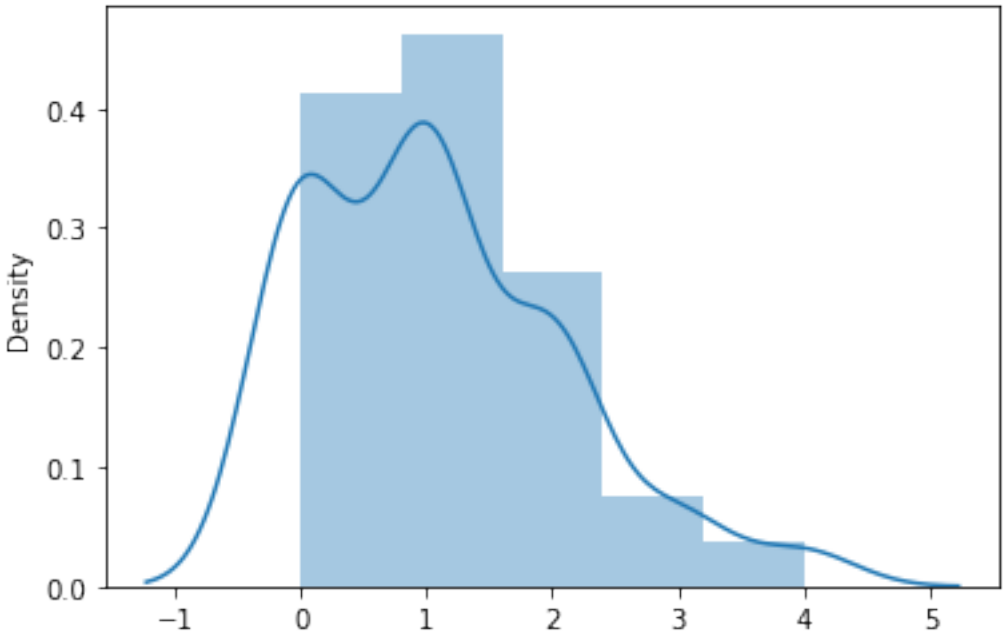
```
Expriment № 3
Chi-square: 3.6354780741086996, G: 9.487729036781154, Pass: True
```



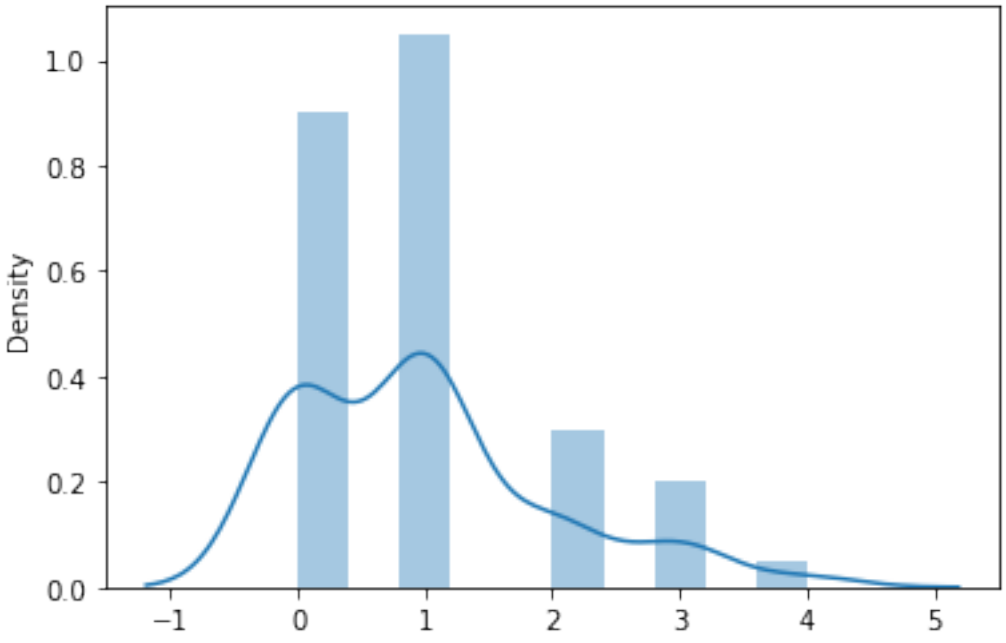

```
Lambda: 1
Expriment № 4
Chi-square: 2.6568966158634426, G: 9.487729036781154, Pass: True
```



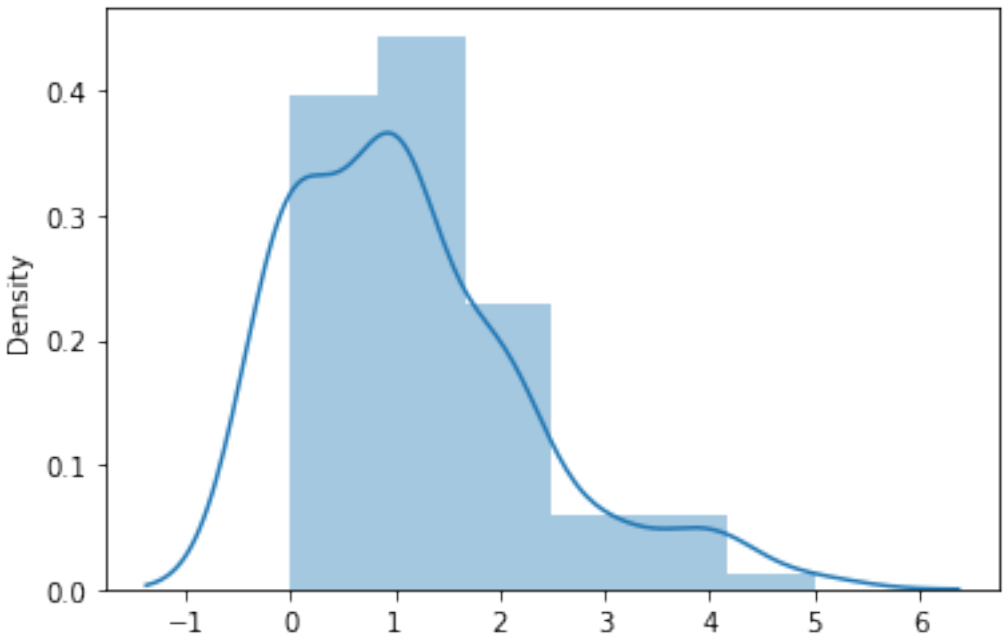
```
Lambda: 1
Expriment № 5
Chi-square: 2.1676058867408146, G: 9.487729036781154, Pass: True
```




```
Lambda: 1
Expriment № 6
Chi-square: 3.68984371067788, G: 9.487729036781154, Pass: True
```

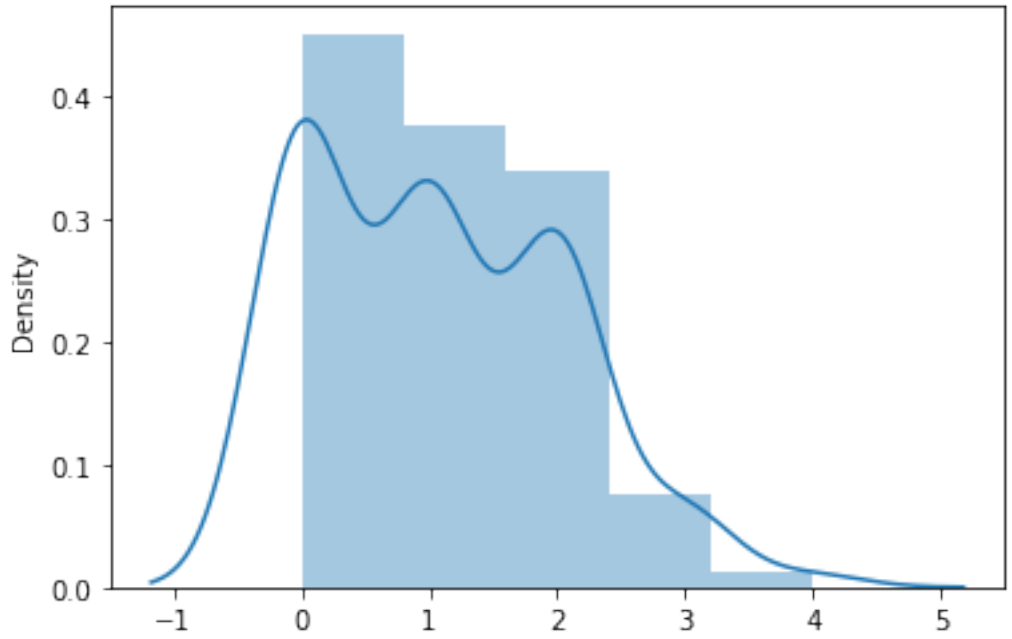



```
Lambda: 1
Expriment № 7
Chi-square: 10.030995570833161, G: 11.070497693516351, Pass: True
```

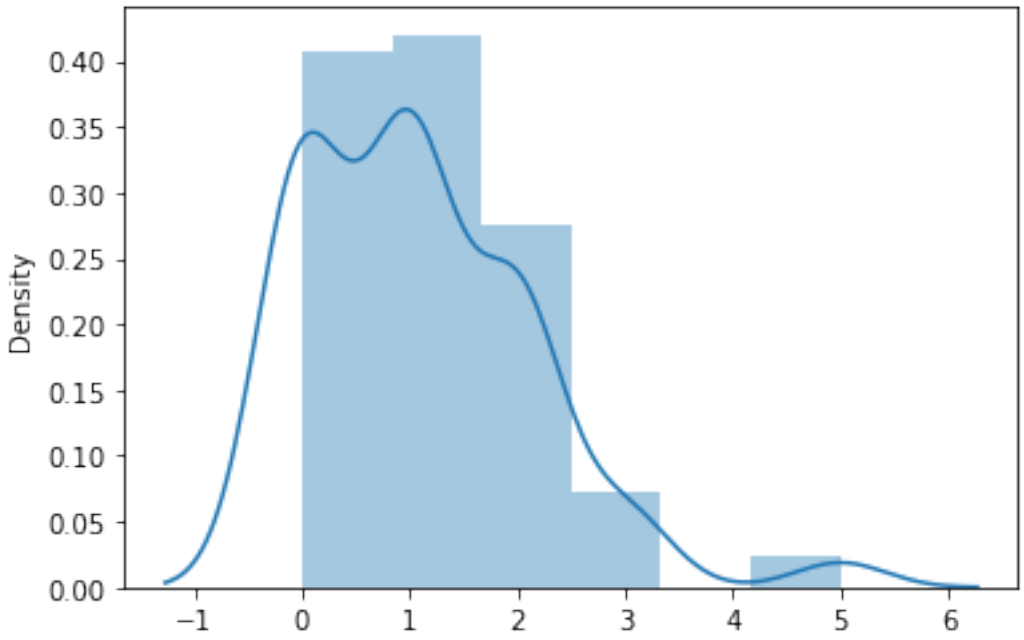



```
Lambda: 1
```

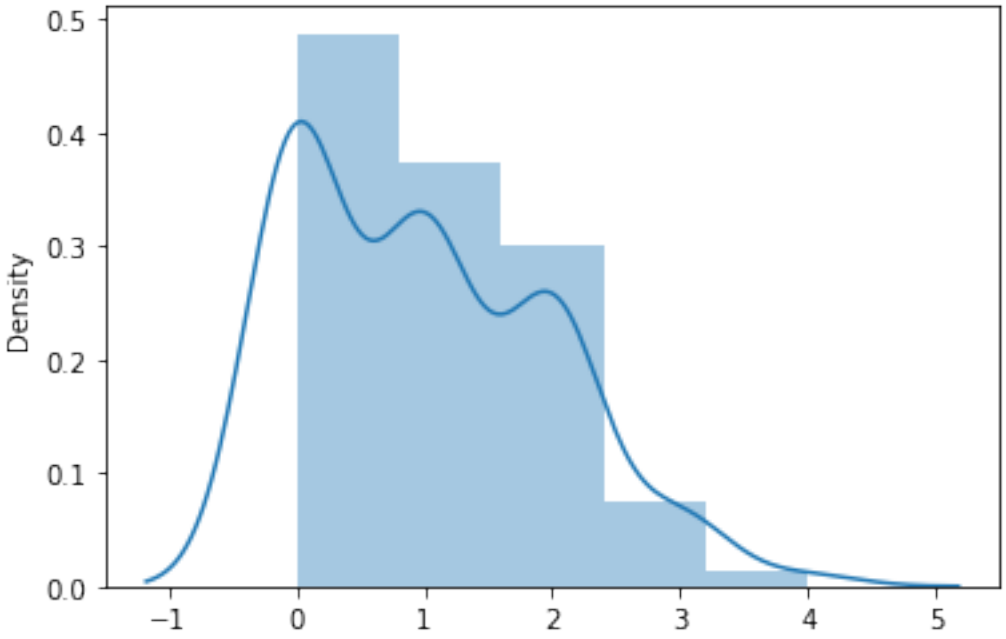
```
Expriment № 8
Chi-square: 5.4839097174608495, G: 9.487729036781154, Pass: True
```



