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

Отчет

по лабораторной работе №2 Способы получения случайных чисел с заданным законом распределения

по дисциплине «Введение в машинное обучение»

Выполнила
Студентка гр. 5130904/10101 Никифорова Е. А.

Руководитель Чуркин В. В.

Санкт-Петербург
2024

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

1. Написать и отладить подпрограммы получения дискретных псевдослучайных чисел в соответствии с алгоритмами, приведенными в описании.
2. Осуществить проверку точности моделирования полученных датчиков псевдослучайных чисел.

Равномерное распределение

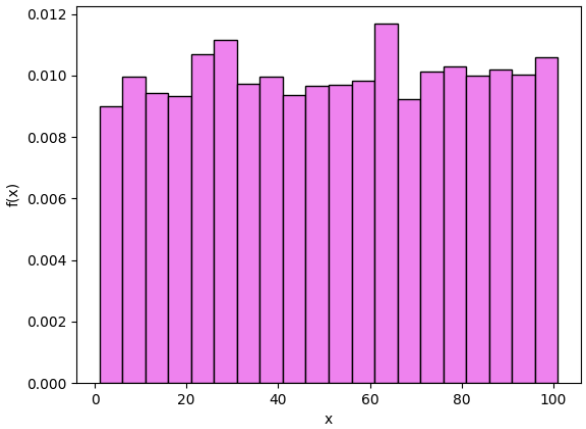

Рисунок 1 Плотность распределения

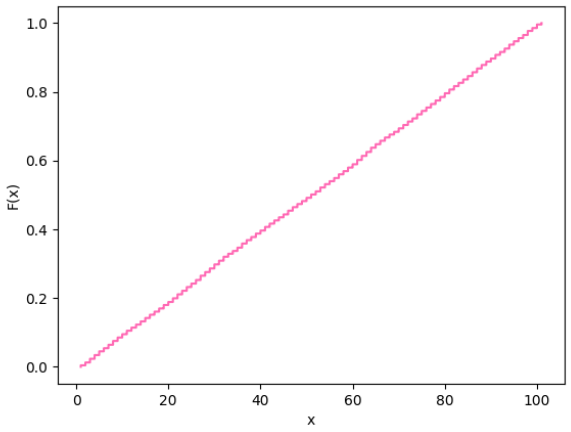

Рисунок 2 Функция распределения

- Мат. ож. 51.0556
- Дисперсия 828.6567086399982
- Погрешность для мат. ож. 0.5555999999999983
- Погрешность для дисперсии -4.5932913600017855

Биномиальное распределение

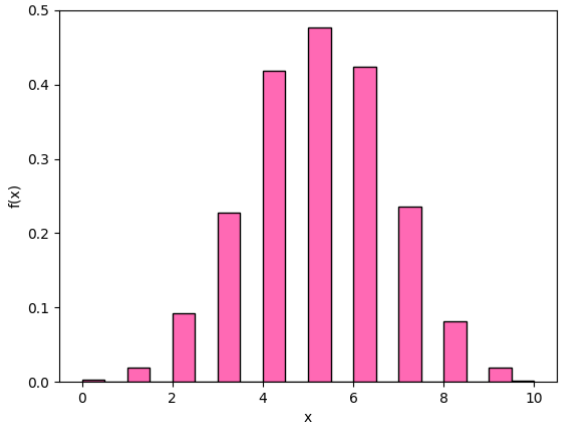

Рисунок 3 Плотность распределения

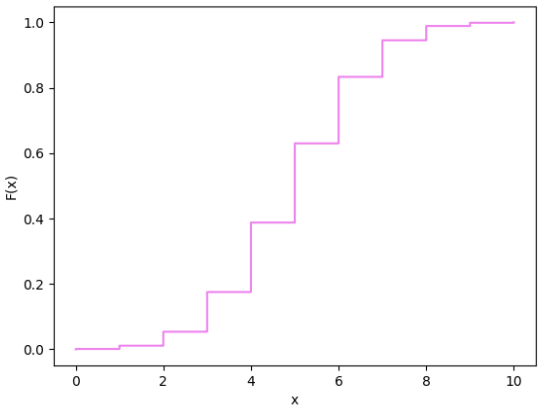

Рисунок 4 Функция распределения

- Мат. ож. 4.9758
- Дисперсия 2.5010143599999224
- Погрешность для мат. ож. -0.024200000000000443
- Погрешность для дисперсии 0.0010143599999223873

