Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Лабораторная работа

СПОСОБЫ ПОЛУЧЕНИЯ СЛУЧАЙНЫХ ЧИСЕЛ С ЗАДАННЫМ ЗАКОНОМ РАСПРЕДЕЛЕНИЯ ЗАКОНОМ

Выполнил
студент гр. 3530904/00103 Плетнева А. Д.

Руководитель Чуркин В. В.

Санкт-Петербург
2023

Оглавление

Цель работы

1. Практическое освоение методов получения случайных величин, имеющих дискретный характер распределения.

2. Разработка программных датчиков дискретных случайных величин.

3. Исследование характеристик моделируемых датчиков:

3.1. Оценка точности моделирования: вычисление математического ожидания и дисперсии, сравнение полученных оценок с соответствующими теоретическими значениями.

4. Графическое представление функции плотности распределения и интегральной функции распределения.

Ход работы

Равномерное распределение

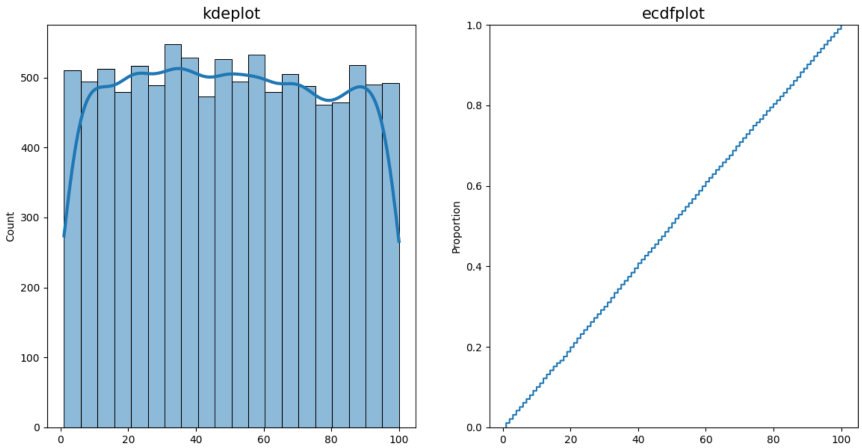
По следующей формуле получили 100000 случайных чисел от 1 до 100, округлив r до ближайшего целого:

```
r = (r_up - r_low + 1)*u + r_low
```

u - псевдослучайное вещественное число, равномерно распределенное в интервале [0,1], полученное с помощью программного датчика случайных чисел - np.random.random()
Посчитали для них математическое ожидание и дисперсию, сравнили с теоретическими значениями:

Оценка распределений	Эксперимент	Теоретическое значение	Отклонение
M	50.44497	50.5	0.05503000000000213
D	831.9030116990193	833.25	1.3469883009806836

Построили графики распределения и плотности распределения:



Биномиальное распределение

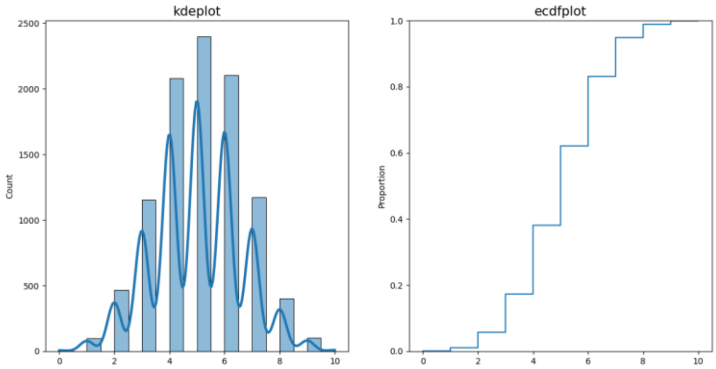
Простейшим случаем биномиального распределения является распределение Бернулли.
Получим 10000 случайных чисел, используя рекуррентный метод моделирования для биномиального закона:

```
p(0)=(1-p)**N,
... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ...
p(r)=p(r-1)*[((N-r)/(r+1))*(p/(1-p))],
```

Посчитали для полученных чисел с N=10 и p=0.5 математическое ожидание и дисперсию, сравнили с теоретическими значениями:

Оценка распределений	Эксперимент	Теоретическое значение	Отклонение
M	4.9984	5	0.0015999999999998238
D	2.4797974399998983	2.5	0.020202560000101677

Построили графики распределения и плотности распределения:



Геометрическое распределение

Для получения геометрически распределенной случайной величины использовали 3 алгоритма

1. Рекуррентный метод

```
p(0) = p
... ... ... ... ...
p(r) =p(r-1)*(1-p), где r=1,2, ... .
```

2. Прямой метод
   Последовательно получает случайные величины от 0 до 1 до тех пор, пока число не будет <=p, порядковый номер числа – искомое случайное число

```
    Используется специальный алгоритм: первый алгоритм может быть
усовершенствован следующим образом:

если положить       p[1]+p[2]+...+p[k] = u,          (2.3.5)

тогда               k = int[ln(u)/ln(q)]+1
```

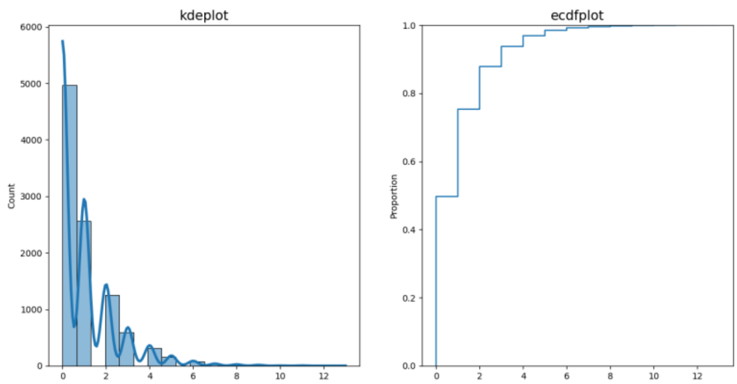
3.

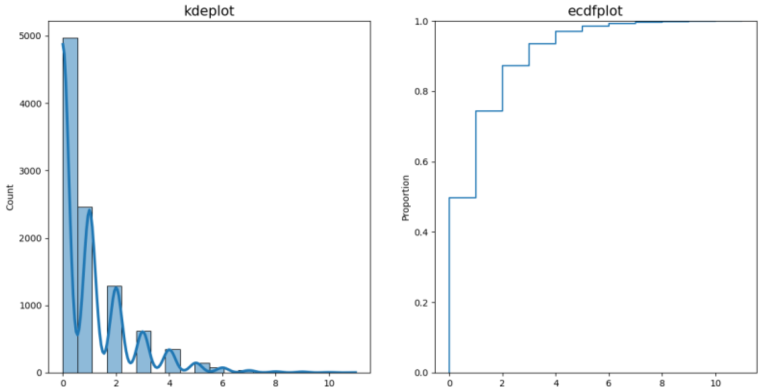
Для данных алгоритмов получаем следующие результаты:

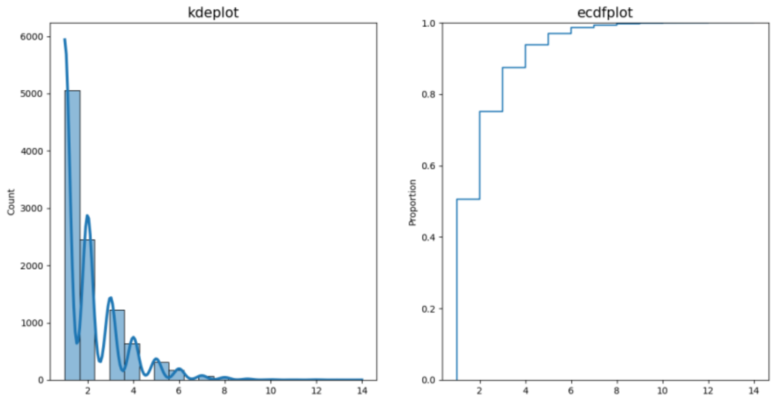
Оценка распределений	Эксперимент1	Эксперимент2	Эксперимент3	Теоретическое значение	Отклонение1	Отклонение2	Отклонение3
M	0.9889	0.9938	2.0134	2.0	1.0110999999999999	1.0062	0.013399999999999985
D	1.9347767900002706	1.9765615600000277	2.009020439999986	2.0	0.06522320999972941	0.023438439999972305	0.00902043999998580

Согласно результатам, M и D 3 алгоритма наиболее близки к теоретическим.