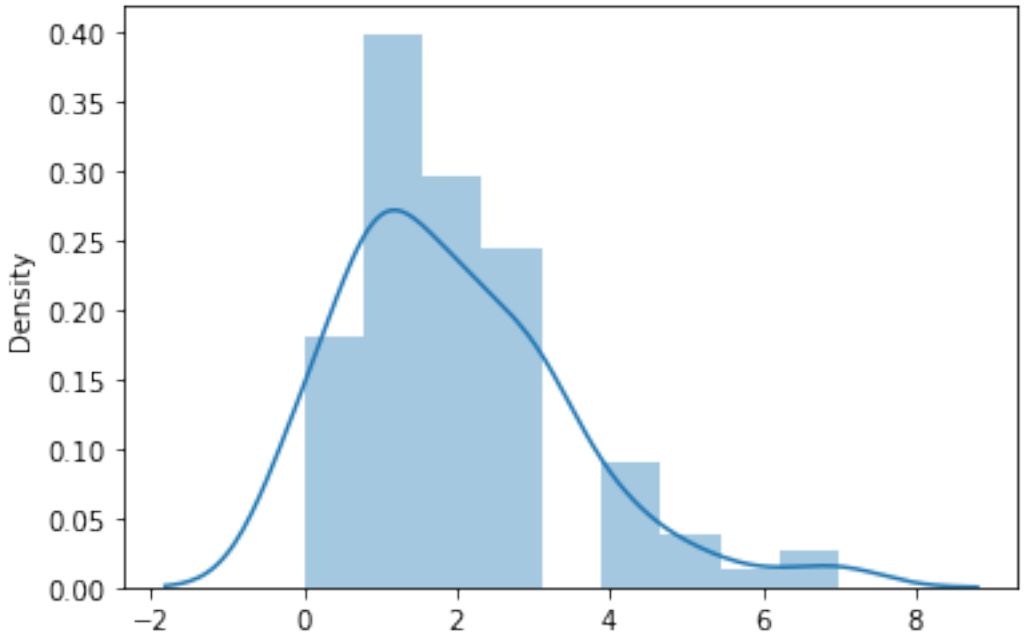

```
Lambda: 1
Expriment № 9
Chi-square: 10.808704120142341, G: 9.487729036781154, Pass: False
```



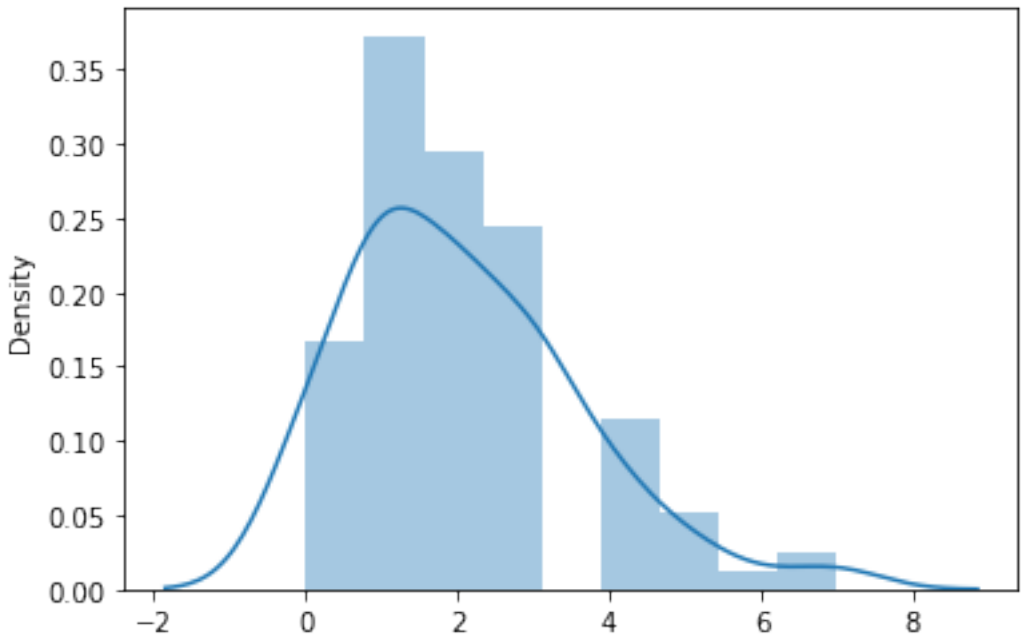
```
Lambda: 1
Expriment № 10
Chi-square: 3.282101436409023, G: 9.487729036781154, Pass: True
```




```
Lambda: 2
Expriment № 1
Chi-square: 9.821010215054878, G: 14.067140449340169, Pass: True
```

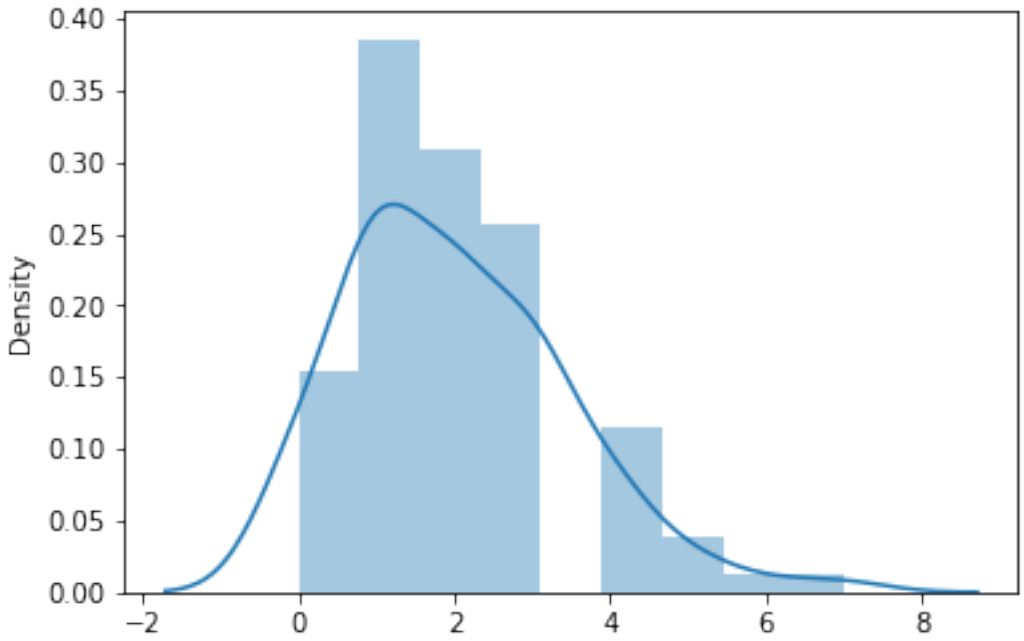



```
Lambda: 2
Expriment № 2
Chi-square: 8.87890556244122, G: 14.067140449340169, Pass: True
```

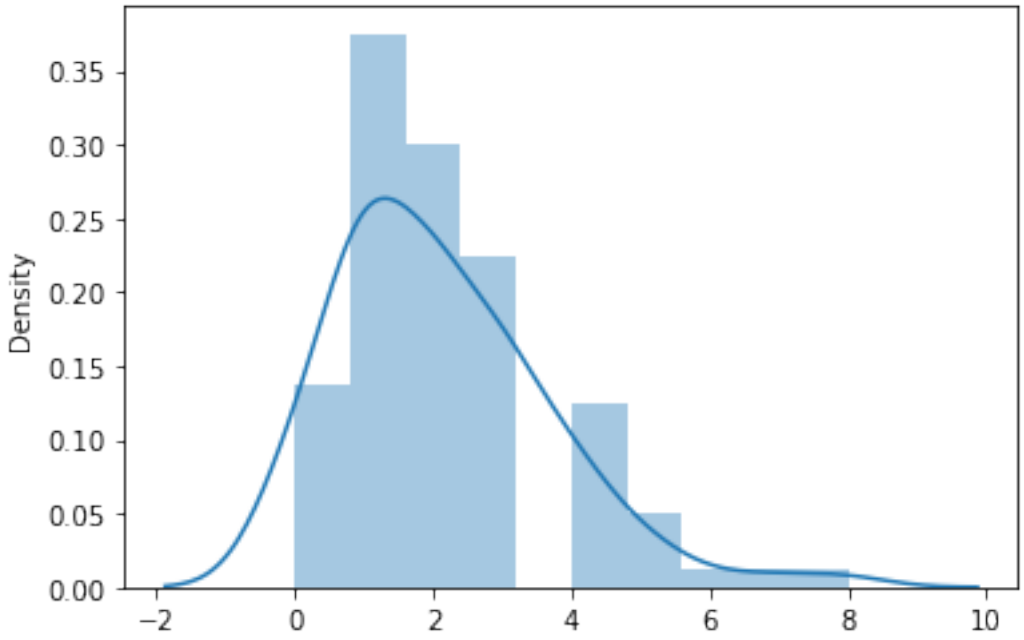



```
Lambda: 2
```

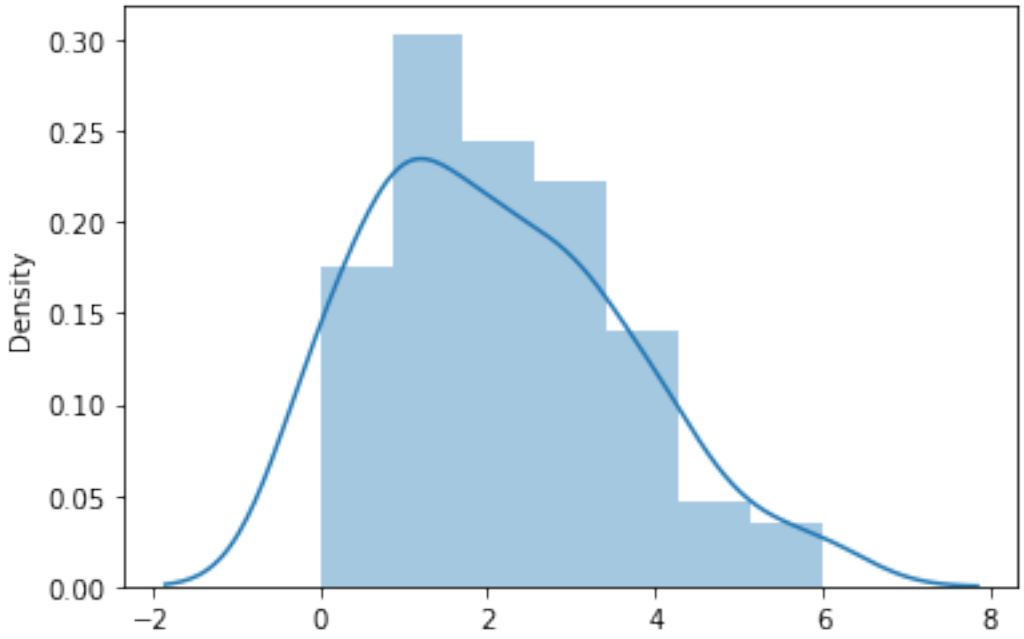
```
Expriment № 3
Chi-square: 2.441190436247892, G: 14.067140449340169, Pass: True
```



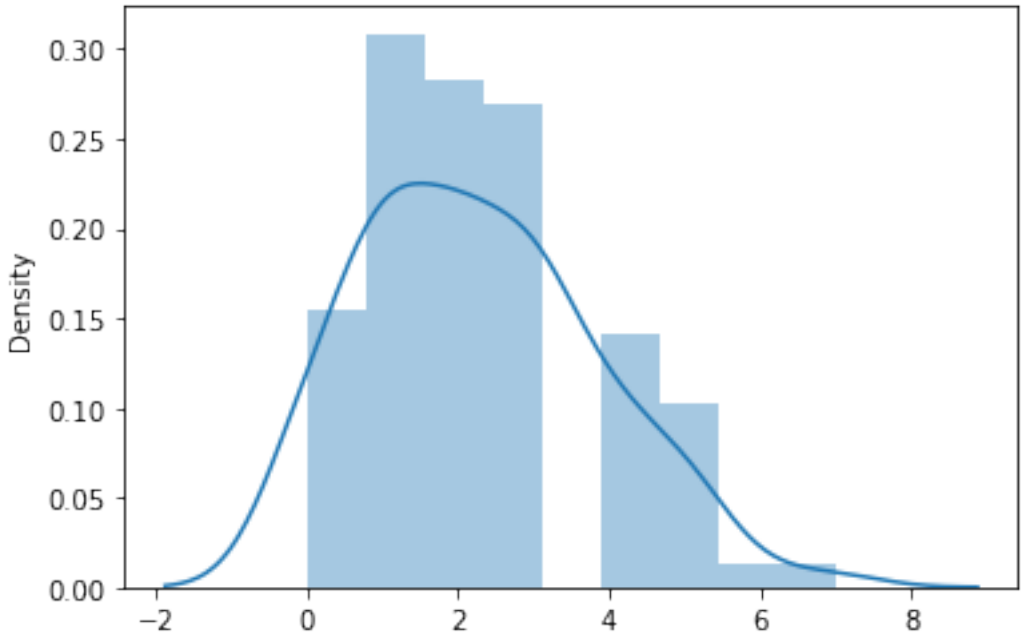

```
Lambda: 2
Expriment № 4
Chi-square: 12.29914429104343, G: 15.50731305586545, Pass: True
```