Геометрическое распределение

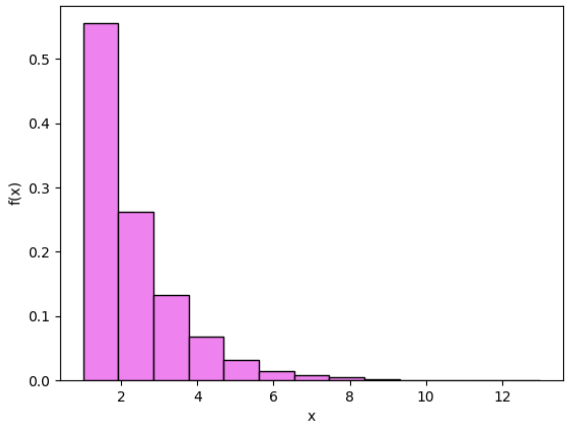

Рисунок 5 Плотность распределения алгоритм 1

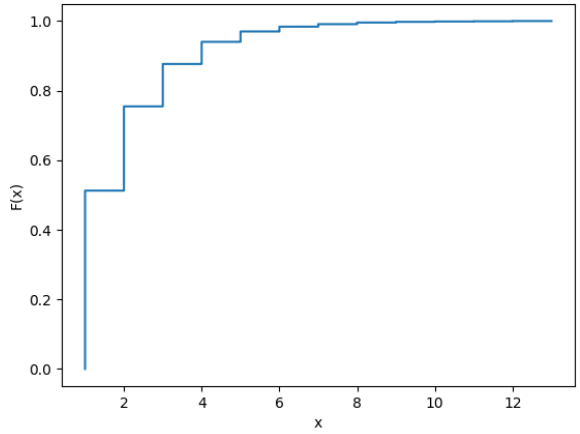

Рисунок 6 Функция распределения алгоритм 1

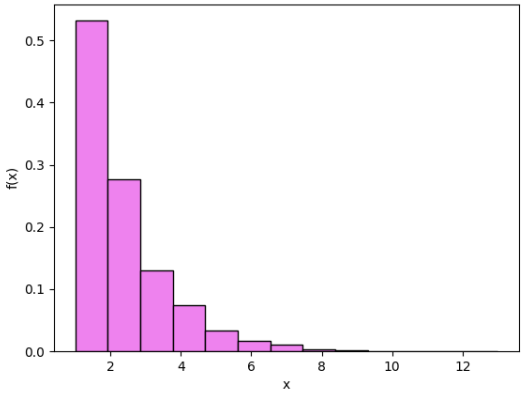

Рисунок 7 Плотность распределения алгоритм 2

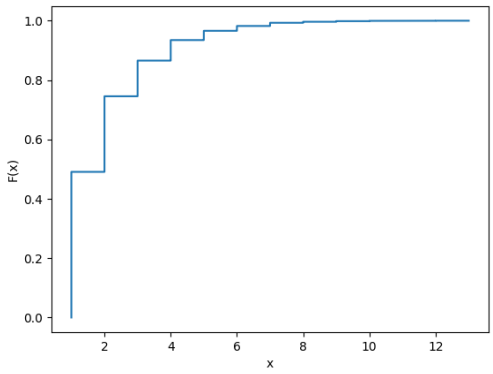

Рисунок 8 Функция распределения алгоритм 2

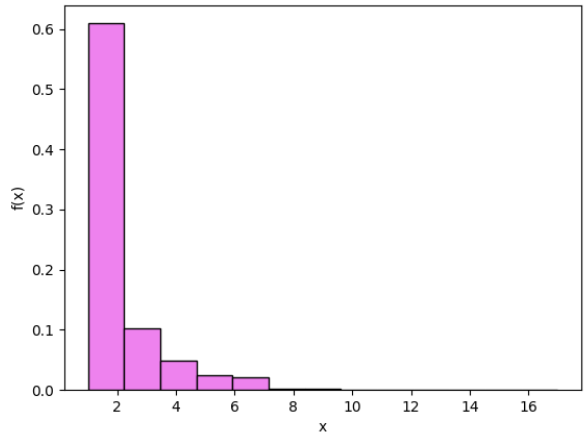

Рисунок 9 Плотность распределения алгоритм 3

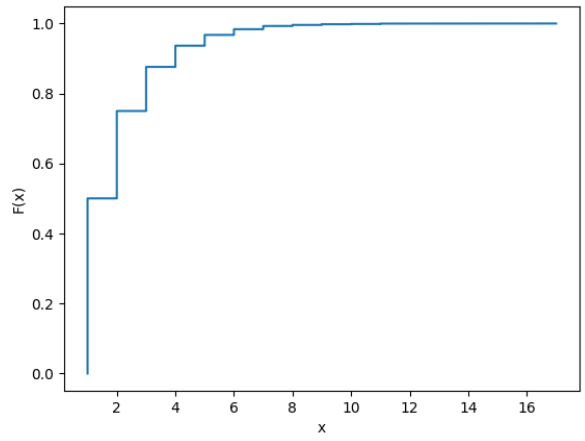

Рисунок 10 Функция распределения алгоритм 3

Алгоритм 1:

- Мат. ож. 2.0102
- Дисперсия 2.0072959599998565
- Погрешность для мат. ож. 0.010200000000000209
- Погрешность для дисперсии -0.19270404000014363

Алгоритм 2:

- Мат. ож. 2.0288
- Дисперсия 2.0387705599997044
- Погрешность для мат. ож. 0.028799999999999937
- Погрешность для дисперсии -0.16122944000029582

Алгоритм 3:

- Мат. ож. 2.0025
- Дисперсия 2.029493750000067
- Погрешность для мат. ож. 0.0024999999999999467
- Погрешность для дисперсии -0.17050624999993325