Построили графики распределения и плотности распределения для всех 3 алгоритмов:







Распределение Пуассона

Для получения случайных чисел, имеющих распределение Пуассона, использовали 2 алгоритма

1. Рекуррентный метод

```
p(0)   =  e **(-mu)
... ... ... ... ... ... ... ... ... ...
P(r) = P(r-1)*mu/r  для  r = 1,2,... ,
```

2. Специальный алгоритм - заключается в перемножении равномерно распределенных случайных чисел до тех пор, пока выполняется условие:

```
Ro             -mu
 П x[i] >= e
i=0
```

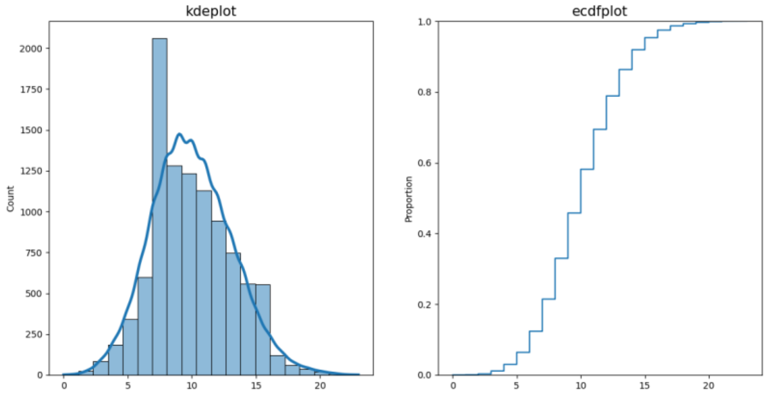
А в случае, если mu>88 используем нормальную аппроксимацию

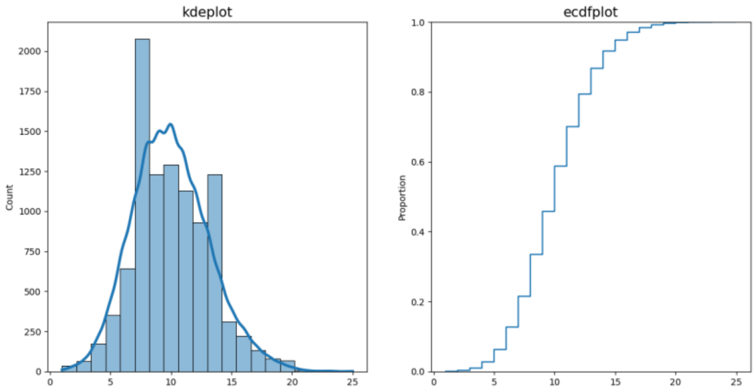
Для данных алгоритмов получаем следующие результаты:

Оценка распределений	Эксперимент1	Эксперимент2	Теоретическое значение	Отклонение1	Отклонение2
M	10.0172	10.0057	10.0	0.01720000000000077	0.005699999999999915
D	9.82910415999988	10.004467510000042	10.0	0.1708958400001208	0.004467510000042196

Согласно результатам, M и D 2 алгоритма наиболее близки к теоретическим.

Построили графики распределения и плотности распределения для обоих алгоритмов:

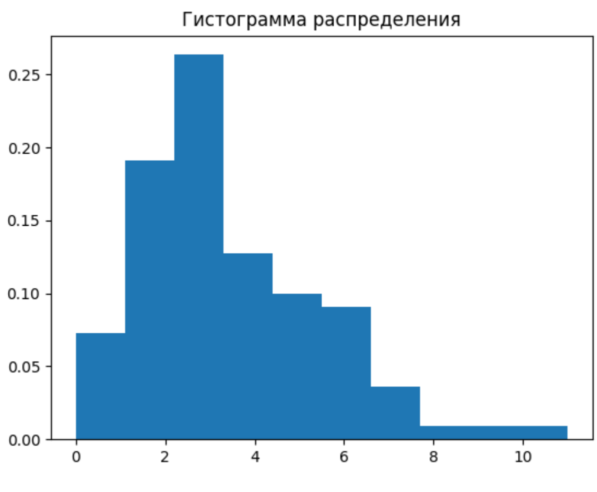




Индивидуальное задание

4. Смоделировать случайную величину X, имеющую закон Пуассона с параметром λ = 4. На основе выборки объема 100 построить гистограмму распределения, провести проверку согласия эмпирического распределения теоретическому критерием Пирсона с уровнем значимости 0,05

Сгенерируем выборку объемом 100, с помощью функции np.random.poisson(lam, n) и построим гистограмму распределения:



Разделим выборку на 10 интервалов, вы числим сначала эмпирическое количество наблюдений в каждом интервале с помощью observed = np.histogram(sample, bins=10), а

затем теоретическое np.array([len(sample) * (poisson.cdf(bins[i+1], lam) - poisson.cdf(bins[i], lam)) for i in range(10)])
Получим:

```
Эмпирическая:  [ 8 21 29 14 11 10  4  1  1  1]
Теоретическая:  [ 7.32625556 14.65251111 19.53668148 19.53668148 15.62934519 10.41956346
  5.95403626  2.97701813  1.32311917  0.72170136]
```

Далее по формуле найдем значение критерия Пирсона:

$$u = \chi^2 = \sum_{i=1}^{l} \frac{(n_i - np_i)^2}{np_i}$$

А также найдем критическое значение chi2.isf(0.05, 9, loc=0, scale=1)
Получаем:

```
Значение хи-квадрат: 12.493228285754345
критическое значение: 16.918977604620448
```

Нет оснований отвергнуть нулевую гипотезу о том, что выборка соответствует распределению Пуассона с заданным параметром, на уровне значимости α=0.05, так как значение хи-квадрат (χ2) не превышает критическое значение при заданном числе степеней свободы.

Текст программы

Биномиальное распределение

```
import math

import numpy as np
from prettytable import PrettyTable

from Lab_2.get_plots import get_plots


def binomial_dist(n, p, N):
    r_array = []
    D = 0
    for i in range(n):
        a = np.random.random()
        p_r = (1 - p) ** N
        m = 0
        while (a - p_r) >= 0:
            a -= p_r
            p_r *= ((N - m) / (m + 1)) / (p / (1 - p))
            m += 1
        r_array.append(m)
    M = sum(r_array) / n
    for i in range(n):
        D += (r_array[i] - M) ** 2
    D /= n
    return r_array, M, D


n = 10000
p = 0.5
N = 10
array, M, D = binomial_dist(n, p, N)
M_theor = 5
D_theor = 2.5
table = PrettyTable(['Оценка распределений', 'Эксперимент', 'Теоретическое 
значение', 'Отклонение'])
table.add_row(["M", M, M_theor, abs(M - M_theor)])
table.add_row(["D", D, D_theor, abs(D - D_theor)])
print(table)

get_plots(array)
```

Геометрическое распределение

```
import math

import numpy as np
from prettytable import PrettyTable

from Lab_2.get_plots import get_plots


def geom_dist_alg_1(n, p):
    r_array = []
    D = 0
    for i in range(n):
        a = np.random.random()
        p_r = p
        m = 0
        while (a - p_r) >= 0:
            a -= p_r
```

```
            p_r *= (1 - p)
            m += 1
        r_array.append(m)
    M = sum(r_array) / n
    for i in range(n):
        D += (r_array[i] - M) ** 2
    D /= n
    return r_array, M, D


def geom_dist_alg_2(n, p):
    r_array = []
    D = 0
    for i in range(n):
        a = np.random.random()
        m = 0
        while a > p:
            a = np.random.random()
            m += 1
        r_array.append(m)
    M = sum(r_array) / n
    for i in range(n):
        D += (r_array[i] - M) ** 2
    D /= n
    return r_array, M, D


def geom_dist_alg_3(n, p):
    r_array = []
    D = 0
    for i in range(n):
        a = np.random.random()
        m = int(math.log(a) / math.log(1 - p)) + 1
        r_array.append(m)
    M = sum(r_array) / n
    for i in range(n):
        D += (r_array[i] - M) ** 2
    D /= n
    return r_array, M, D


n = 10000
p = 0.5
array_1, M_1, D_1 = geom_dist_alg_1(n, p)
array_2, M_2, D_2 = geom_dist_alg_2(n, p)
array_3, M_3, D_3 = geom_dist_alg_3(n, p)
M_theor = 2.0
D_theor = 2.0
table = PrettyTable(
    ['Оценка распределений', 'Эксперимент1', 'Эксперимент2', 'Эксперимент3',
'Теоретическое значение', 'Отклонение1',
     'Отклонение2', 'Отклонение3'])
table.add_row(["M", M_1, M_2, M_3, M_theor, abs(M_1 - M_theor), abs(M_2 -
M_theor), abs(M_3 - M_theor)])
table.add_row(["D", D_1, D_2, D_3, D_theor, abs(D_1 - D_theor), abs(D_2 -
D_theor), abs(D_3 - D_theor)])
print(table)

get_plots(array_1)
get_plots(array_2)
get_plots(array_3)
```