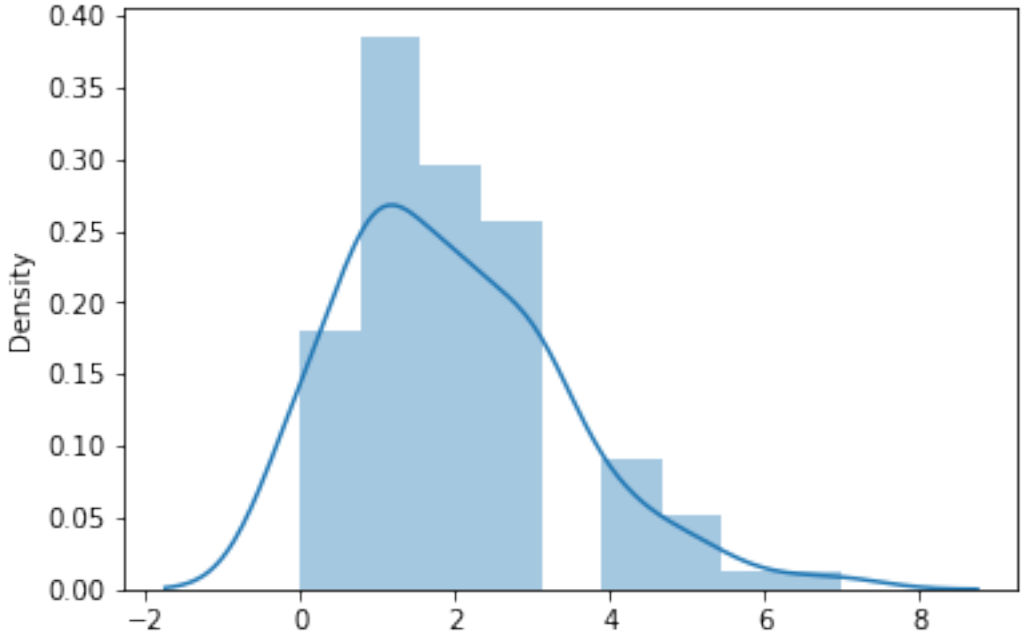
```
Lambda: 2
Expriment № 5
Chi-square: 5.320957503567371, G: 12.591587243743977, Pass: True
```




```
Lambda: 2
Expriment № 6
Chi-square: 9.01745036429617, G: 14.067140449340169, Pass: True
```

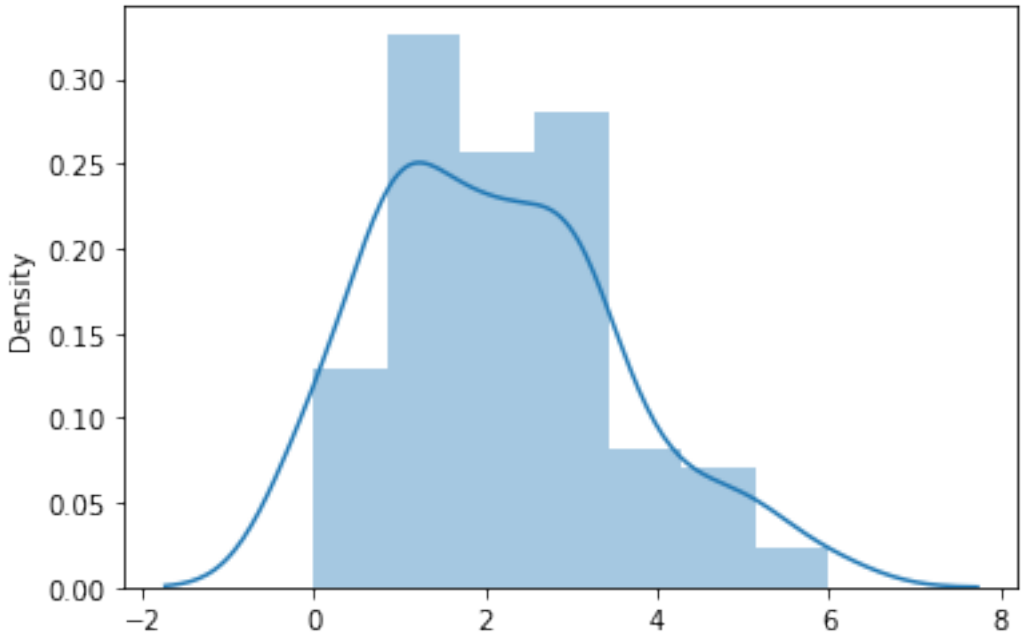



```
Lambda: 2
Expriment № 7
Chi-square: 2.93995172292571, G: 14.067140449340169, Pass: True
```

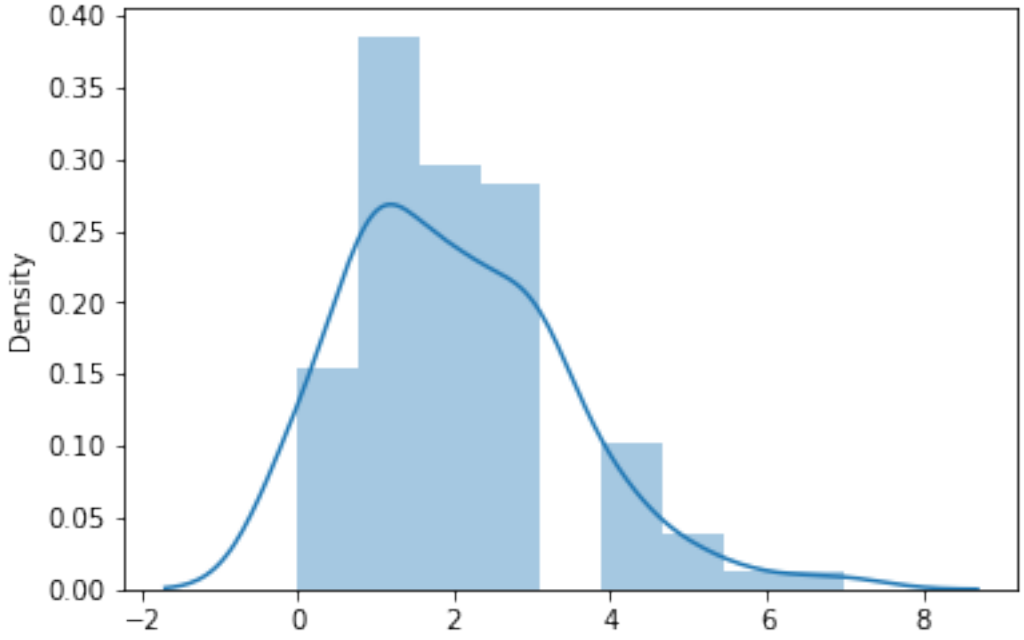



```
Lambda: 2
```

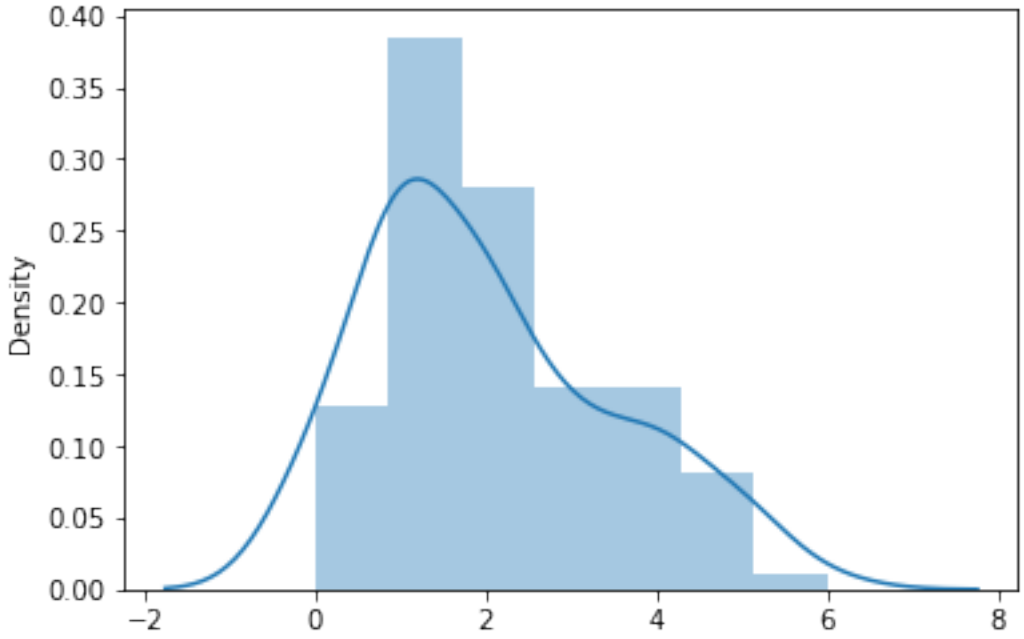
```
Expriment № 8
Chi-square: 5.985972552471129, G: 12.591587243743977, Pass: True
```



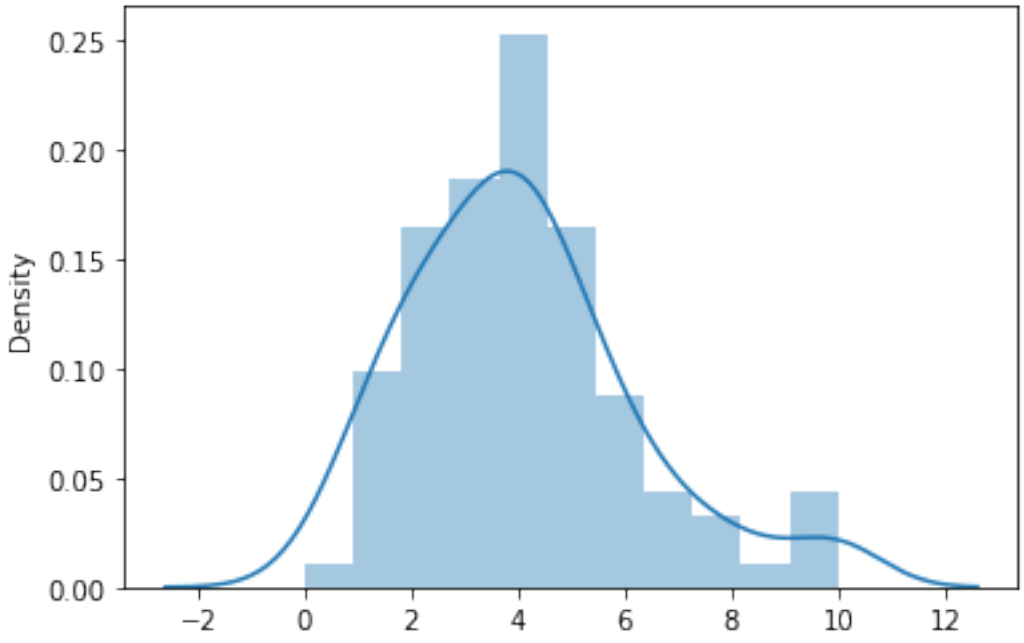

```
Lambda: 2
Expriment № 9
Chi-square: 3.4756582900981825, G: 14.067140449340169, Pass: True
```



