МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Автоматизация схемотехнического проектирования»

на тему «Генерация модельных наборов данных»

Студенты гр. 1301 _____ Семейкин С.А.

_____ Гальченко М.А.

Преподаватель _____ Боброва Ю.О.

Санкт-Петербург

2025

Цель:

Получение навыков работы с numpy-массивами и написание функций на языке Python на примере генерации массивов произвольно распределенных данных.

Ход работы:

1. Создадим переменные, распределение по нормальному закону с незначительно различными средними и дисперсиями.
2. Создадим переменные, соответствующие классам.
3. Визуализируем результаты, чтобы оценить пересекаемость классов.
4. В первую очередь построим гистограммы для каждого признака и скаттерограммы для каждой пары признаков.

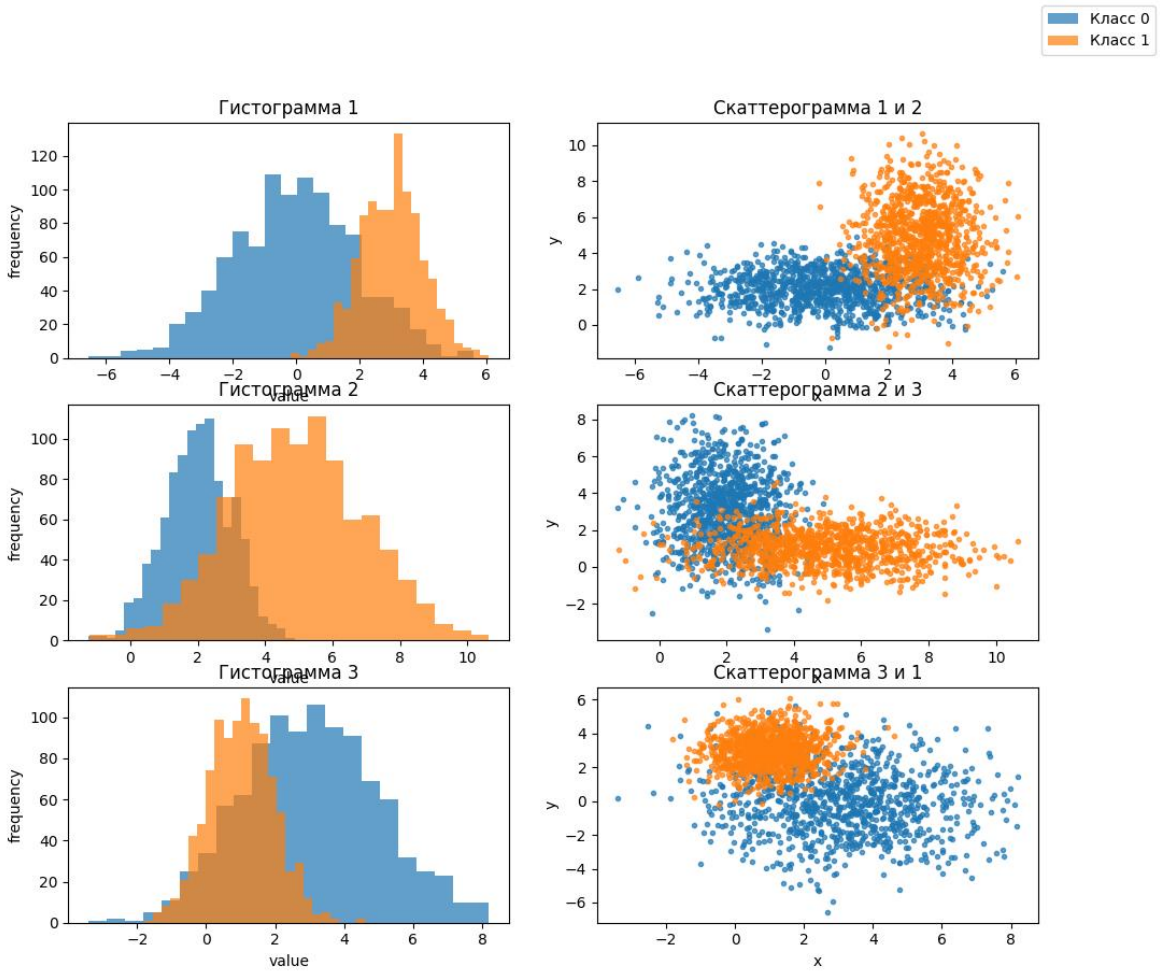

Рисунок 1 Гистограммы и скаттерограммы признаков

