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

Отчет

по лабораторной работе №3 Способы получения случайных чисел с требуемым 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 3

Ход работы 3

Равномерное распределение 4

Нормальное распределение 4

Экспоненциальное распределение 5

Хи квадрат распределение 6

Распределение Стьюдента 6

Индивидуальное задание 7

Цель работы 7

Результаты 7

Вывод 7

Приложение 8

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

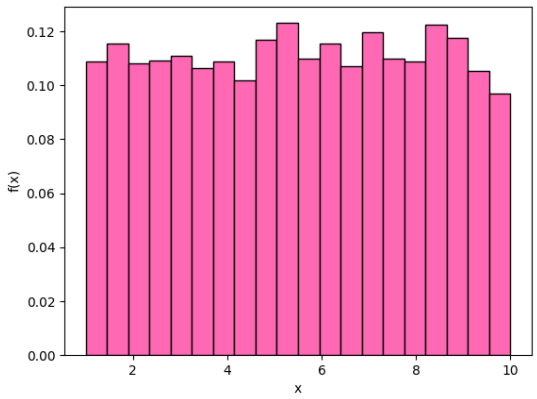

Рисунок 1 Плотность распределения

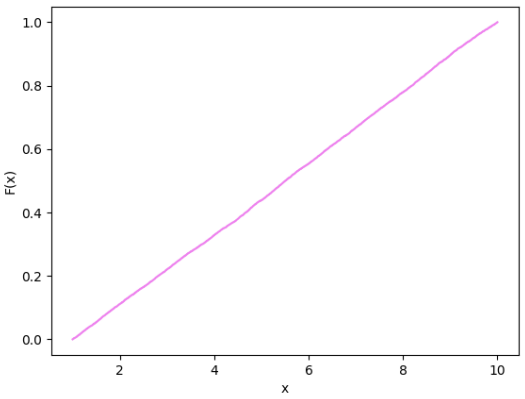

Рисунок 2 Функция распределения

- Мат. ож. 5.513064885193142
- Дисперсия 6.689614532010323
- Теоретическое Мат. ож. 5.5
- Теоретическая дисперсия 6.75
- Погрешность для мат. ож. 0.013064885193141862
- Погрешность для дисперсии -0.060385467989677366

Нормальное распределение

Рисунок 3 Плотность распределения

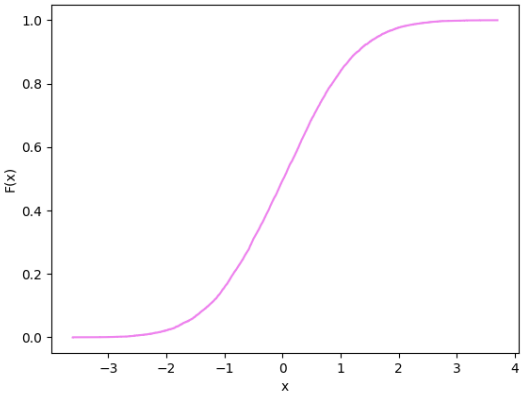

Рисунок 4 Функция распределения

- Мат. ож. 0.003390549260619092
- Дисперсия 1.0122043515826946
- Погрешность для мат. ож. 0.003390549260619092
- Погрешность для дисперсии 0.012204351582694617

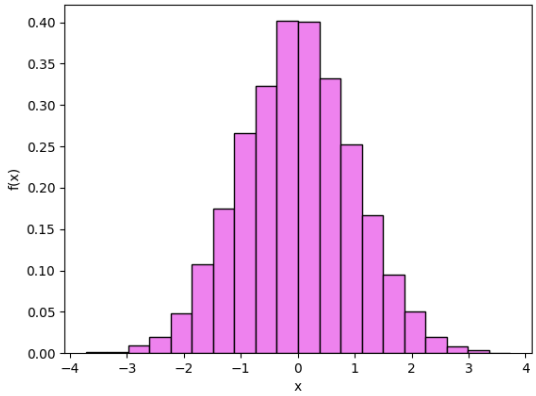

Рисунок 5 Плотность распределения

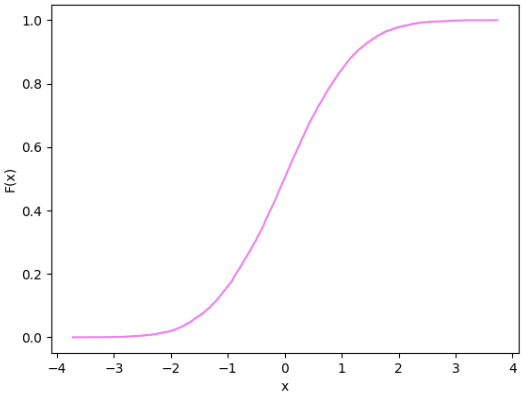

Рисунок 6 Функция распределения

- Мат. ож. -0.003077319772422606
- Дисперсия 1.0292694215803169
- Погрешность для мат. ож. -0.003077319772422606
- Погрешность для дисперсии 0.029269421580316868