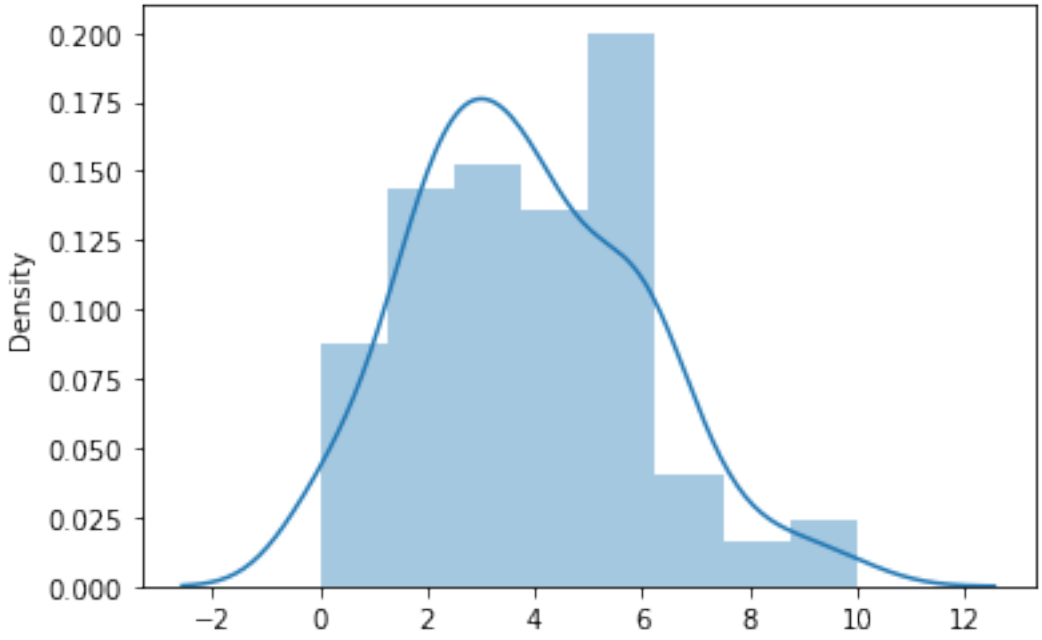
```
Lambda: 2
Expriment № 10
Chi-square: 8.350470504128937, G: 12.591587243743977, Pass: True
```




```
Lambda: 4
Expriment № 1
Chi-square: 25.779899385619288, G: 18.307038053275146, Pass: False
```

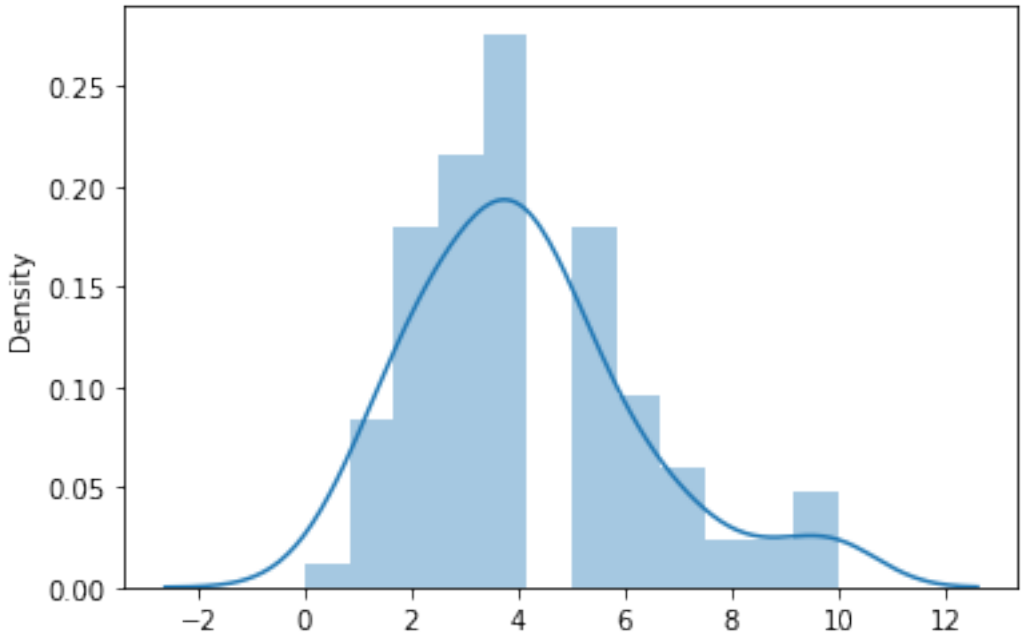



```
Lambda: 4
Expriment № 2
Chi-square: 12.109700956300166, G: 18.307038053275146, Pass: True
```

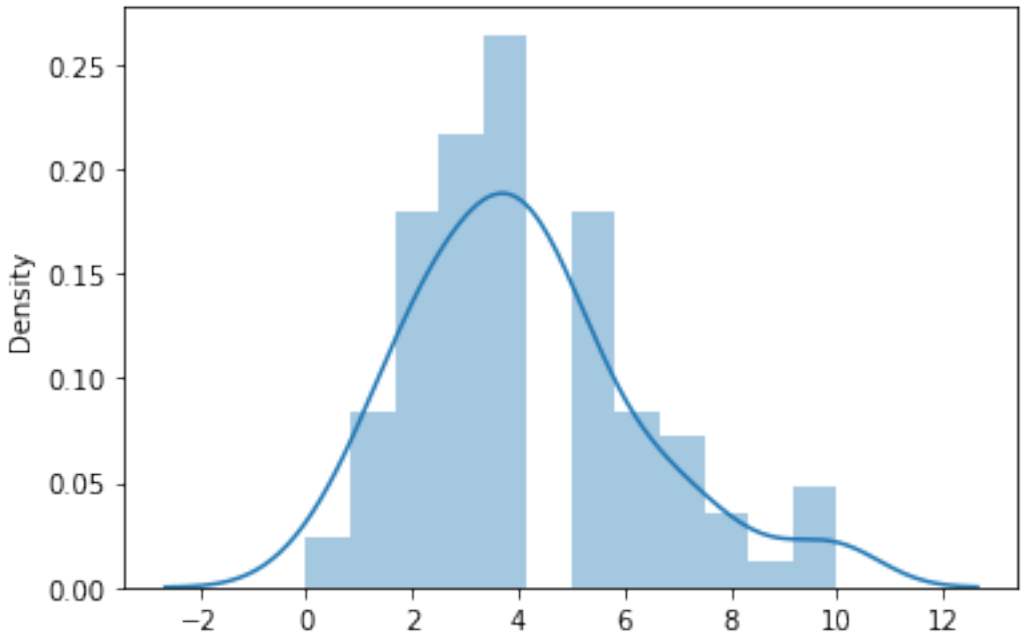



```
Lambda: 4
```

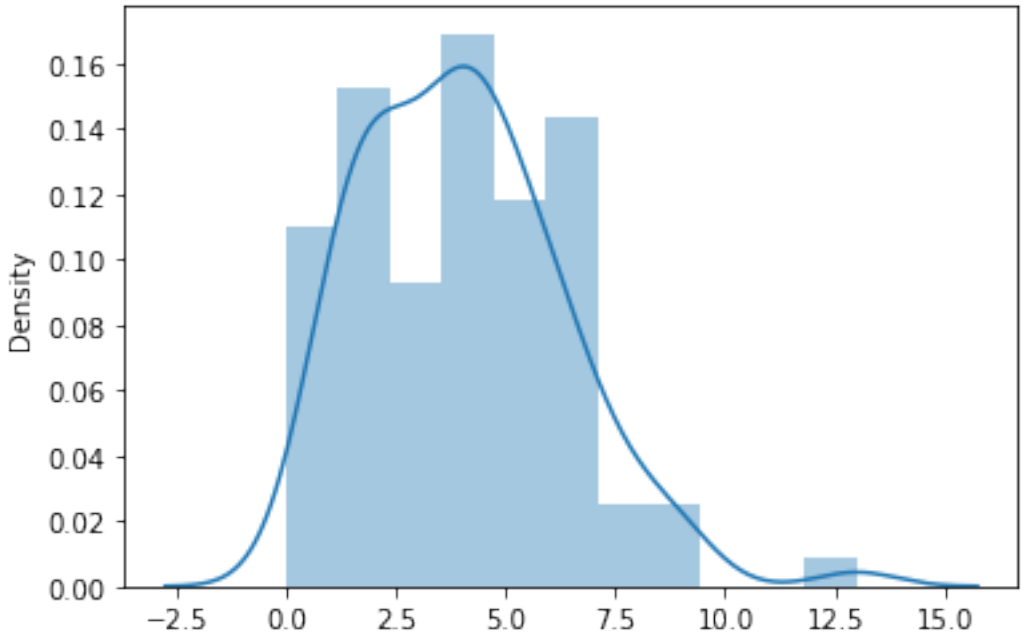
```
Expriment № 3
Chi-square: 25.302965350417654, G: 18.307038053275146, Pass: False
```



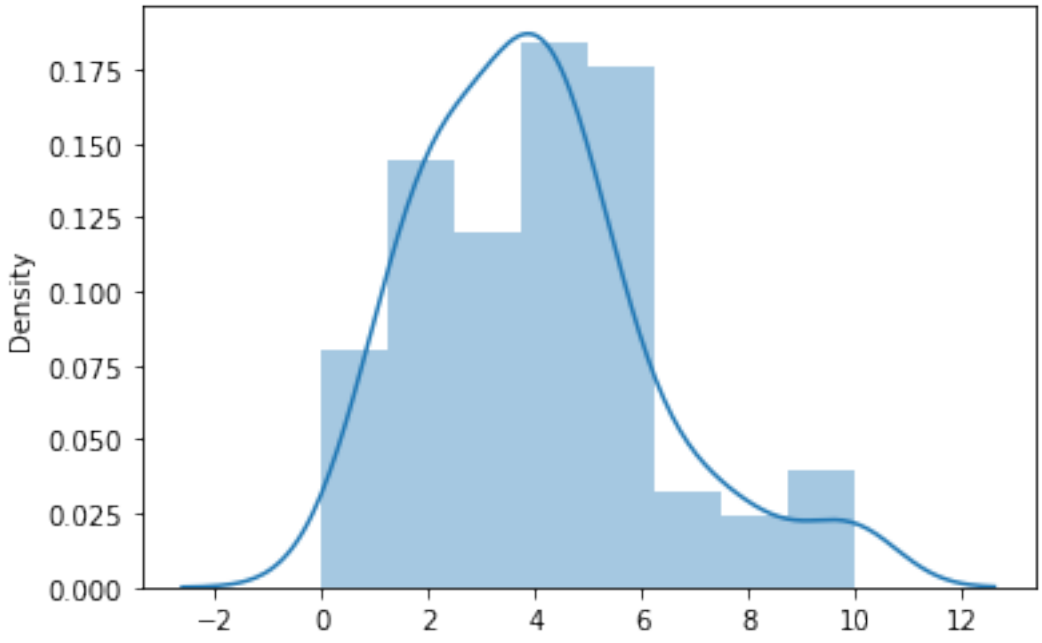

```
Lambda: 4
Expriment № 4
Chi-square: 24.457600439504077, G: 18.307038053275146, Pass: False
```



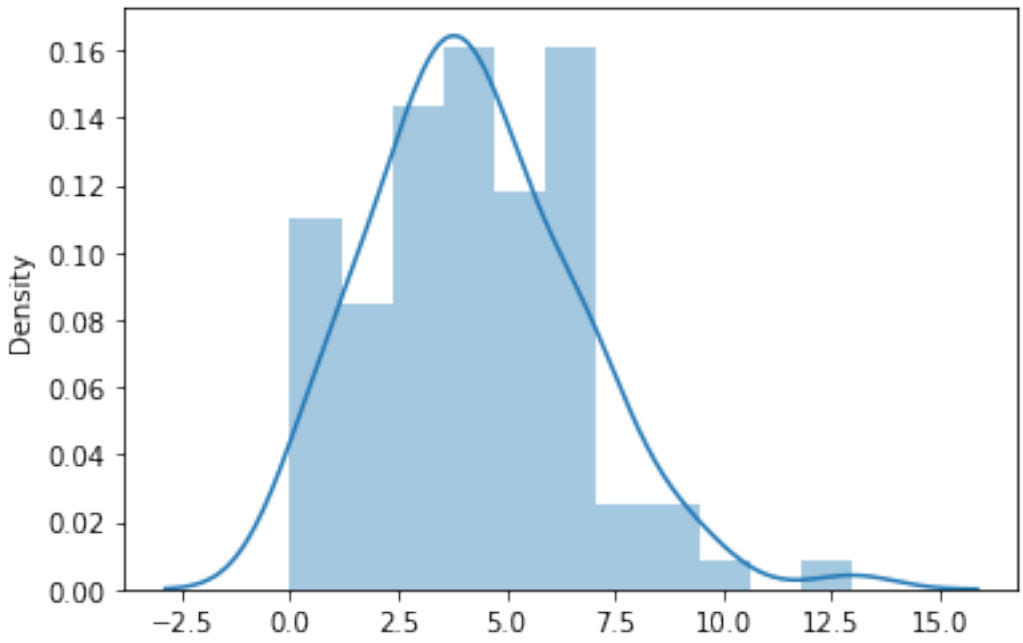
```
Lambda: 4
Expriment № 5
Chi-square: 58.87427059378406, G: 18.307038053275146, Pass: False
```




```
Lambda: 4
Expriment № 6
Chi-square: 28.556727172461237, G: 18.307038053275146, Pass: False
```