Пуассоновское распределение

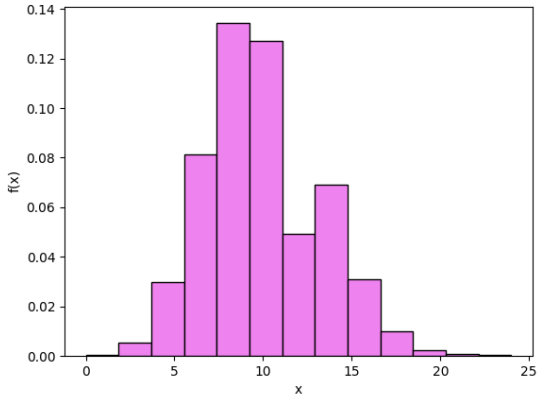

Рисунок 11 Плотность распределения алгоритм 1

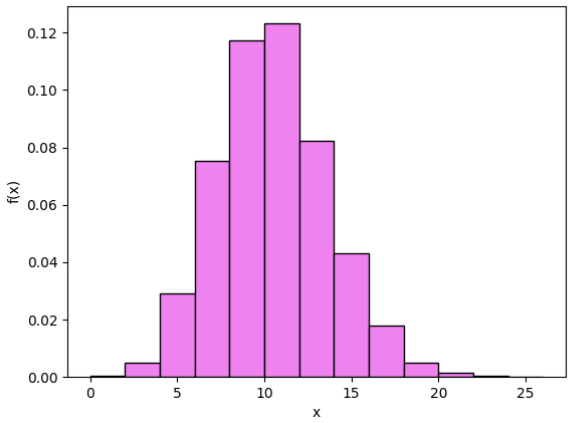

Рисунок 13 Плотность распределения алгоритм 2

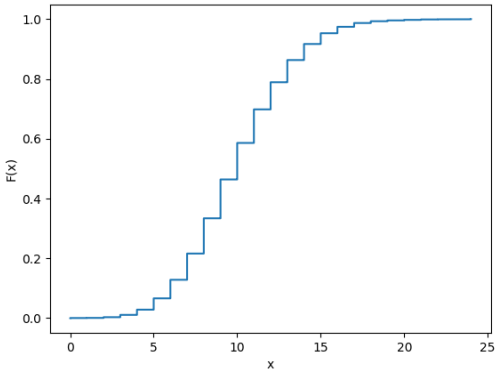

Рисунок 12 Функция распределения алгоритм 1

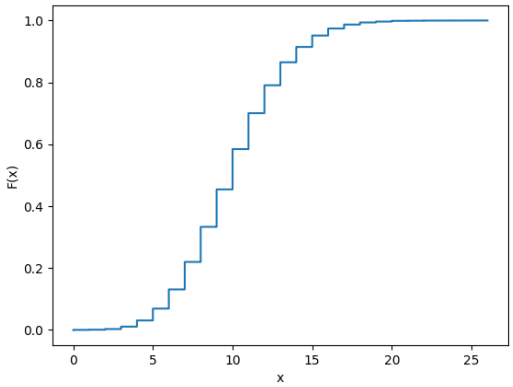

Рисунок 14 Функция распределения алгоритм 2

Алгоритм 1:

- Мат. ож. 9.9976
- Дисперсия 9.964394240000608
- Погрешность для мат. ож. -0.0023999999999997357
- Погрешность для дисперсии -0.03560575999939175

Алгоритм 2:

- Мат. ож. 9.9942
- Дисперсия 10.004766359999845
- Погрешность для мат. ож. -0.0058000000000006935
- Погрешность для дисперсии 0.004766359999845093

Индивидуальное задание

Цель работы

8. Смоделировать случайную величину $\boldsymbol{X}$, имеющую геометрический закон распределения с параметром p=0.7. На основе выборки объема 50 провести проверку согласия эмпирического распределения теоретическому критерием Пирсона с уровнем значимости 0,05.

Результаты

Статистика хи-квадрат: 4.5053

Критическое значение для статистики хи-квадрат на уровне значимости 0.05: 9.4877

p-значение: 0.3419

Степень свободы: 4

Нет оснований отвергнуть гипотезу о соответствии на уровне значимости 0.05.

Вывод

В результате выполненной лабораторной работы были разработаны программные модули для генерации дискретных случайных величин. Для каждого модуля были вычислены характеристики распределения - математическое ожидание и дисперсия, их сопоставлено с теоретическими значениями. Кроме того, были построены графики, отражающие функцию плотности вероятности и функцию распределения, что дало возможность визуально оценить их соответствие теоретическим представлениям.