5. Создадим метод класса DataGenerator nonlinear_dataset_N, который генерирует двумерный массив данных, распределенный в пространстве в виде заданных фигур на рисунке 2.

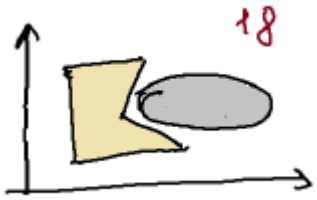

Рисунок 2 Необходимый вид распределения

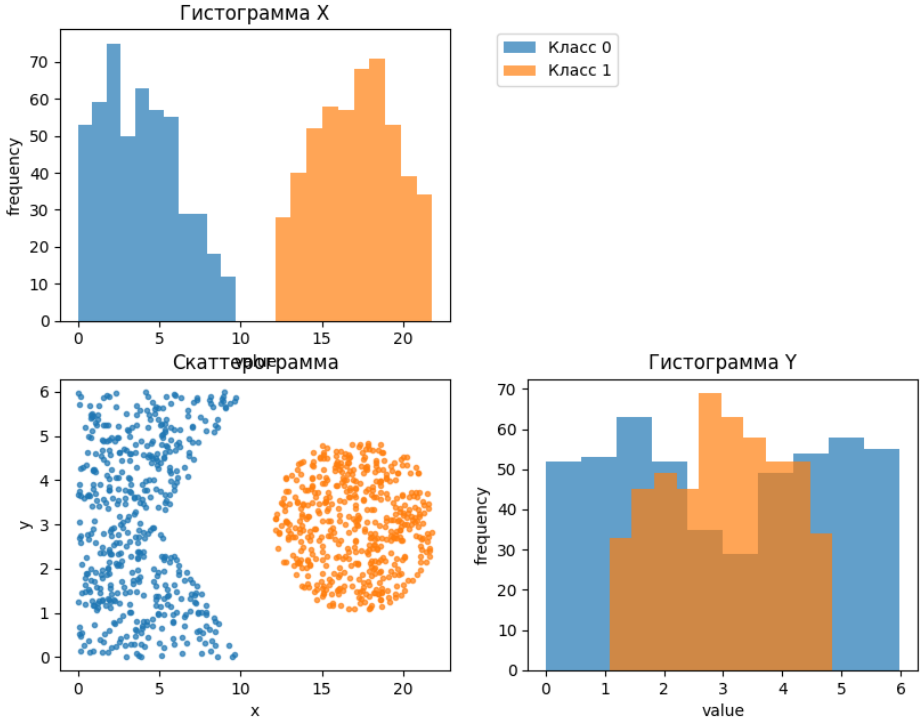

Рисунок 3 Гистограммы и скаттерограмма заданного распределения

Вывод:

В ходе лабораторной работы были получены навыки работы с numpy-массивами и написания функций на языке Python на примере генерации массивов произвольно распределенных данных.

Благодаря изучению numpy-массивов была получена возможность эффективно обрабатывать и манипулировать данными. Кроме того, реализация функций для генерации массивов с произвольным распределением данных позволила углубить понимание работы с различными статистическими распределениями и их влиянием на создаваемые данные.