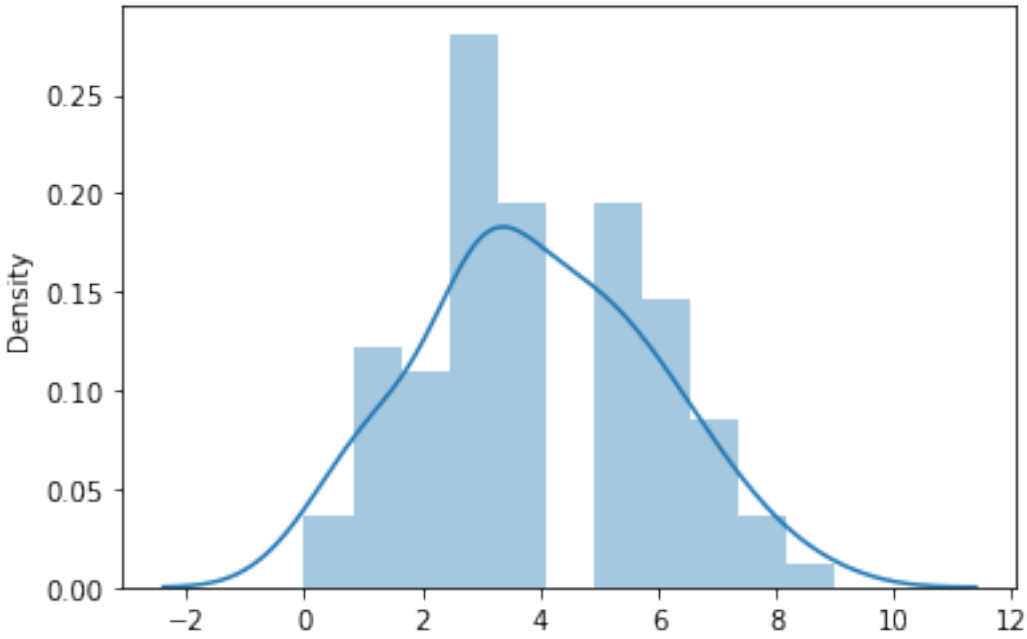



```
Lambda: 4
Expriment № 7
Chi-square: 59.10906681263892, G: 19.67513757268249, Pass: False
```

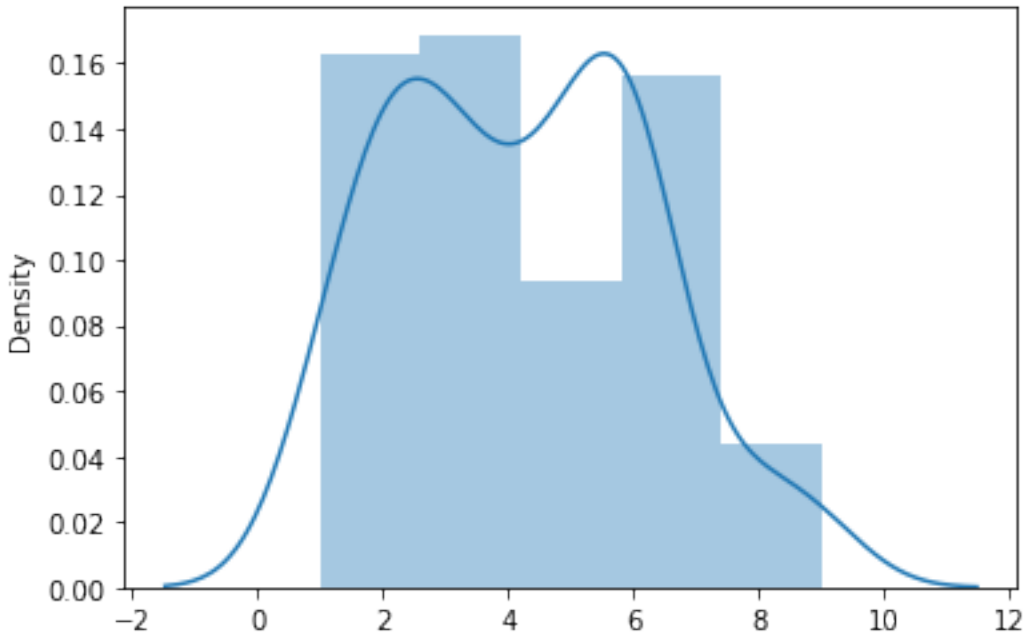



```
Lambda: 4
```

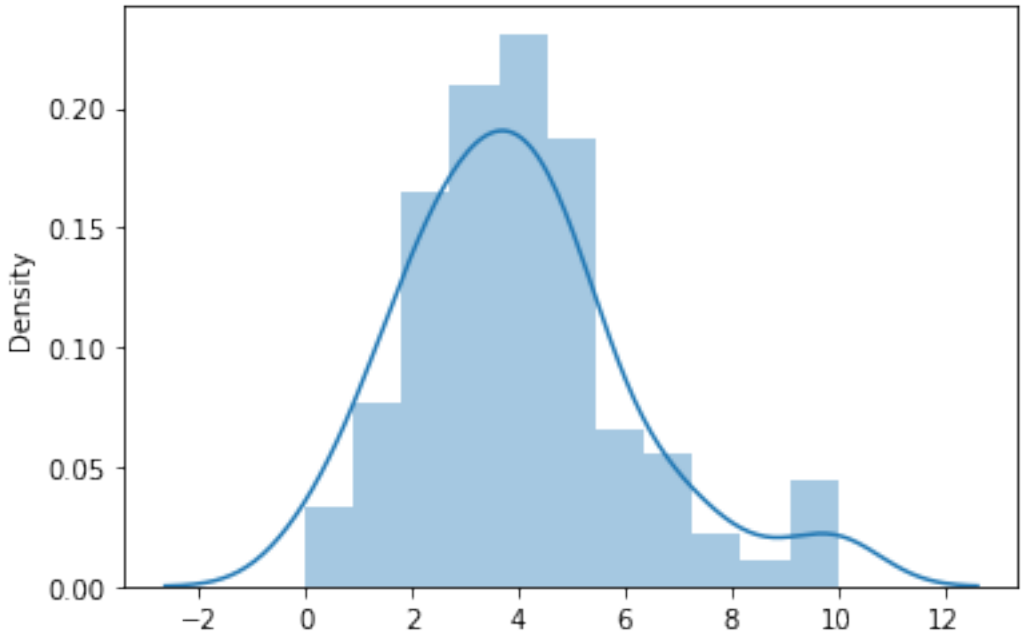
```
Expriment № 8
Chi-square: 5.667297688999192, G: 16.918977604620448, Pass: True
```



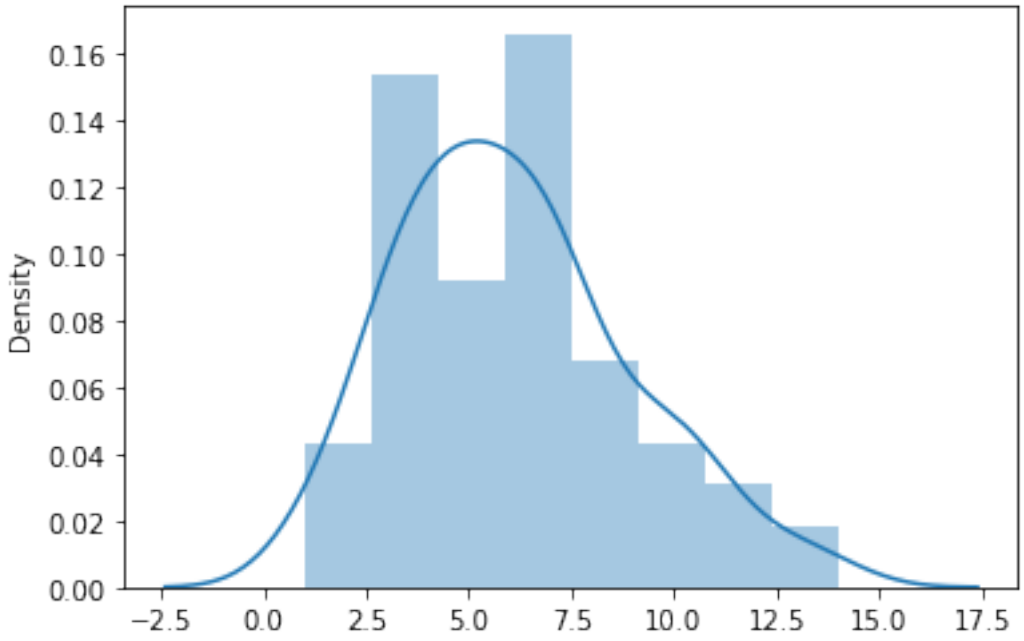

```
Lambda: 4
Expriment № 9
Chi-square: 22.035472601213133, G: 15.50731305586545, Pass: False
```



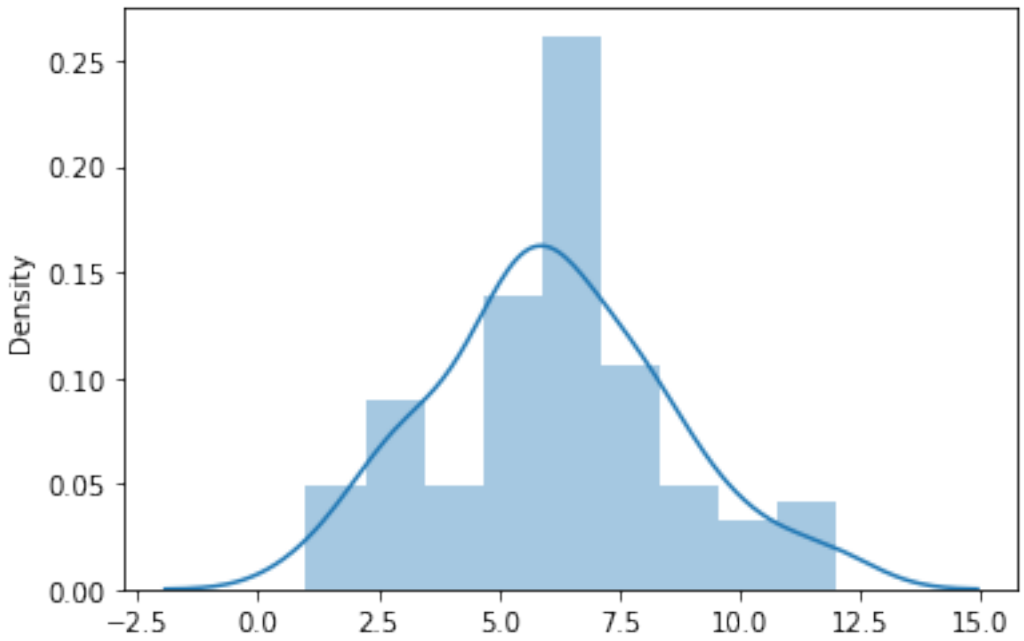
```
Lambda: 4
Expriment № 10
Chi-square: 26.200582397105137, G: 18.307038053275146, Pass: False
```




```
Lambda: 6
Expriment № 1
Chi-square: 16.432938280527456, G: 22.362032494826934, Pass: True
```

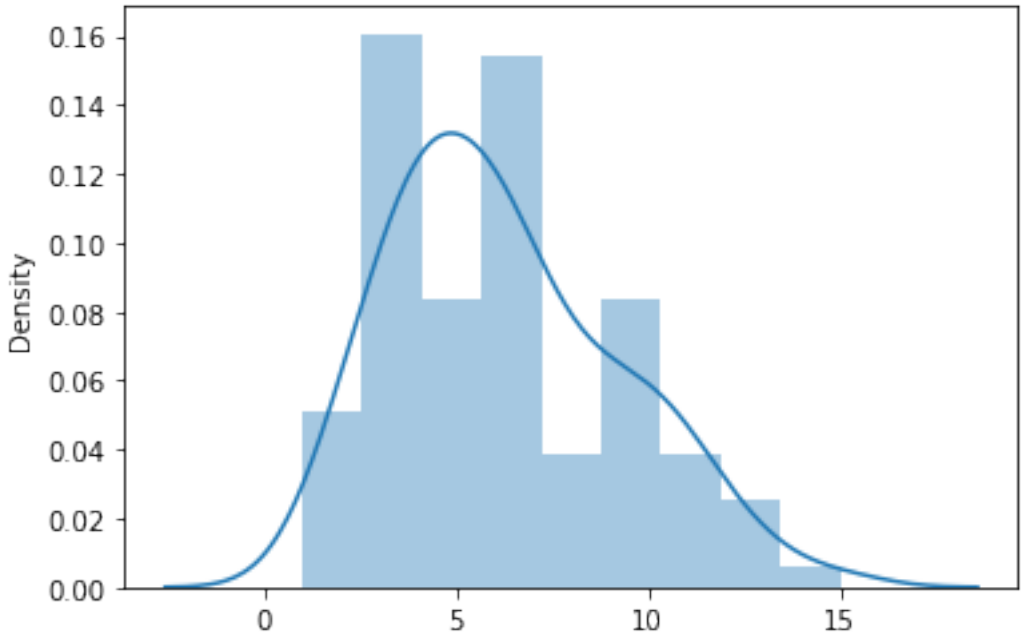



```
Lambda: 6
Expriment № 2
Chi-square: 9.984309890214421, G: 19.67513757268249, Pass: True
```