Приложение

```
import random
import matplotlib.pyplot as plt
from statsmodels.distributions.empirical_distribution import ECDF


def uniform_distribution(low_value: float, up_value: float) -> int:
    u = random.random()
    return int(round((up_value - low_value + 1) * u + low_value, 0))


def show_density_distribution_function(arr: []) -> None:
    bins_num = 3
    if len(arr) > 10:
        bins_num = 20
    plt.hist(arr, bins=bins_num,
             color='violet', edgecolor='black', density=True)
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.show()


def show_distribution_function(arr: []) -> None:
    ecdf = ECDF(arr)
    plt.step(ecdf.x, ecdf.y, color="hotpink")
    plt.xlabel("x")
    plt.ylabel("F(x)")
    plt.show()


if __name__ == '__main__':
    low_value = 1
    up_value = 100
    n = 10 ** 4

    values = [uniform_distribution(low_value, up_value) for _ in range(n)]

    mat_og = sum(values) / n
    dispersion = sum([(value - mat_og) ** 2 for value in values]) / n

    print(f"Мат. ож. {mat_og}")
    print(f"Дисперсия {dispersion}")
    print(f"Погрешность для мат. ож. {mat_og - 50.5}")
    print(f"Погрешность для дисперсии {dispersion - 833.25}")

    show_density_distribution_function(values)
    show_distribution_function(values)
```

```
import random
import matplotlib.pyplot as plt
from statsmodels.distributions.empirical_distribution import ECDF
import numpy as np


def binomial_distribution(n: int, p: float) -> int:
    result_m = 0
    if n < 100:
        value = random.random()
        p_func = (1 - p) ** n
        while value >= p_func:
            value = value - p_func
            p_func = p_func * ((p * (n - result_m)) / ((result_m + 1) * (1 - p)))
            result_m = result_m + 1
    else:
        result_m = round(np.random.normal(loc=(n * p), scale=np.sqrt(n * p * (1 - p))))
    return result_m


def show_density_distribution_function(arr: []) -> None:
    bins_num = 20
    plt.hist(arr, bins=bins_num,
             color='hotpink', edgecolor='black', density=True)
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.show()


def show_distribution_function(arr: []) -> None:
    ecdf = ECDF(arr)
    plt.step(ecdf.x, ecdf.y, color='violet')
    plt.xlabel("x")
    plt.ylabel("F(x)")
    plt.show()


if __name__ == '__main__':
    count = 10 ** 4
    N = 10
    probability = 0.5
    values = [binomial_distribution(N, probability) for _ in range(count)]

    mat_og = sum(values) / count
    dispersion = sum([(value - mat_og) ** 2 for value in values]) / count

    print(f"Мат. ож. {mat_og}")
    print(f"Дисперсия {dispersion}")
```

```
    print(f"Погрешность для мат. ож. {mat_og - 5}")
    print(f"Погрешность для дисперсии {dispersion - 2.5}")

    show_density_distribution_function(values)
    show_distribution_function(values)
```

```
import math
import random
import matplotlib.pyplot as plt
from statsmodels.distributions.empirical_distribution import ECDF
import numpy as np


def first_algorithm(p: float) -> int:
    value = random.random()
    p_func = p
    result_m = 0
    while value >= p_func:
        value = value - p_func
        p_func = p_func * (1 - p)
        result_m = result_m + 1
    return result_m + 1


def second_algorithm(p: float) -> int:
    value = random.random()
    result_m = 0
    p_func = p
    while value >= p_func:
        value = random.random()
        result_m = result_m + 1
    return result_m + 1


def third_algorithm(p: float) -> int:
    value = random.random()
    result_m = math.floor(np.log(value) / np.log(1 - p)) + 1
    return result_m


def show_density_distribution_function(arr: []) -> None:
    bins_num = 13
    plt.hist(arr, bins=bins_num,
             color='violet', edgecolor='black', density=True)

    plt.xlabel('x')
    plt.ylabel('f(x)')
```

```
    plt.show()


def show_distribution_function(arr: []) -> None:
    ecdf = ECDF(arr)
    plt.step(ecdf.x, ecdf.y)
    plt.xlabel("x")
    plt.ylabel("F(x)")
    plt.show()


def show_info_for_data(values: []) -> None:
    mat_og = sum(values) / count
    dispersion = sum([(value - mat_og) ** 2 for value in values]) / count

    print(f"Мат. ож. {mat_og}")
    print(f"Дисперсия {dispersion}")
    print(f"Погрешность для мат. ож. {mat_og - 2}")
    print(f"Погрешность для дисперсии {dispersion - 2.2}")
    print()

    show_density_distribution_function(values)