Листинг:

data_generator.py

```
import numpy as np
from typing import Any
from matplotlib.path import Path


def is_point_in_pentagon(x, y, rect_x, rect_y, rect_width, rect_height):
    # Определяем вершины пятиугольника
    bottom_left = (rect_x, rect_y)
    bottom_right = (rect_x + rect_width, rect_y)
    top_right = (rect_x + rect_width, rect_y + rect_height)
    top_left = (rect_x, rect_y + rect_height)
    center_right = (rect_x + rect_width / 2, rect_y + rect_height / 2)

    pentagon_vertices = [bottom_left, bottom_right, center_right, top_right,
top_left, bottom_left]

    pentagon_path = Path(np.array(pentagon_vertices))

    return pentagon_path.contains_point((x, y))  # , pentagon_vertices


def is_point_in_ellipse(x, y, center_x, center_y, width, height):
    norm_x = (x - center_x) / (width / 2)
    norm_y = (y - center_y) / (height / 2)
    return norm_x ** 2 + norm_y ** 2 <= 1


class DataGenerator:
    @staticmethod
    def norm dataset(mu: tuple[list[int], list[int]],
                     sigma: tuple[list[int], list[int]],
                     N: int
                     ) -> \
            tuple[np.ndarray[np.floating[np.float64], Any],
            np.ndarray[np.floating[np.float64], Any],
            np.ndarray[np.floating[np.float64], Any],
            np.ndarray[np.floating[np.float64], Any]]:
        mu0 = mu[0]
        mu1 = mu[1]
        sigma0 = sigma[0]
        sigma1 = sigma[1]

        col = len(mu0)
        class0 = np.random.normal(mu0[0], sigma0[0], [N, 1])  #
инициализируем первый столбец

        class1 = np.random.normal(mu1[0], sigma1[0], [N, 1])
        for i in range(1, col):
            v0 = np.random.normal(mu0[i], sigma0[i], [N, 1])
            class0 = np.hstack((class0, v0))
            v1 = np.random.normal(mu1[i], sigma1[i], [N, 1])
            class1 = np.hstack((class1, v1))
        X = np.vstack((class0, class1))
        # print(X)
        # print(X.size)
        Y0 = np.zeros((N, 1), dtype=bool)
        Y1 = np.ones((N, 1), dtype=bool)
        Y = np.vstack((Y0, Y1)).ravel()

        rng = np.random.default_rng()
```

```
        arr = np.arange(2 * N)  # индексы для перемешивания [ 0, 1 ,2 ....
1999]
        rng.shuffle(arr)
        X = X[arr]
        Y = Y[arr]

        return X, Y, class0, class1

    @staticmethod
    def nonlinear_dataset_N(num_points):
        """
        Генерирует два массива точек, распределенных в пределах пятиугольника
и эллипса.
        """
        pentagon_params = (0, 0, 10, 6)  # x, y, width, height
        ellipse_params = (17, 3, 10, 4)  # center_x, center_y, width, height

        pentagon_points = []
        ellipse_points = []

        x_min, x_max = min(pentagon_params[0], ellipse_params[0] -
ellipse_params[2] / 2), max(
            pentagon_params[0] + pentagon_params[2], ellipse_params[0] +
ellipse_params[2] / 2)
        y_min, y_max = min(pentagon_params[1], ellipse_params[1] -
ellipse_params[3] / 2), max(
            pentagon_params[1] + pentagon_params[3], ellipse_params[1] +
ellipse_params[3] / 2)

        while len(pentagon_points) < num_points or len(ellipse_points) <
num_points:
            x = np.random.uniform(x_min, x_max)
            y = np.random.uniform(y_min, y_max)

            if len(pentagon_points) < num_points and is_point_in_pentagon(x,
y, *pentagon_params):
                pentagon_points.append((x, y))

            if len(ellipse_points) < num_points and is_point_in_ellipse(x, y,
*ellipse_params):
                ellipse_points.append((x, y))
        class0 = np.array(pentagon_points)
        class1 = np.array(ellipse_points)
        X = np.vstack((class0,class1))
        Y0 = np.zeros((num_points, 1), dtype=bool)
        Y1 = np.ones((num_points, 1), dtype=bool)
        Y = np.vstack((Y0, Y1)).ravel()

        rng = np.random.default_rng()
        arr = np.arange(num_points)
        rng.shuffle(arr)
        X = X[arr]
        Y = Y[arr]
        return X, Y, class0, class1
```