Экспоненциальное распределение

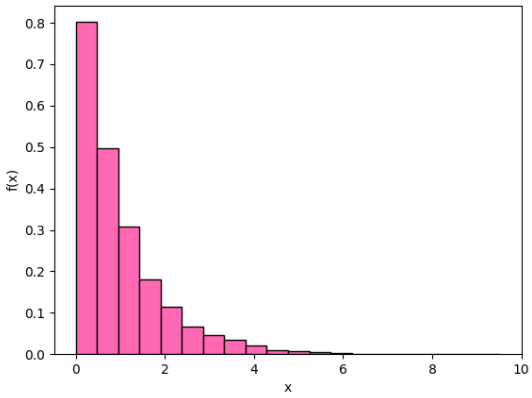

Рисунок 7 Плотность распределения алгоритм

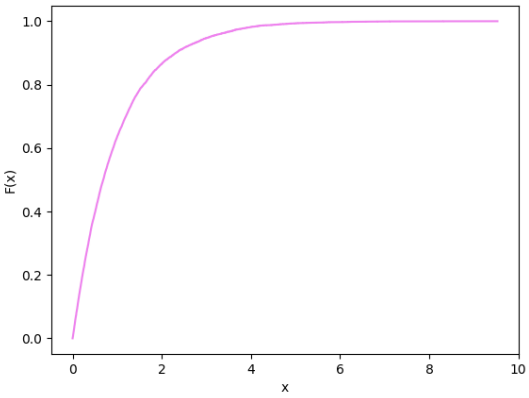

Рисунок 8 Функция распределения алгоритм

- Мат. ож. 1.0110128384095323
- Дисперсия 1.0150026632828406
- Погрешность для мат. ож. 0.011012838409532266
- Погрешность для дисперсии 0.015002663282840611

Хи квадрат распределение

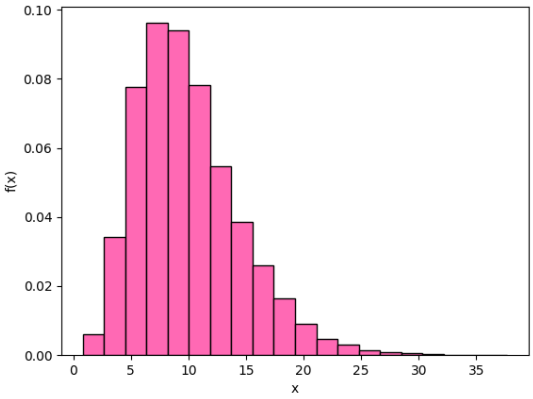

Рисунок 9 Плотность распределения алгоритм

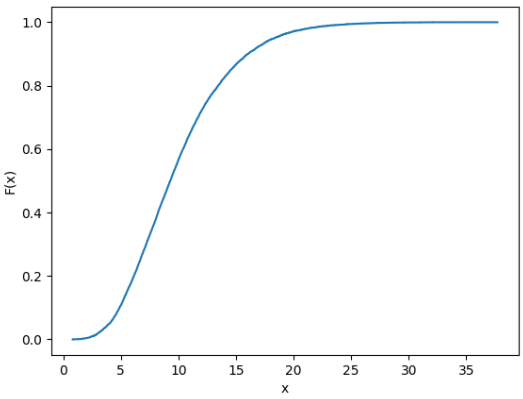

Рисунок 10 Функция распределения алгоритм

- Мат. ож. 9.961139546324581
- Дисперсия 20.04764235872303
- Погрешность для мат. ож. -0.03886045367541868
- Погрешность для дисперсии 0.047642358723031464