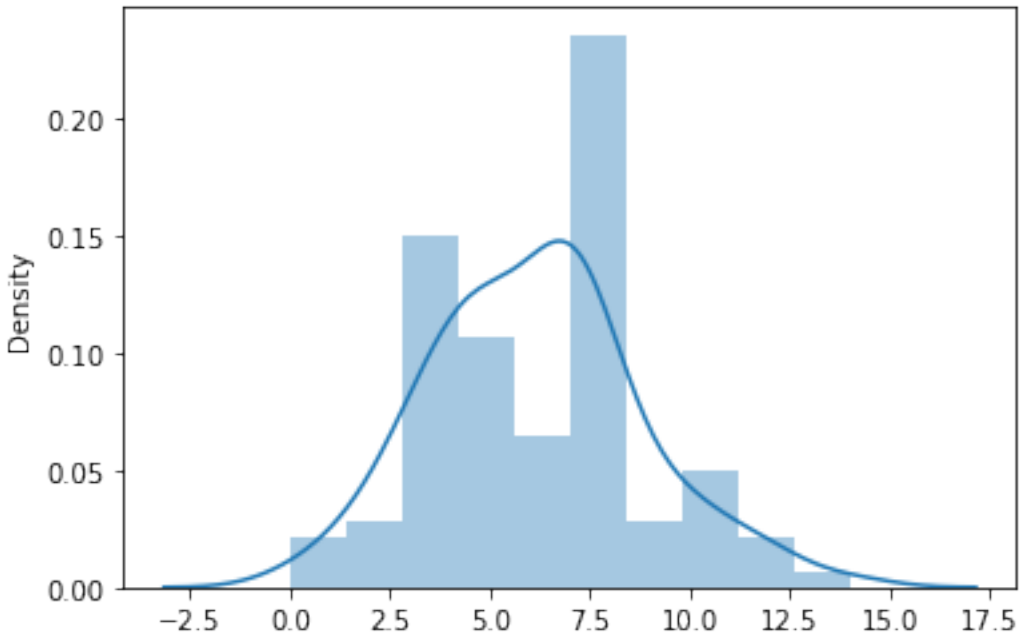



```
Lambda: 6
```

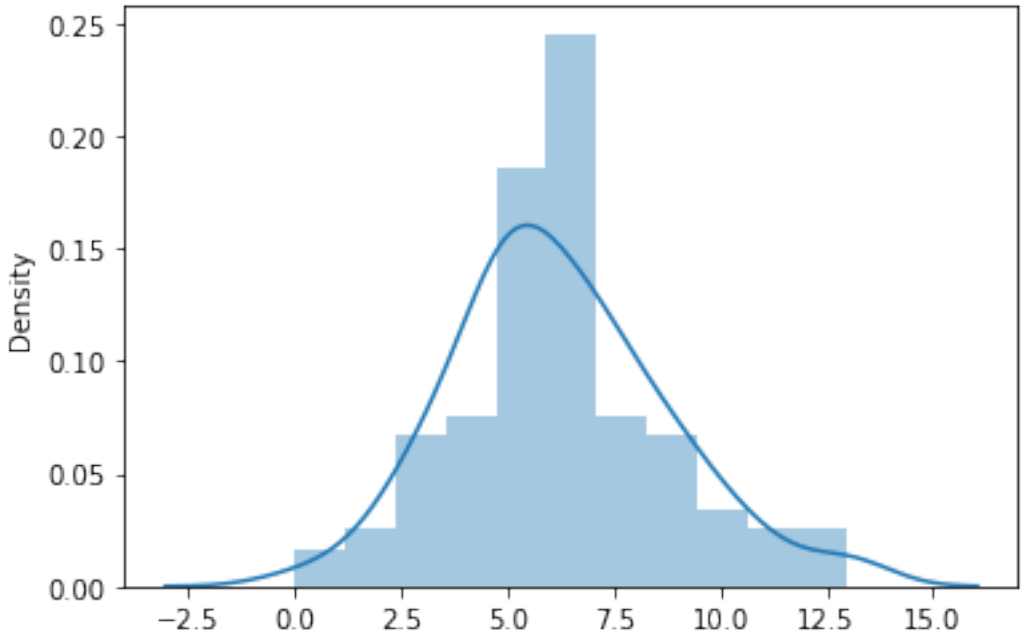
```
Expriment № 3
Chi-square: 27.087980273480227, G: 22.362032494826934, Pass: False
```



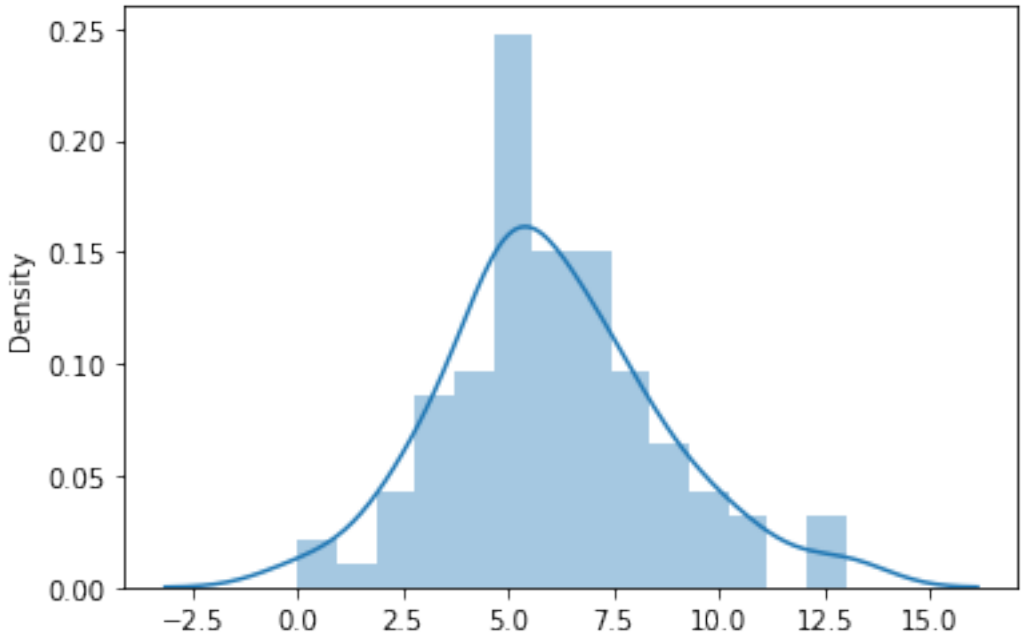

```
Lambda: 6
Expriment № 4
Chi-square: 19.236743839951956, G: 22.362032494826934, Pass: True
```



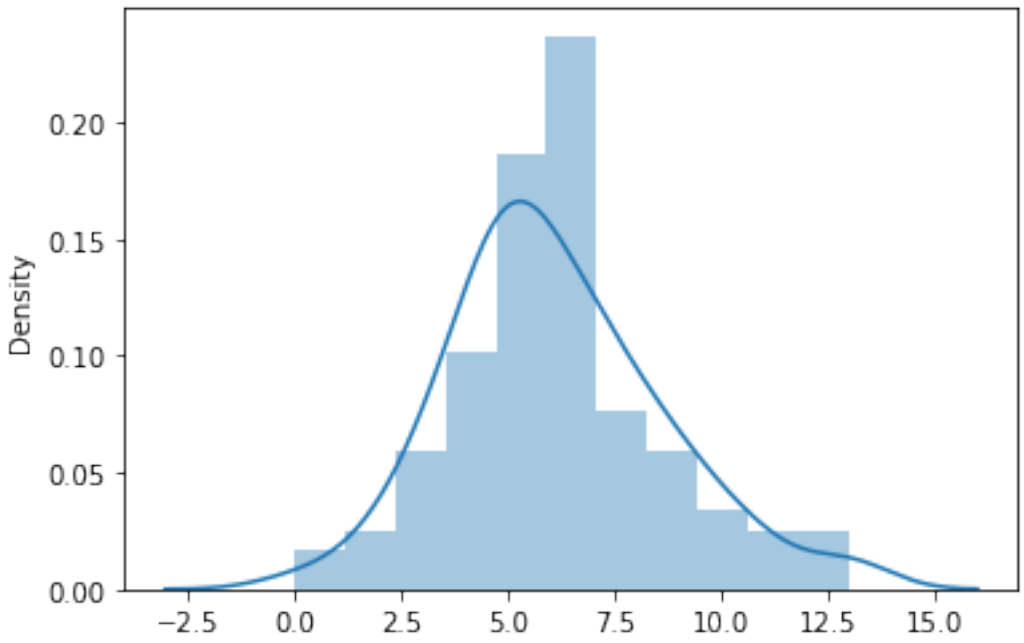
```
Lambda: 6
Expriment № 5
Chi-square: 19.457022482312052, G: 21.02606981748307, Pass: True
```




```
Lambda: 6
Expriment № 6
Chi-square: 29.756493172149728, G: 21.02606981748307, Pass: False
```

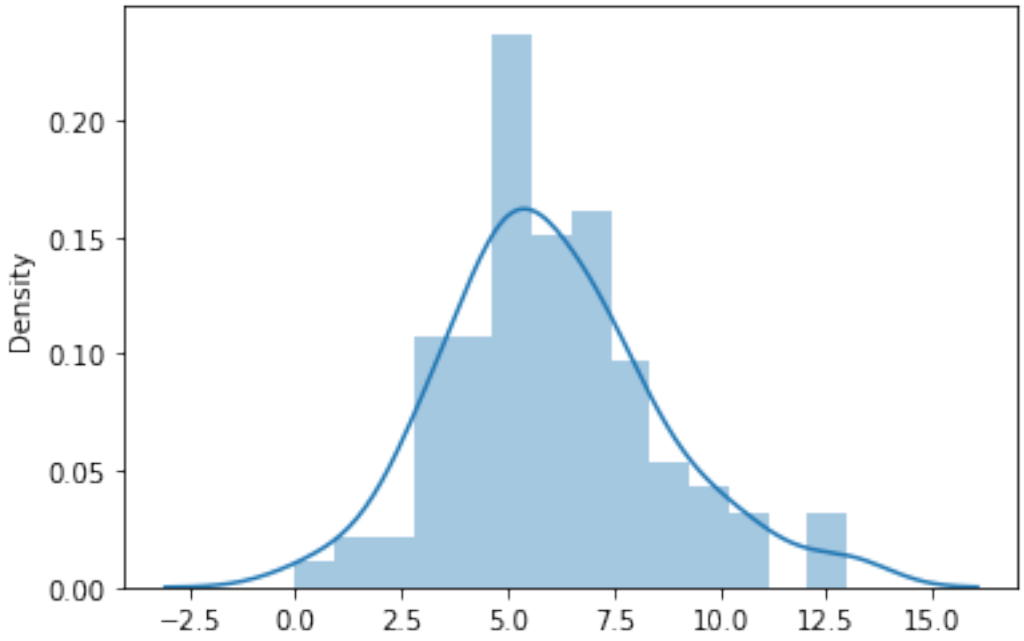



```
Lambda: 6
Expriment № 7
Chi-square: 18.041701303494943, G: 21.02606981748307, Pass: True
```