norm_dataset.py

```
import matplotlib.pyplot as plt
from data_generator import DataGenerator

mu0 = [0, 2, 3]
mu1 = [3, 5, 1]
sigma0 = [2, 1, 2]
sigma1 = [1, 2, 1]
col = len(mu0)
N = 1000
X, Y, class0, class1 = DataGenerator.norm_dataset((mu0, mu1), (sigma0,
sigma1), N)

train_size = 0.7
trainCount = round(train_size * N * 2)  # *2 потому что было 2 класса
Xtrain = X[0:trainCount]
Xtest = X[trainCount:N * 2 + 1]
Ytrain = Y[0:trainCount]
Ytest = Y[trainCount:N * 2 + 1]

if __name__ == "__main__":
    figure, axis = plt.subplots(col, 2)
    for i in range(0, col):
        axis[i, 0].set_title(f"Гистограмма {i + 1}")
        hist_class0 = axis[i, 0].hist(class0[:, i], bins='auto', alpha=0.7,
                                      label='Класс 0')
        hist_class1 = axis[i, 0].hist(class1[:, i], bins='auto', alpha=0.7,
label='Класс 1')
        axis[i, 0].set_xlabel("value")
        axis[i, 0].set_ylabel("frequency")
        axis[i, 1].set_xlabel("x")
        axis[i, 1].set_ylabel("y")
        if i != col - 1:
            axis[i, 1].set_title(f"Скаттерограмма {i + 1} и {i + 2}")
            axis[i, 1].scatter(class0[:, i], class0[:, i + 1], marker=".",
alpha=0.7, label='Класс 0')
            axis[i, 1].scatter(class1[:, i], class1[:, i + 1], marker=".",
alpha=0.7, label='Класс 1')
        else:
            axis[i, 1].set_title(f"Скаттерограмма {i + 1} и {1}")

            axis[i, 1].scatter(class0[:, i], class0[:, 0], marker=".",
alpha=0.7, label='Класс 0')
            axis[i, 1].scatter(class1[:, i], class1[:, 0], marker=".",
alpha=0.7, label='Класс 1')

    figure.legend(['Класс 0', 'Класс 1'])
    plt.show()
```

custom_dataset.py

```
import matplotlib.pyplot as plt
from data_generator import DataGenerator

col = 2
N = 500
X, Y, class0, class1 = DataGenerator.nonlinear_dataset_N(N)
figure, axis = plt.subplots(2, 2)

axis[0, 0].set_title(f"Гистограмма X")
axis[0, 0].set_xlabel("value")
```

```
axis[0, 0].set_ylabel("frequency")
axis[0, 0].hist(class0[:, 0], bins='auto', alpha=0.7,
                label='5 угольник')  # параметр alpha позволяет задать
прозрачность цвета\
axis[0, 0].hist(class1[:, 0], bins='auto', alpha=0.7,
                label='Овал')  # параметр alpha позволяет задать прозрачность
цвета

axis[1, 1].set_title(f"Гистограмма Y")
axis[1, 1].set_xlabel("value")
axis[1, 1].set_ylabel("frequency")
axis[1, 1].hist(class0[:, 1], bins='auto', alpha=0.7,
                label='5 угольник')  # параметр alpha позволяет задать
прозрачность цвета
axis[1, 1].hist(class1[:, 1], bins='auto', alpha=0.7,
                label='Овал')  # параметр alpha позволяет задать прозрачность
цвета
axis[1, 0].set_title(f"Скаттерограмма")
axis[1, 0].set_xlabel("x")
axis[1, 0].set_ylabel("y")
axis[1, 0].scatter(class0[:, 0], class0[:, 1], marker=".", alpha=0.7,
label='5 угольник')
axis[1, 0].scatter(class1[:, 0], class1[:, 1], marker=".", alpha=0.7,
label='Овал')

figure.legend(['Класс 0', 'Класс 1'])
plt.savefig("test_jpg.png")
plt.show()
```