Распределение Стьюдента

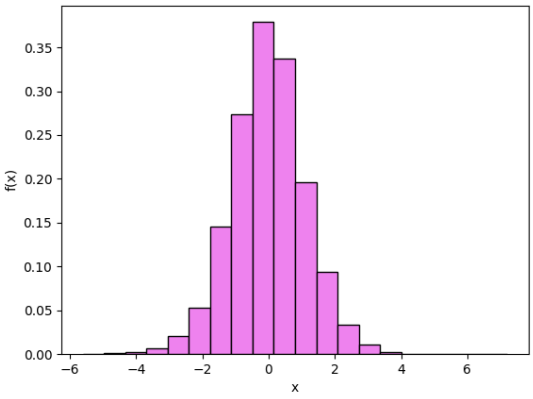

Рисунок 11 Плотность распределения алгоритм

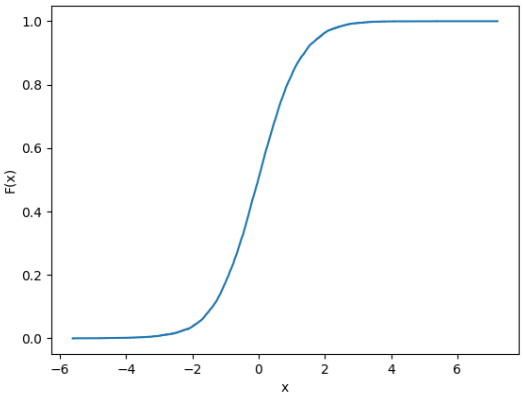

Рисунок 12 Функция распределения алгоритм

- Мат. ож. 0.002429123840703911
- Дисперсия 1.2642293063113301
- Погрешность для мат. ож. 0.002429123840703911
- Погрешность для дисперсии 0.014229306311330125

Индивидуальное задание

Цель работы

8. Смоделировать случайную величину ***X***, имеющую треугольное распределение с параметрами a=0, b=2. Смоделировать случайную величину Y, имеющую нормальный закон с параметрами m=1, σ =1. На основе выборок объема 50 каждой случайной величины исследовать однородность их распределений (то есть гипотеза H0 состоит в совпадении функций распределения СВ X и Y) критерием знаков с уровнем значимости 0,05.

Результаты

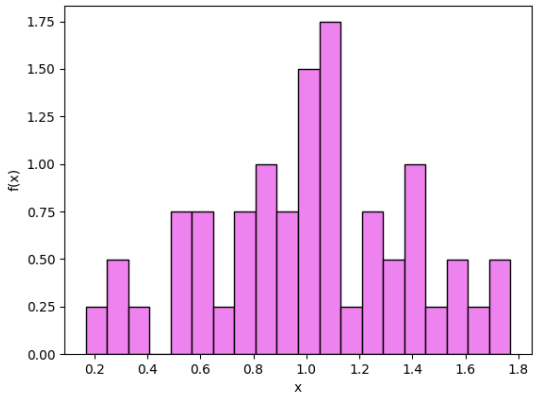

Рисунок 13 Плотность треугольного распределения

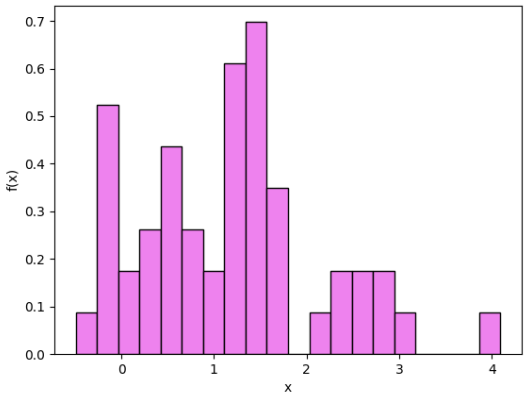

Рисунок 14 Плотность нормального распределения

Гипотеза H0 не отвергается. Распределения однородны.

p-value: 0.46

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(a: float, b: float) -> float:
    u = random.random()
    return (b - a) * u + a


def show_density_distribution_function(arr: []) -> None:
    bins_num = 3
    if len(arr) > 10:
        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__':
    low_value = 1
    up_value = 10
    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"Теоретическое Мат. ож. {(low_value + up_value) / 2}")
    print(f"Теоретическая дисперсия {((up_value - low_value) ** 2) / 12}")
    print(f"Погрешность для мат. ож. {mat_og - (low_value + up_value) / 2}")
    print(f"Погрешность для дисперсии {dispersion - ((up_value - low_value) ** 2) /
12}")

    show_density_distribution_function(values)
    show_distribution_function(values)
```

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


def normal_distribution_first_algorithm() -> float:
    r_12 = [random.random() for _ in range(12)]
    return sum(r_12) - 6


def normal_distribution_second_algorithm() -> float:
    u1 = random.random()
    u2 = random.random()
    return np.sqrt(-2 * np.log(u2)) * np.sin(2 * np.pi * u1)


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='violet')
    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 - 0}")
    print(f"Погрешность для дисперсии {dispersion - 1}")
    print()

    show_density_distribution_function(values)
    show_distribution_function(values)


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

    values = [normal_distribution_first_algorithm() for _ in range(count)]
```

```
    show_info_for_data(values)

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

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


def exponent_distribution(beta: float) -> float:
    u = random.random()
    return - (beta * np.log(u))


def show_density_distribution_function(arr: []) -> None:
    bins_num = 3
    if len(arr) > 10:
        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()


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 - 1}")
    print(f"Погрешность для дисперсии {dispersion - 1}")
    print()

    show_density_distribution_function(values)
    show_distribution_function(values)


if __name__ == '__main__':
    beta = 1
    count = 10 ** 4
```

```
    values = [exponent_distribution(beta) for _ in range(count)]
    show_info_for_data(values)
```

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


def normal_distribution() -> float:
    u1 = random.random()
    u2 = random.random()
    return np.sqrt(-2 * np.log(u2)) * np.sin(2 * np.pi * u1)


def xi_distribution(n: int) -> float:
    result = []
    for _ in range(n):
        result.append(normal_distribution() ** 2)
    return sum(result)


def show_density_distribution_function(arr: []) -> None:
    bins_num = 3
    if len(arr) > 10:
        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)
    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 - 20}")
    print()

    show_density_distribution_function(values)
    show_distribution_function(values)
```

```
if __name__ == '__main__':
    count = 10 ** 4
    N = 10

    values = [xi_distribution(N) for _ in range(count)]
    show_info_for_data(values)
```

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


def normal_distribution() -> float:
    u1 = random.random()
    u2 = random.random()
    return np.sqrt(-2 * np.log(u2)) * np.sin(2 * np.pi * u1)


def xi_distribution(n: int) -> float:
    result = []
    for _ in range(n):
        result.append(normal_distribution() ** 2)
    return sum(result)


def student_distribution(n: int) -> float:
    z = normal_distribution()
    Yn = xi_distribution(n)
    return z / np.sqrt(Yn / n)


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)
    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 - 0}")
    print(f"Погрешность для дисперсии {dispersion - 1.25}")
    print()

    show_density_distribution_function(values)
    show_distribution_function(values)


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

    values = [student_distribution(N) for _ in range(count)]
    show_info_for_data(values)
```

```
import random

import numpy as np
from matplotlib import pyplot as plt


def triangular_alg(n, a, b):
    # Генерируем случайные числа равномерно распределенные на отрезке [0, 1]
    u = np.random.rand(n)

    # Масштабируем u к интервалу [a, b] для заданных параметров a и b
    scaled_u = (b - a) * u + a

    # Вычисляем значения случайной величины X в соответствии с треугольным
распределением
    X = np.where(scaled_u <= 1, np.sqrt(scaled_u), b - np.sqrt(b - scaled_u))

    return X


def normal_arg(n, mean, std_dev):
    # Генерируем две выборки равномерно распределенных на [0, 1]
    u1 = np.random.rand(n)
    u2 = np.random.rand(n)

    # Используем преобразование Бокса-Мюллера для генерации значений с нормальным
распределением
    z0 = np.sqrt(-2 * np.log(u1)) * np.cos(2 * np.pi * u2)
    Y = mean + std_dev * z0

    return Y


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 sign_test(X, Y, alpha=0.05):
    differences = X - Y
    positive_count = np.sum(differences > 0)
    negative_count = np.sum(differences < 0)

    n = len(differences)

    # Рассчитываем p-value
    p_value = min(positive_count, negative_count) / n

    # Проводим тест
    if p_value < alpha:
        print("Гипотеза H0 отвергается. Распределения неоднородны.")
    else:
        print("Гипотеза H0 не отвергается. Распределения однородны.")
    print("p-value:", p_value)


if __name__ == '__main__':
    count = 50

    a = 0
    b = 2
    m = 1
    std_dev = 1

    values_X = triangular_alg(count, a, b)
    values_Y = normal_arg(count, m, std_dev)
    show_density_distribution_function(values_X)
    show_density_distribution_function(values_Y)

    # Проведем тест
    sign_test(values_X, values_Y)
```