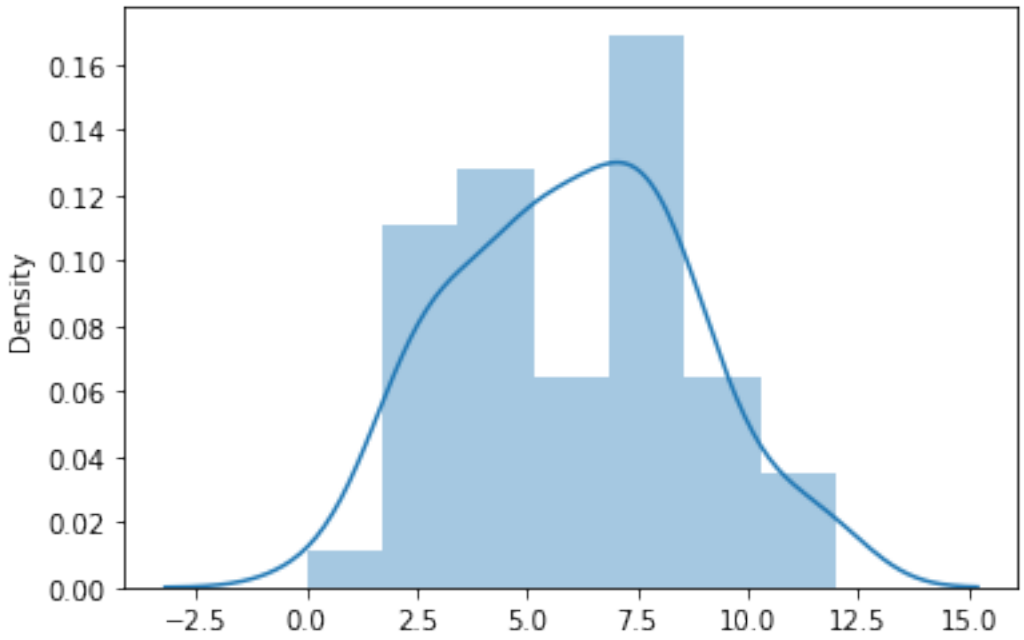



```
Lambda: 6
```

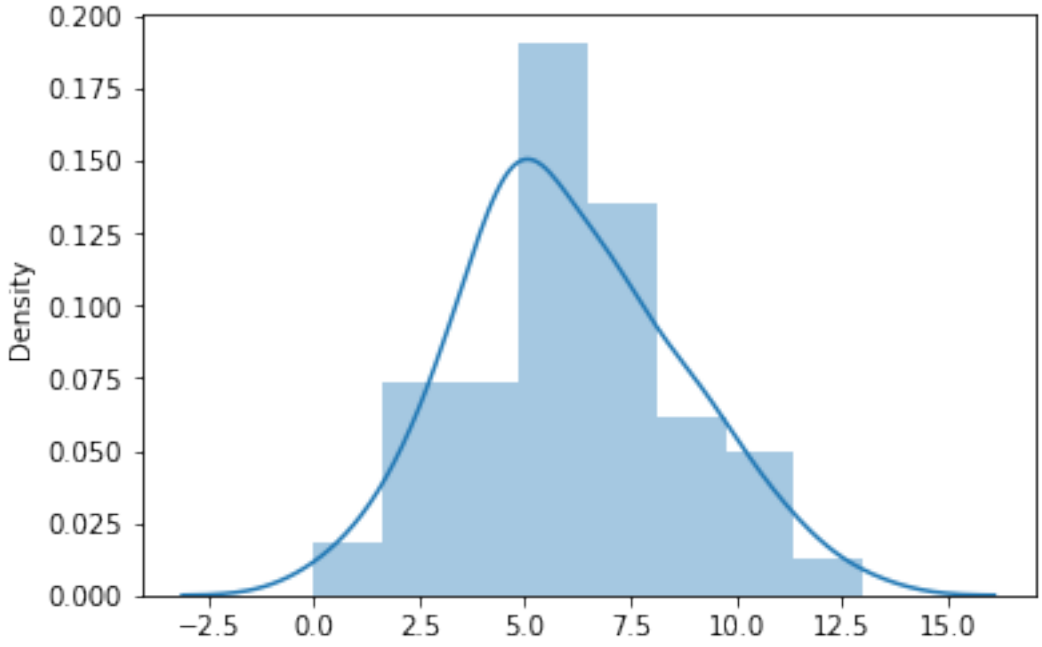
```
Expriment № 8
Chi-square: 20.14185531077194, G: 21.02606981748307, Pass: True
```



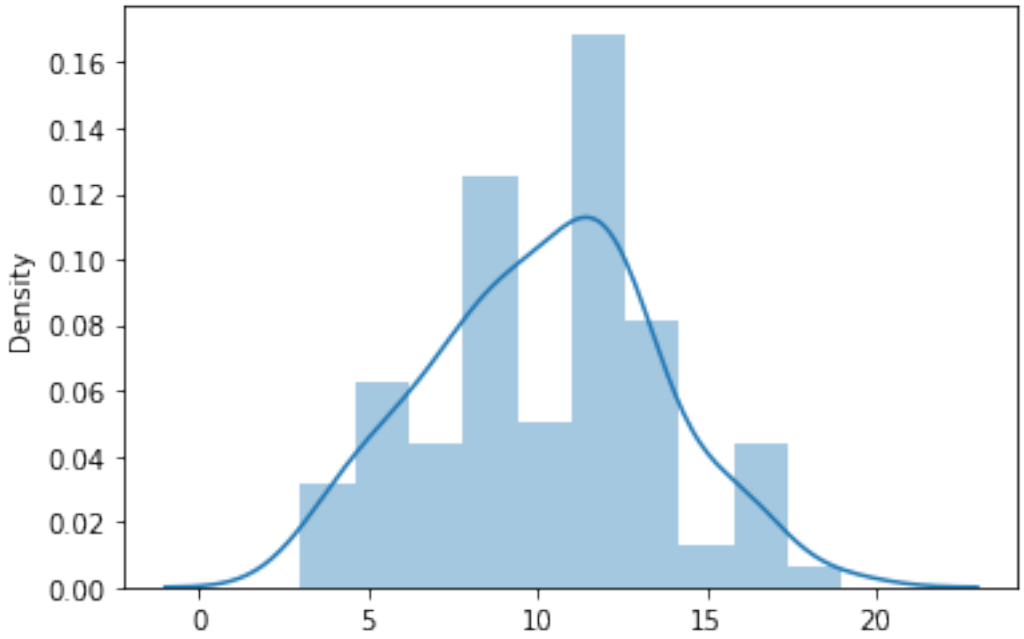

```
Lambda: 6
Expriment № 9
Chi-square: 15.731830828957431, G: 21.02606981748307, Pass: True
```



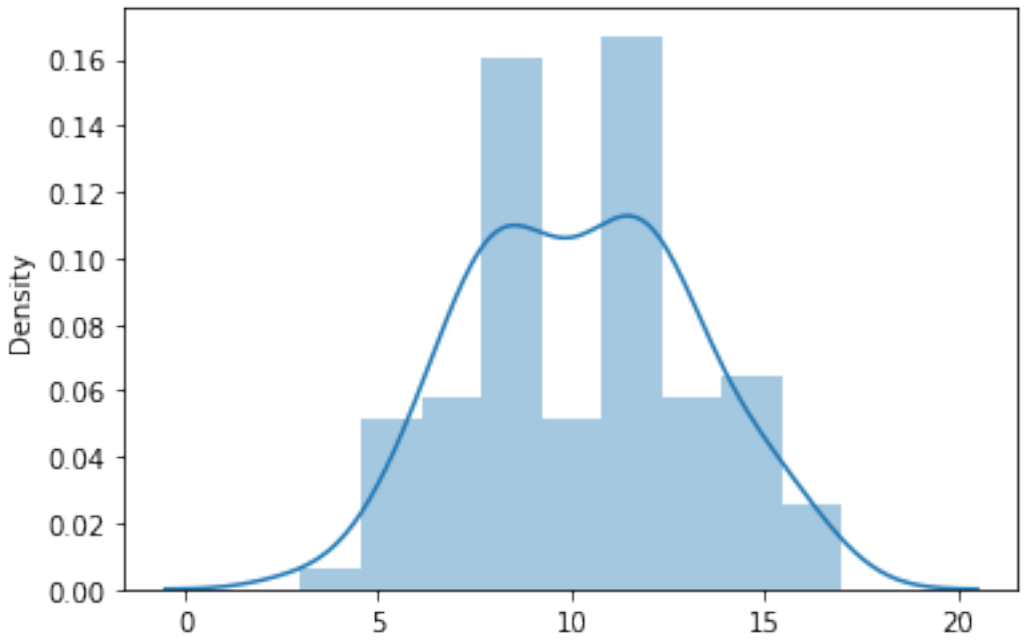
```
Lambda: 6
Expriment № 10
Chi-square: 13.447066617256409, G: 22.362032494826934, Pass: True
```




```
Lambda: 10
Expriment № 1
Chi-square: 21.068254737710657, G: 24.995790139728616, Pass: True
```

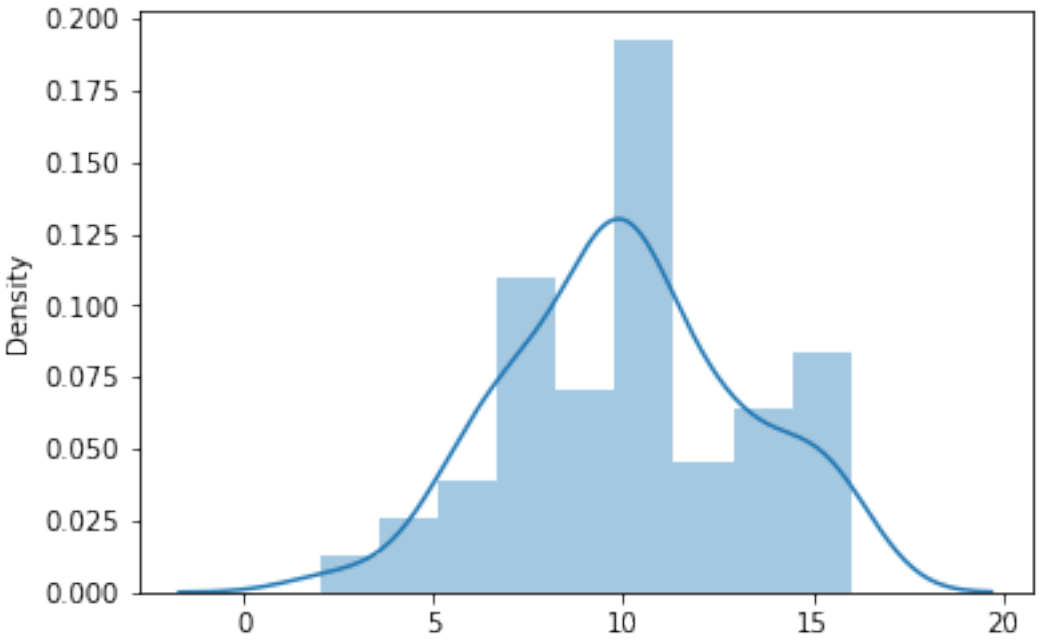



```
Lambda: 10
Expriment № 2
Chi-square: 6.637548437841705, G: 22.362032494826934, Pass: True
```

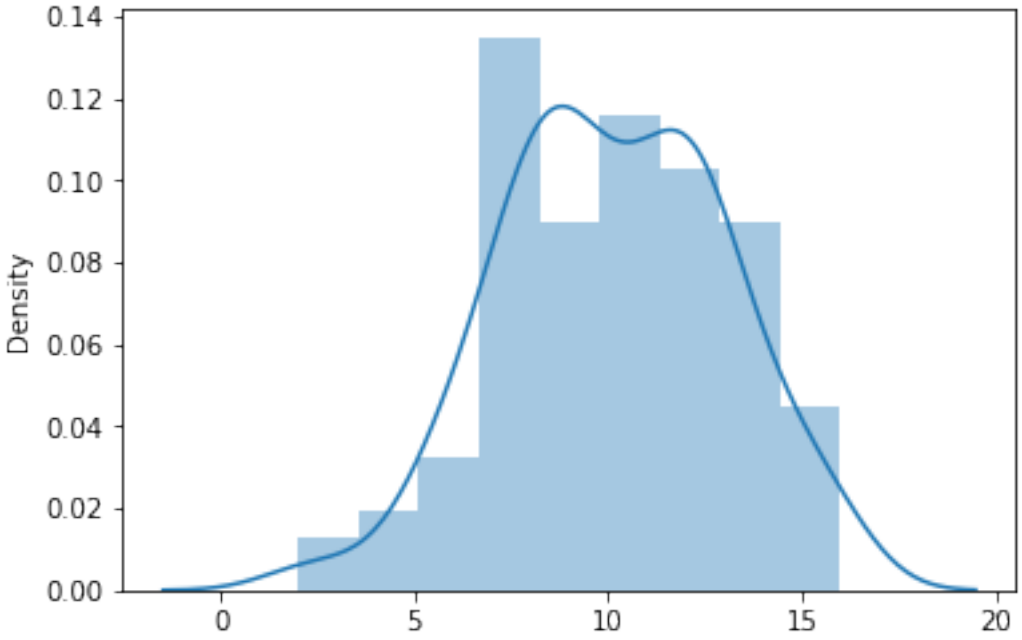



```
Lambda: 10
```

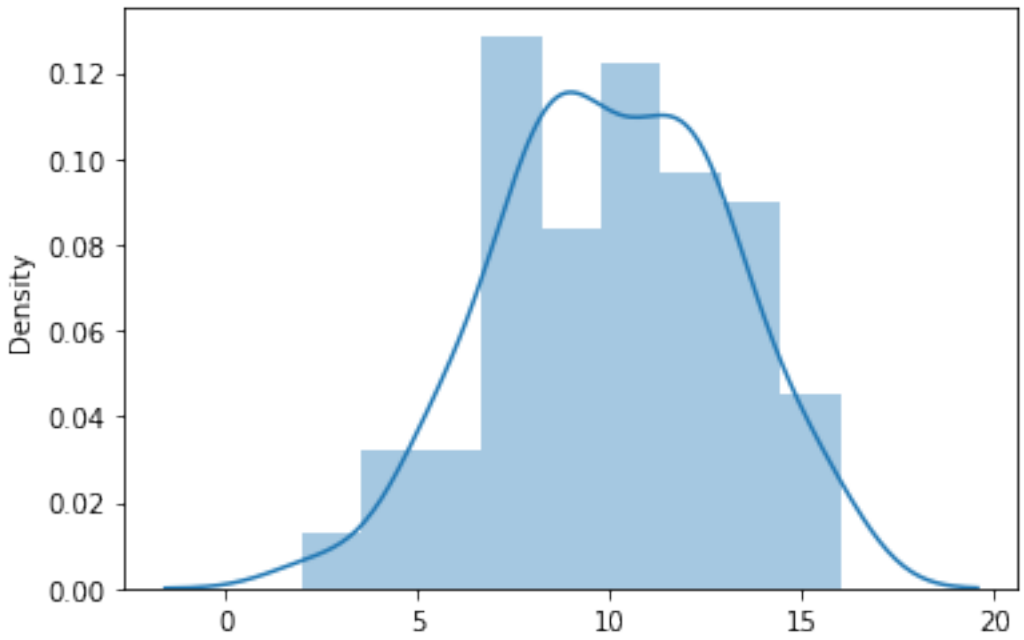
```
Expriment № 3
Chi-square: 23.93531507637661, G: 22.362032494826934, Pass: False
```