    show_distribution_function(values)


if __name__ == '__main__':
    count = 10 ** 4
    probability = 0.5

    values = [first_algorithm(probability) for _ in range(count)]
    show_info_for_data(values)

    values = [second_algorithm(probability) for _ in range(count)]
    show_info_for_data(values)

    values = [third_algorithm(probability) for _ in range(count)]
    show_info_for_data(values)
```

```
import random
import matplotlib.pyplot as plt
from statsmodels.distributions.empirical_distribution import ECDF
import numpy as np


def first_algorithm(mu: float) -> int:
    result_m = 0
    if mu < 88:
        value = random.random()
```

```
        p_func = np.exp(-mu)
        result_m = 1
        while value >= p_func:
            value = value - p_func
            p_func = p_func * (mu / result_m)
            result_m = result_m + 1
    else:
        result_m = round(np.random.normal(1, mu, mu))
    return result_m - 1


def second_algorithm(mu: float) -> int:
    result_m = 0
    if mu < 88:
        result_m = 1
        p_func = random.random()
        exp_value = np.exp(-mu)
        while p_func >= exp_value:
            value = random.random()
            p_func = p_func * value
            result_m = result_m + 1
    else:
        result_m = round(np.random.normal(1, mu, mu))
    return result_m - 1


def show_density_distribution_function(arr: []) -> None:
    bins_num = 13
    plt.hist(arr, bins=bins_num,
             color='violet', edgecolor='black', density=True)

    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.show()


def show_distribution_function(arr: []) -> None:
    ecdf = ECDF(arr)
    plt.step(ecdf.x, ecdf.y)
    plt.xlabel("x")
    plt.ylabel("F(x)")
    plt.show()


def show_info_for_data(values: []) -> None:
    mat_og = sum(values) / count
    dispersion = sum([(value - mat_og) ** 2 for value in values]) / count
```

```
    print(f"Мат. ож. {mat_og}")
    print(f"Дисперсия {dispersion}")
    print(f"Погрешность для мат. ож. {mat_og - 10}")
    print(f"Погрешность для дисперсии {dispersion - 10}")
    print()

    show_density_distribution_function(values)
    show_distribution_function(values)


if __name__ == '__main__':
    count = 10 ** 4
    mu = 10

    values = [first_algorithm(mu) for _ in range(count)]
    show_info_for_data(values)

    values = [second_algorithm(mu) for _ in range(count)]
    show_info_for_data(values)
```

```
import numpy as np
import scipy.stats as stats
import random
import math

def third_algorithm(p: float) -> int:
    value = random.random()
    result_m = math.floor(np.log(value) / np.log(1 - p)) + 1
    return result_m

p = 0.7
sample_size = 50
alpha = 0.05

np.random.seed(42)  # Для воспроизводимости
random.seed(42)  # Установка seed для функции random
sample = [third_algorithm(p) for _ in range(sample_size)]

print("Сгенерированная выборка:", sample)

values, counts = np.unique(sample, return_counts=True)
empirical_probs = counts / sample_size

print("Эмпирические значения:", values)
print("Эмпирические частоты:", counts)
max_value = values.max()
theoretical_probs = [(1 - p)**(k - 1) * p for k in range(1, max_value + 1)]
```

```
print("Теоретические вероятности:", theoretical_probs)

observed = np.zeros(max_value)
for i, value in enumerate(values):
   observed[value - 1] = counts[i]

expected = np.array([sample_size * prob for prob in theoretical_probs])

nonzero_indices = expected > 0
observed = observed[nonzero_indices]
expected = expected[nonzero_indices]

observed_sum = observed.sum()
expected_sum = expected.sum()
observed_normalized = observed * (expected_sum / observed_sum)

print("Наблюдаемые частоты (нормализованные):", observed_normalized)
print("Ожидаемые частоты:", expected)

chi2_stat, p_value = stats.chisquare(f_obs=observed_normalized, f_exp=expected)

degrees_of_freedom = len(observed) - 1

print(f"Статистика хи-квадрат: {chi2_stat:.4f}")
print(f"p-значение: {p_value:.4f}")
print(f"Степень свободы: {degrees_of_freedom}")

if p_value < alpha:
   print("Гипотеза о соответствии отвергается на уровне значимости 0.05.")
else:
   print("Нет оснований отвергнуть гипотезу о соответствии на уровне значимости 0.05.")
critical_value = stats.chi2.ppf(1 - alpha, degrees_of_freedom)

# Вывод критического значения
print(f"Критическое значение для статистики хи-квадрат на уровне значимости {alpha}:
{critical_value:.4f}")
```