Построение графиков

```
from matplotlib import pyplot as plt
import seaborn as sns


def get_plots(array):
    fig = plt.figure()

    ax_1 = fig.add_subplot(1, 2, 1)
    ax_2 = fig.add_subplot(1, 2, 2)

    sns.histplot(
        ax=ax_1,
        data=array,
        kde=True,
        bins=20,
        line_kws={"lw": 3})
    sns.ecdfplot(
        ax=ax_2,
        data=array)

    ax_1.set_title('kdeplot',
                   fontsize=15)
    ax_2.set_title('ecdfplot',
                   fontsize=15)

    fig.set_figwidth(14)
    fig.set_figheight(7)

    plt.show()
```

Распределение Пуассона

```
import math

import numpy as np
from prettytable import PrettyTable


from Lab_2.get_plots import get_plots


def puasson_dist_alg_1(n, mu):
    r_array = []
    D = 0
    if mu >= 88:
        r_array = np.random.normal(mu, mu, n)
    else:
        for i in range(n):
            a = np.random.random()
            p_r = math.exp(-mu)
            m = 0
            while (a - p_r) >= 0:
                a -= p_r
                m += 1
                p_r *= mu / m
            r_array.append(m)
    M = sum(r_array) / n
    for i in range(n):
        D += (r_array[i] - M) ** 2
    D /= n
    return r_array, M, D


def puasson_dist_alg_2(n, mu):
    r_array = []
    D = 0
```

```
    if mu >= 88:
        r_array = np.random.normal(mu, mu, n)
    else:
        for i in range(n):
            a = np.random.random()
            m = 0
            while a >= math.exp(-mu):
                a *= np.random.random()
                m += 1
            r_array.append(m)
    M = sum(r_array) / n
    for i in range(n):
        D += (r_array[i] - M) ** 2
    D /= n
    return r_array, M, D


n = 10000
mu = 10
array_1, M_1, D_1 = puasson_dist_alg_1(n, mu)
array_2, M_2, D_2 = puasson_dist_alg_2(n, mu)
M_theor = 10.0
D_theor = 10.0
table = PrettyTable(
    ['Оценка распределений', 'Эксперимент1', 'Эксперимент2', 'Теоретическое значение', 'Отклонение1', 'Отклонение2'])
table.add_row(["M", M_1, M_2, M_theor, abs(M_1 - M_theor), abs(M_2 - M_theor)])
table.add_row(["D", D_1, D_2, D_theor, abs(D_1 - D_theor), abs(D_2 - D_theor)])
print(table)

get_plots(array_1)
get_plots(array_2)
```

Равномерное распределение

```
import math

import numpy as np
from prettytable import PrettyTable

from Lab_2.get_plots import get_plots


def uniform_dist(n, r_low, r_up):
    r_array = []
    D = 0
    for i in range(n):
        r = math.floor((r_up - r_low + 1) * np.random.random() + r_low)
        r_array.append(r)
    M = sum(r_array)/n
    for i in range(n):
        D += (r_array[i] - M) ** 2
    D /= n
    return r_array, M, D


n = 10000
r_low = 1
r_up = 100
array, M, D = uniform_dist(n, r_low, r_up)
M_theor = (r_low + r_up) / 2
D_theor = ((r_up - r_low + 1) ** 2 - 1) / 12
table = PrettyTable(['Оценка распределений', 'Эксперимент', 'Теоретическое значение', 'Отклонение'])
```

```
table.add_row(["M", M, M_theor, abs(M - M_theor)])
table.add_row(["D", D, D_theor, abs(D - D_theor)])
print(table)

get_plots(array)
```

Индивидуальное задание

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson, chi2

lam = 4
n = 100
sample = np.random.poisson(lam, n)

plt.hist(sample, bins=10, density=True)
plt.title('Гистограмма распределения')
plt.show()

observed, bins = np.histogram(sample, bins=10)
print("Эмпирическая: ", observed)
expected = np.array([len(sample) * (poisson.cdf(bins[i+1], lam) -
poisson.cdf(bins[i], lam)) for i in range(10)])
print("Теоретическая: ", expected)

hi2 = np.sum((observed - expected) ** 2 / expected)
critical_value = chi2.isf(0.05, 9, loc=0, scale=1)

print('Значение хи-квадрат:', hi2)
print("критическое значение:", critical_value)
```