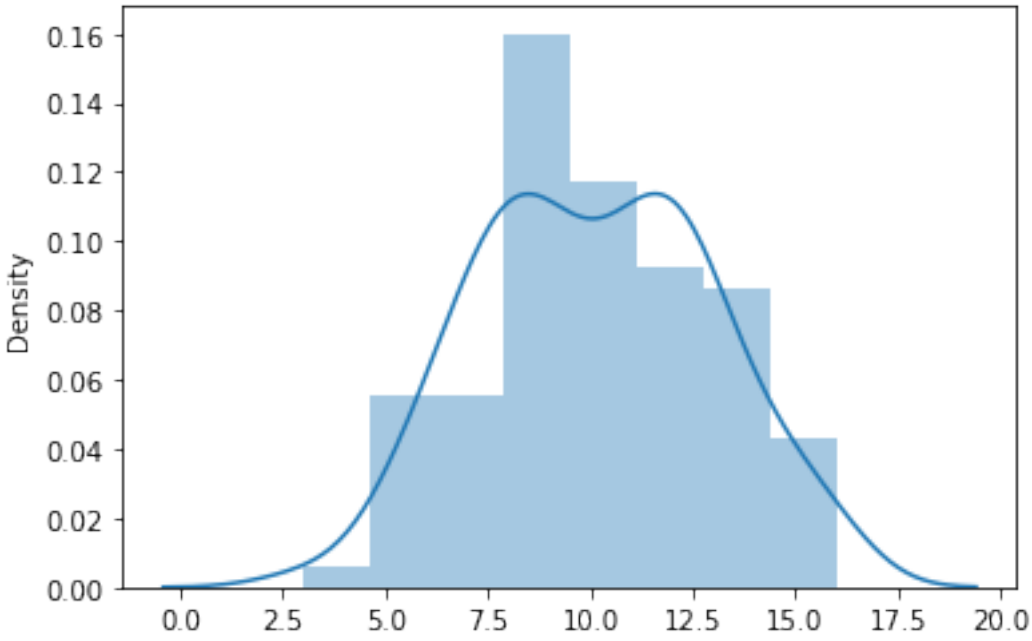

```
Lambda: 10
Expriment № 4
Chi-square: 11.210079722083421, G: 22.362032494826934, Pass: True
```



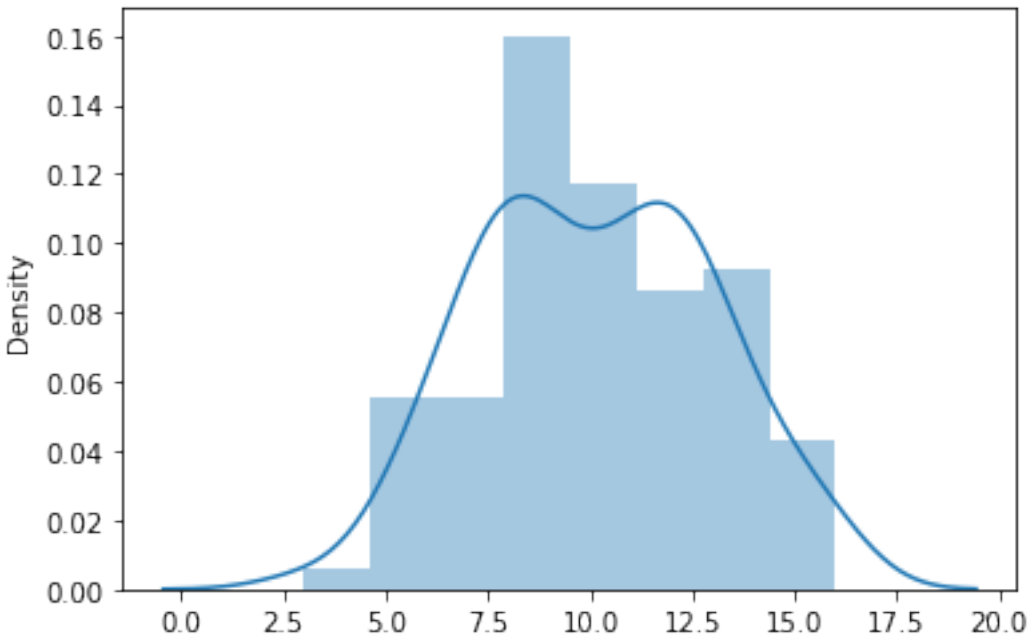
```
Lambda: 10
Expriment № 5
Chi-square: 10.085829172203134, G: 22.362032494826934, Pass: True
```




```
Lambda: 10
Expriment № 6
Chi-square: 6.605749600742023, G: 21.02606981748307, Pass: True
```

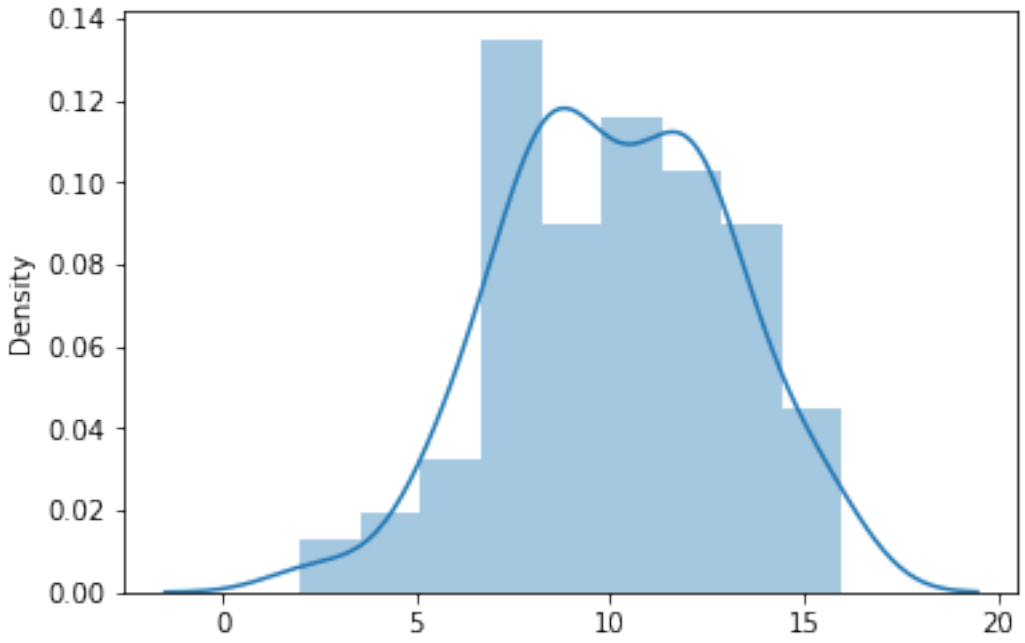



```
Lambda: 10
Expriment № 7
Chi-square: 6.889197863048225, G: 21.02606981748307, Pass: True
```

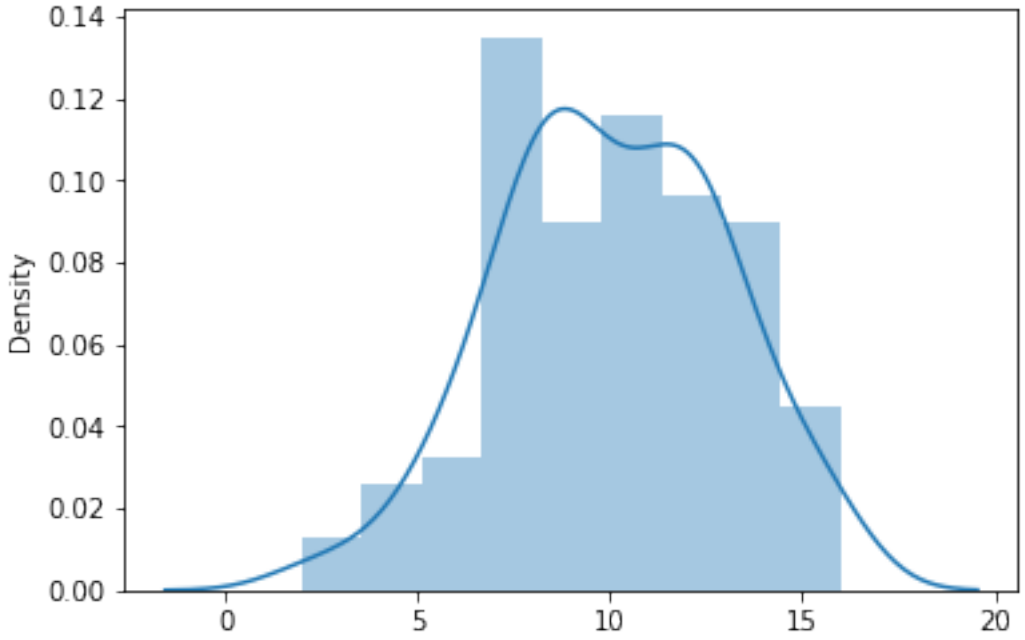



```
Lambda: 10
```

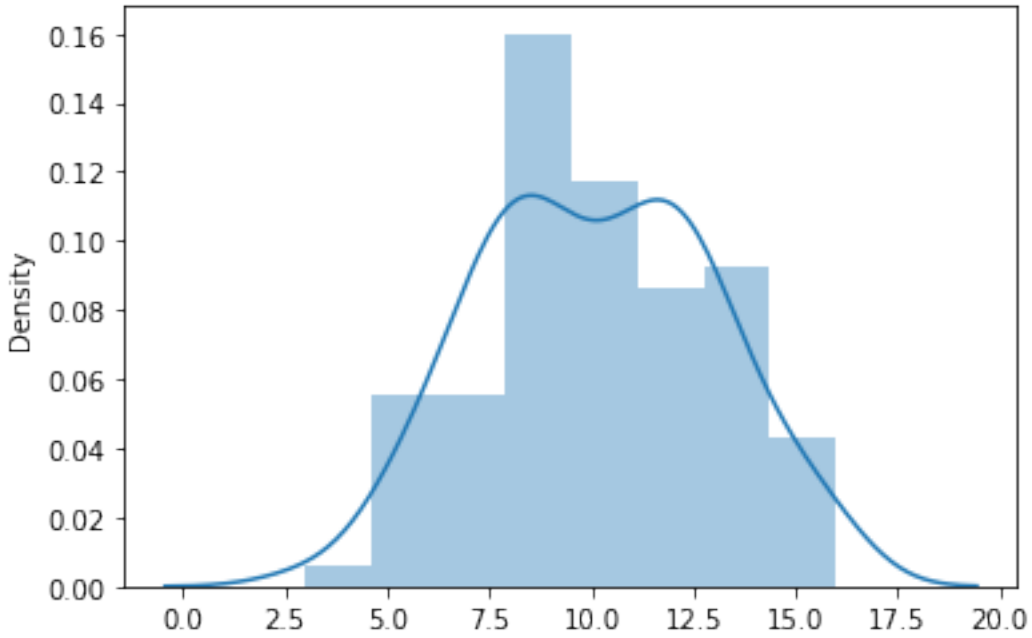
```
Expriment № 8
Chi-square: 11.210079722083421, G: 22.362032494826934, Pass: True
```




```
Lambda: 10
Expriment № 9
Chi-square: 10.359657810264434, G: 23.684791304840576, Pass: True
```



```
Lambda: 10
Expriment № 10
Chi-square: 6.257796005160614, G: 21.02606981748307, Pass: True
```



[8]:
```
for s in success_probas:
    success_probas[s] = sum(success_probas[s]) / len(success_probas[s])
```

[9]:
```
column_names = ['Lambda', 'Chi-square criteria passed probability']
df_by_text = pd.DataFrame(success_probas.items(), columns=column_names)
df_by_text
```

[9]:

	Lambda	Chi-square criteria passed probability
0	0.5	1.0
1	1.0	0.9
2	2.0	1.0
3	4.0	0.2
4	6.0	0.8
5	10.0	0.9

[ ]: