Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
дисциплины
«Искусственный интеллект и машинное обучение»

Выполнил:
Хохлачев Вадим Александрович
2 курс, группа ИТС-б-о-23-1,
11.03.02«Инфокоммуникационные технологии и системы связи», очная форма обучения

_____
(подпись)

Проверил:
Ассистеннт департамента цифровых, робототехнических систем и электроники Хацукова А.И.

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Тема: Основы работы с библиотекой matplotlib

Цель: исследовать базовые возможности библиотеки matplotlib языка программирования Python

Порядок выполнения работы:

1. Ознакомились с теоретическим материалом

2. Создание репозитория

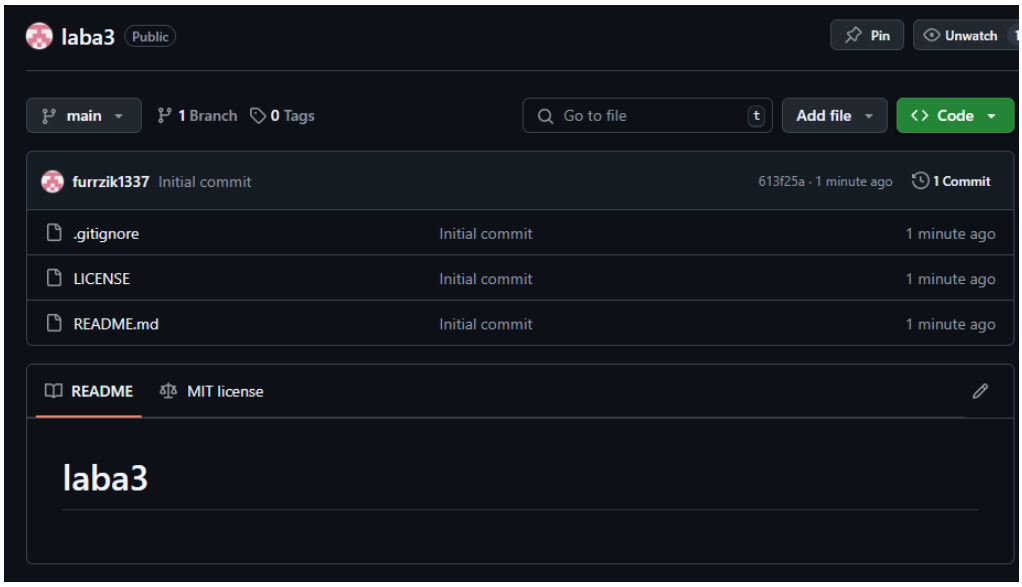

Рисунок 1. Репозиторий

3. Выполнено клонирование репозитория

```
C:\Users\furrzik>git clone https://github.com/furrzik1337/laba3.git
Cloning into 'laba3'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (5/5), done.
```

Рисунок 2. Клонирование

4. Выполнено практическое задание 1.

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt

# 1. Создание данных:
x = np.linspace(-10, 10, 100)  # 100 точек на интервале [-10, 10]
y = x**2                       # Вычисляем y = x^2 для каждого x

# 2. Создание графика:
plt.figure(figsize=(8, 6))    # Создаем новое окно для графика (опционально, но полезно)

# 3. Построение графика:
plt.plot(x, y, label='y = x²') # Строим график и добавляем метку для легенды (опционально)

# 4. Настройка графика:
plt.xlabel('x')                # Подпись оси X
plt.ylabel('y')                # Подпись оси Y
plt.title('График функции y = x²') # Заголовок графика
plt.grid(True)                 # Включаем сетку
plt.legend()                  # Показываем легенду (если есть метки в plt.plot)

# 5. Отображение графика:
plt.show()                     # Показываем график (обязательно!)
```

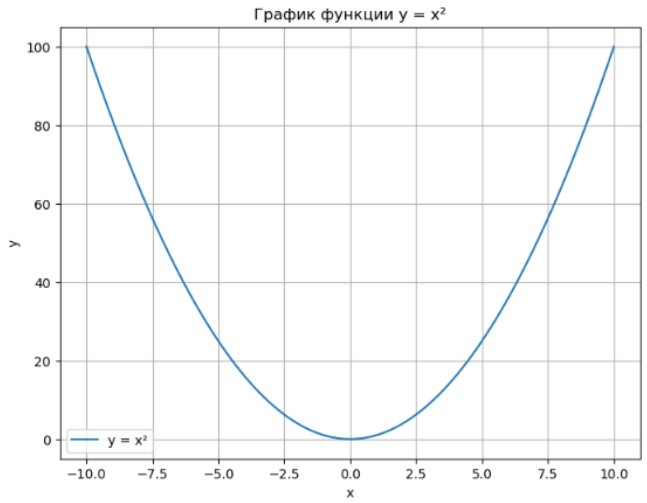

Рисунок 3. График функции

5. Выполнено практическое задание 2.

```
[1]: import numpy as np
     import matplotlib.pyplot as plt

     x = np.linspace(-10, 10, 100)
     plt.plot(x, x, linestyle='--', color='blue', label='y = x')
     plt.plot(x, x**2, linestyle='-.', color='green', label='y = x²')
     plt.plot(x, x**3, linestyle='-', color='red', label='y = x³')
     plt.xlabel('x')
     plt.ylabel('y')
     plt.title('Графики функций')
     plt.grid(True)
     plt.legend()
     plt.gca().set_aspect('equal', adjustable='box')
     plt.show()
```

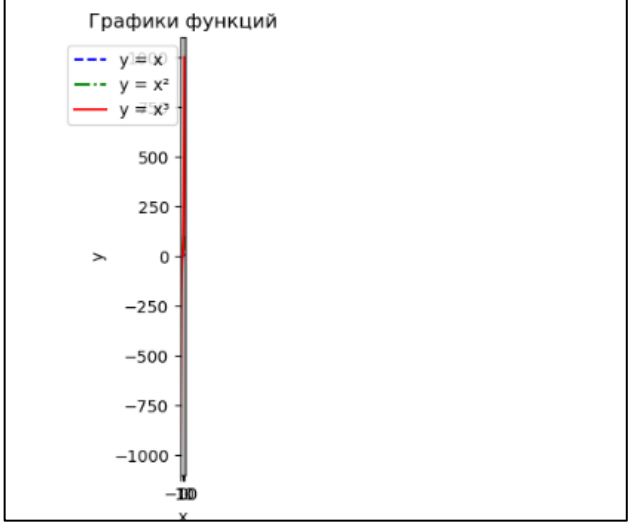

Рисунок 4. Настройка стилей и цвета

6. Выполнено практическое задание 3.

```python
import numpy as np
import matplotlib.pyplot as plt

# 1. Генерация случайных данных:
np.random.seed(42)  # Для воспроизводимости результатов
x = np.random.rand(50) * 20 - 10  # 50 случайных x-координат в диапазоне [-10, 10]
y = np.random.rand(50) * 20 - 10  # 50 случайных y-координат в диапазоне [-10, 10]

# 2. Задание цветов и размеров:
colors = x      # Цвет зависит от x-координаты
sizes = y * 10 # Размер зависит от y-координаты (умножаем, чтобы лучше видеть разницу)

# Важно: размеры точек должны быть положительными
sizes = np.abs(sizes) + 1  # Добавляем 1, чтобы все размеры были положительными и не были слишком маленькими

# 3. Построение диаграммы рассеяния:
plt.figure(figsize=(8, 6)) # Опционально: задаем размер графика
plt.scatter(x, y, c=colors, s=sizes, alpha=0.7, cmap='viridis') # c - цвет, s - размер, alpha - прозрачность, cmap - цветовая схема

# 4. Настройка графика:
plt.xlabel('x')
plt.ylabel('y')
plt.title('Диаграмма рассеяния с зависимостью цвета и размера от координат')
plt.colorbar(label='Координата x (цвет)') # Добавляем цветовую шкалу для понимания соответствия цвета x

# 5. Отображение графика:
plt.show()
```

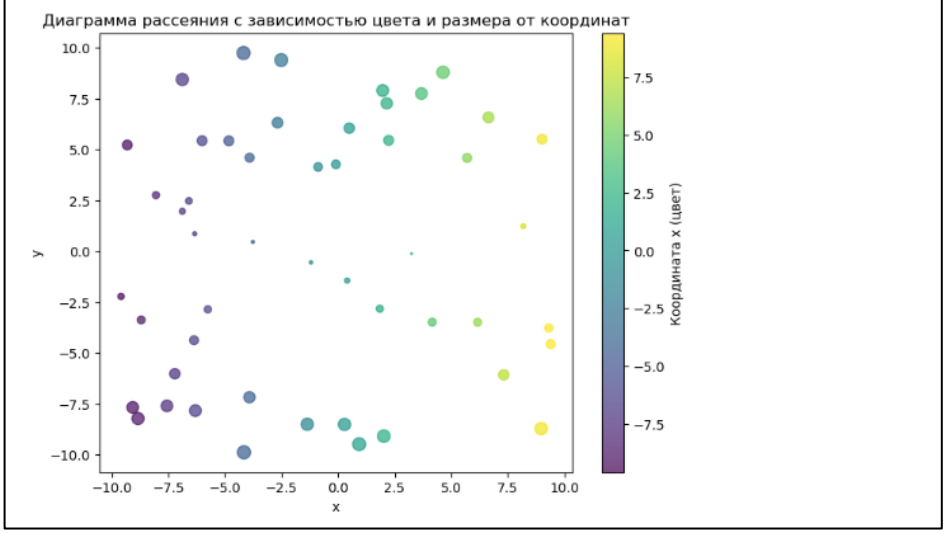

Рисунок 5. Различные типы графиков

7. Выполнено практическое задание 4.

```python
import numpy as np
import matplotlib.pyplot as plt

# 1. Генерация случайных чисел:
mu = 0       # Среднее значение (мат. ожидание)
sigma = 1    # Стандартное отклонение
num_samples = 1000  # Количество случайных чисел
num_bins = 30     # Количество бинов в гистограмме

data = np.random.normal(mu, sigma, num_samples) # Генерируем случайные числа

# 2. Построение гистограммы:
plt.figure(figsize=(10, 6)) # Опционально: размер графика
plt.hist(data, bins=num_bins, density=False, alpha=0.7, color='skyblue', edgecolor='black') # Строим гистограмму

# Параметры:
#   - data:  Исходные данные
#   - bins:  Количество бинов
#   - density: Если True, нормирует гистограмму так, чтобы сумма площадей всех столбцов равнялась 1. False - абсолютные значения.
#   - alpha: Прозрачность столбцов (от 0 до 1)
#   - color:  Цвет столбцов
#   - edgecolor: Цвет границ столбцов

# 3. Добавление вертикальной линии в среднем значении:
mean_value = np.mean(data)      # Вычисляем среднее значение
plt.axvline(mean_value, color='red', linestyle='dashed', linewidth=2, label=f'Среднее: {mean_value:.2f}') # Добавляем вертикальную линию

# 4. Настройка графика:
plt.xlabel('Значение')
plt.ylabel('Частота')
plt.title('Гистограмма нормального распределения')
plt.legend() # Показываем легенду

# 5. Отображение графика:
plt.show()
```

Рисунок 6. Листинг программы

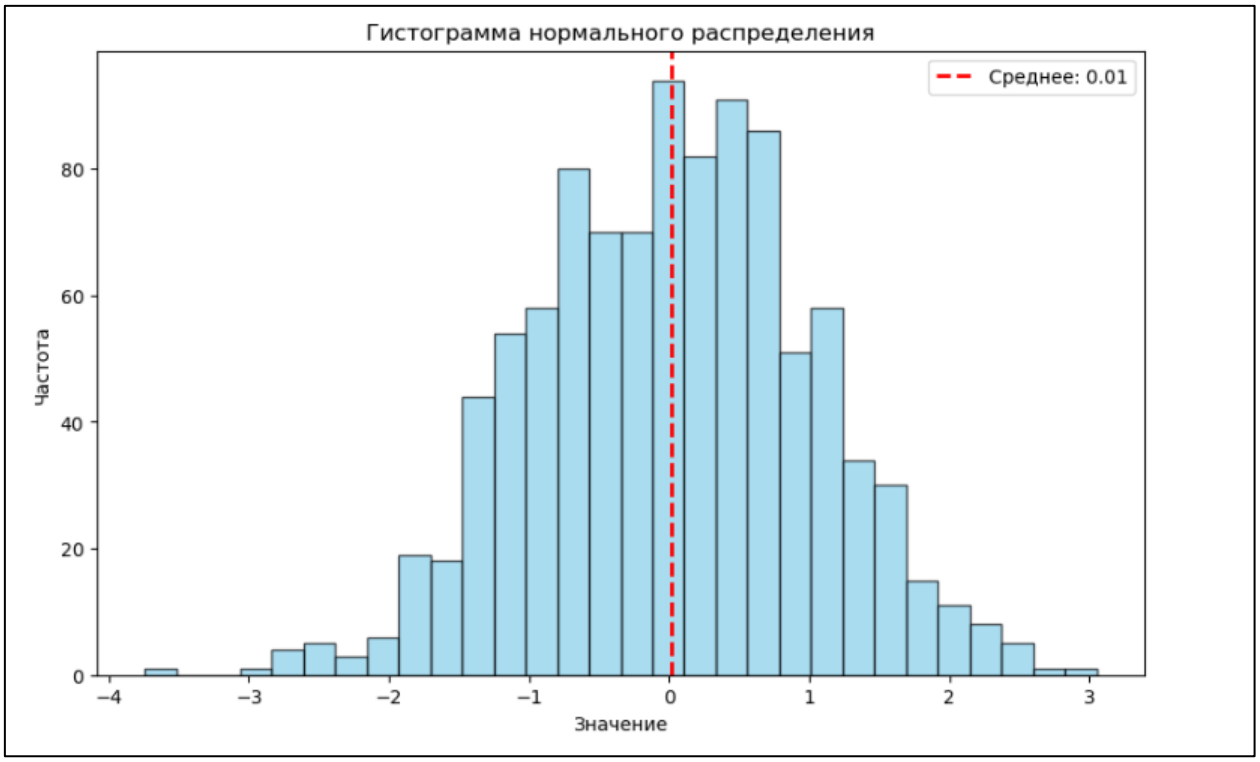

Рисунок 7. Гистограмма

8. Выполнено практическое задание 5.

```
1]: import matplotlib.pyplot as plt
    import numpy as np

    # 1. Данные:
    grades = ['Отлично', 'Хорошо', 'Удовлетворительно', 'Неудовлетворительно']
    counts = [20, 35, 30, 15]

    # 2. Создание столбчатой диаграммы:
    plt.figure(figsize=(8, 6))

    # Вариант 1: Используя plt.bar() напрямую
    #plt.bar(grades, counts, color='skyblue', edgecolor='black')

    # Вариант 2: С более явным созданием осей (Axes)
    ax = plt.gca()  # Получаем текущие оси (Axes)
    ax.bar(grades, counts, color='skyblue', edgecolor='black', width=0.6) # Задаем ширину столбцов

    # 3. Настройка графика:
    plt.xlabel('Оценка')  # Подпись оси X
    plt.ylabel('Количество студентов')  # Подпись оси Y
    plt.title('Распределение оценок студентов')  # Заголовок

    # Вариант 1: Просто plt.show()
    #plt.show()

    # Вариант 2: Более детальная настройка отображения (например, для сохранения в файл)
    plt.tight_layout() # Предотвращает обрезание подписей
    plt.show()
```

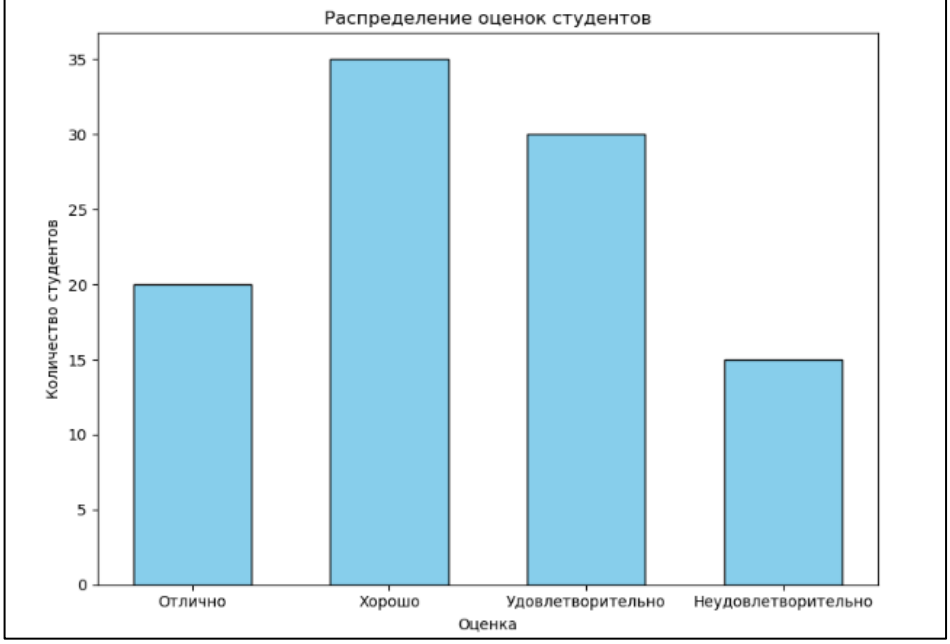

Рисунок 8. Столбчатая диаграмма

9. Выполнено практическое задание 6.

```python
import matplotlib.pyplot as plt

# 1. Данные:
grades = ['Отлично', 'Хорошо', 'Удовлетворительно', 'Неудовлетворительно']
counts = [20, 35, 30, 15]

# 2. Создание круговой диаграммы:
plt.figure(figsize=(8, 6))

plt.pie(counts, labels=grades, autopct='%1.1f%%', startangle=90,
        explode=(0, 0.1, 0, 0),  # "Выдвигаем" сектор "Хорошо" для наглядности
        shadow=True,             # Добавляем тень
        colors=['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']) # Задаем цвета секторов

# Параметры:
#   - counts: Размеры секторов
#   - labels: Подписи секторов
#   - autopct:  Формат процентных подписей (например, '%1.1f%%' - одно число после запятой)
#   - startangle: Угол, с которого начинается первый сектор (90 - "Отлично" будет сверху)
#   - explode: Расстояние, на которое "выдвигается" сектор (список, соответствующий секторам)
#   - shadow:  Добавляет тень к диаграмме
#   - colors: Список цветов для каждого сектора.

# 3. Настройка графика:
plt.title('Распределение оценок студентов')

# 4. Отображение графика:
plt.show()
```

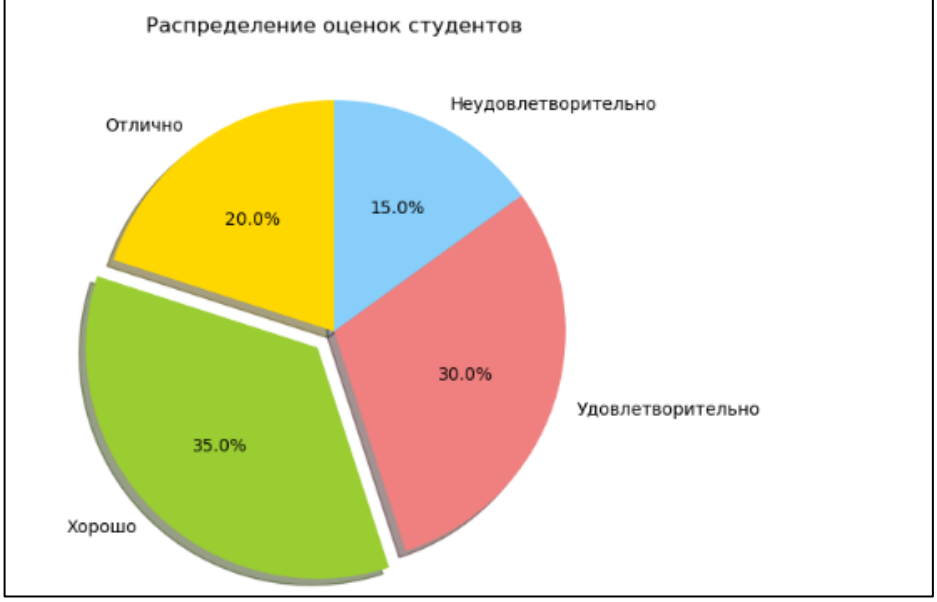

Рисунок 9. Круговая диаграмма

10. Выполнено практическое задание 7.

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# 1. Создание сетки координат:
x = np.linspace(-5, 5, 50) # Создаем 50 точек от -5 до 5 по оси x
y = np.linspace(-5, 5, 50) # Создаем 50 точек от -5 до 5 по оси y
X, Y = np.meshgrid(x, y) # Создаем сетку (матрицы X и Y, где каждая ячейка соответствует координате (x,y))

# 2. Вычисление значений Z:
Z = np.sin(np.sqrt(X**2 + Y**2)) # Вычисляем z = sin(√(x² + y²))

# 3. Создание 3D-графика:
fig = plt.figure(figsize=(10, 8))  # Создаем окно для графика
ax = fig.add_subplot(111, projection='3d') # Создаем 3D-подграфик

# 4. Построение поверхности:
surf = ax.plot_surface(X, Y, Z, cmap='viridis')  # Рисуем поверхность

# Другие варианты графиков:
# wireframe = ax.plot_wireframe(X,Y,Z)  # Построение каркасной модели
# surface = ax.plot_surface(X,Y,Z, rstride=1, cstride=1, cmap = cm.viridis) #Более гибкий способ отрисовки

# 5. Настройка графика:
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('График функции z = sin(√(x² + y²))')
fig.colorbar(surf)  # Добавляем цветовую шкалу

# 6. Отображение графика:
plt.show()
```

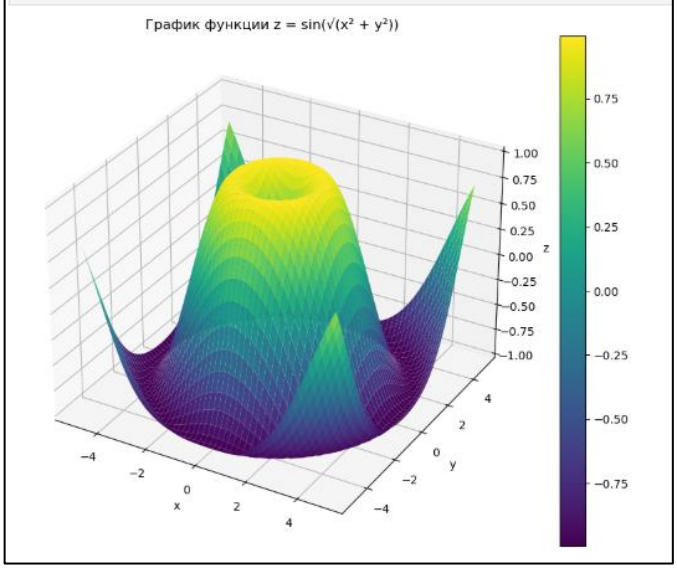

Рисунок 10. Трехмерный график поверхности

11. Выполнено практическое задание 8.

```
[1]: import numpy as np
     import matplotlib.pyplot as plt

     # 1. Данные:
     x = np.linspace(-10, 10, 100)
     y1 = x
     y2 = x**2
     y3 = np.sin(x)
     y4 = np.cos(x)

     # 2. Создание фигуры и подграфиков:
     fig, axs = plt.subplots(2, 2, figsize=(12, 10))

     # 3. Заполнение подграфиков:
     axs[0, 0].plot(x, y1, color='blue')
     axs[0, 0].set_title('Линейный график y = x')
     axs[0, 0].set_xlabel('x')
     axs[0, 0].set_ylabel('y')
     axs[0, 0].grid(True)

     axs[0, 1].plot(x, y2, color='green')
     axs[0, 1].set_title('Парабола y = x²')
     axs[0, 1].set_xlabel('x')
     axs[0, 1].set_ylabel('y')
     axs[0, 1].grid(True)

     axs[1, 0].plot(x, y3, color='red')
     axs[1, 0].set_title('Синус y = sin(x)')
     axs[1, 0].set_xlabel('x')
     axs[1, 0].set_ylabel('y')
     axs[1, 0].grid(True)

     axs[1, 1].plot(x, y4, color='purple')
     axs[1, 1].set_title('Косинус y = cos(x)')
     axs[1, 1].set_xlabel('x')
     axs[1, 1].set_ylabel('y')
     axs[1, 1].grid(True)

     # 4. Настройка общего графика (опционально):
     fig.suptitle('Четыре графика в одной фигуре', fontsize=16)
     plt.tight_layout(rect=[0, 0.03, 1, 0.95])

     # 5. Отображение графика:
     plt.show()
```

Рисунок 11. Листинг программы

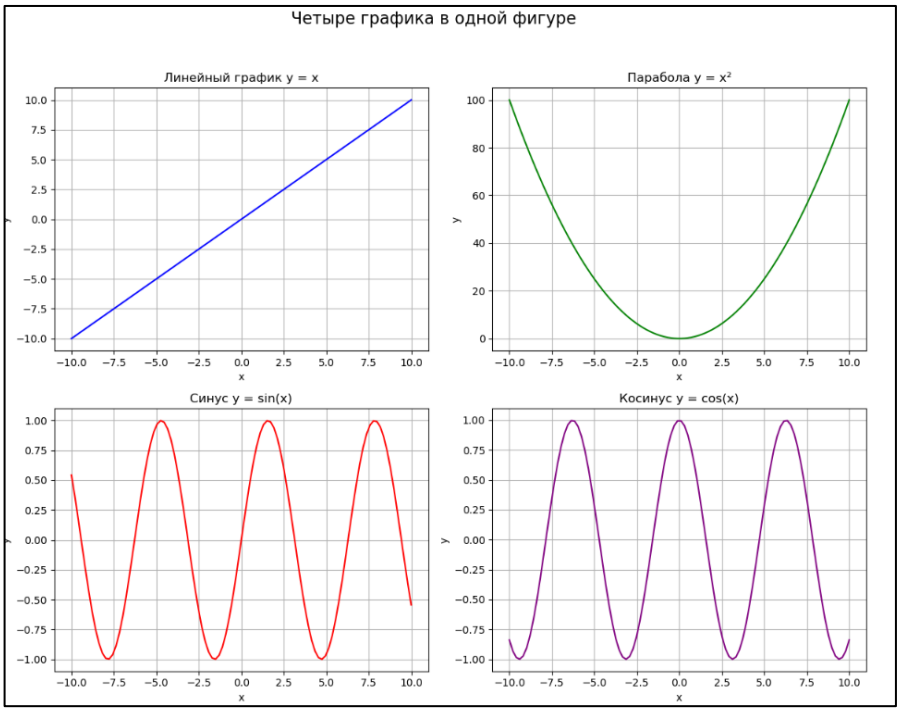

Рисунок 12. Множественные подграфики

12. Выполнено практическое задание 9.

```
]: import numpy as np
import matplotlib.pyplot as plt

# 1. Создание случайной матрицы:
np.random.seed(42)  # Для воспроизводимости результатов
matrix = np.random.rand(10, 10)  # Матрица 10x10 со случайными числами от 0 до 1

# 2. Визуализация как тепловой карты:
plt.figure(figsize=(8, 6)) # Опционально: устанавливаем размер фигуры

# Используем imshow() для создания тепловой карты:
heatmap = plt.imshow(matrix, cmap='viridis', interpolation='nearest') #cmap - цветовая карта, interpolation - сглаживание

# 3. Настройка графика:
plt.title('Тепловая карта случайной матрицы') # Добавляем заголовок
plt.xlabel('Столбец') # Подпись оси X
plt.ylabel('Строка') # Подпись оси Y
plt.colorbar(heatmap, label='Значение') # Добавляем цветовую шкалу (colorbar)

# 4. Отображение графика:
plt.show()
```

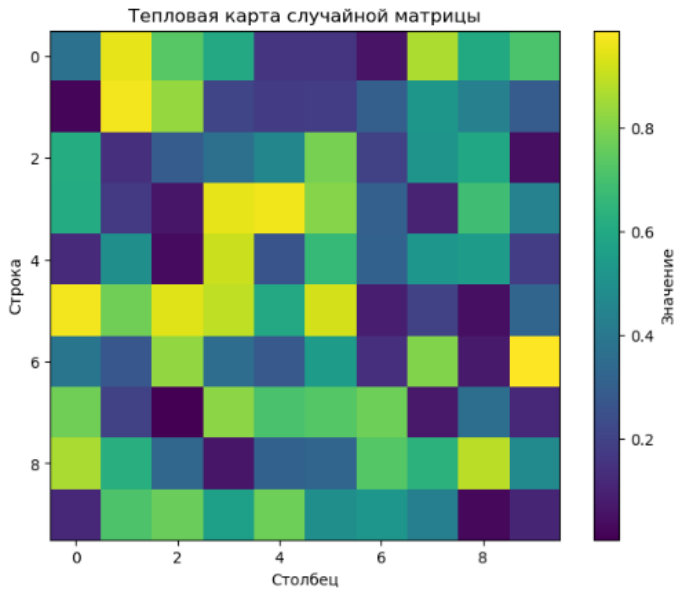

Рисунок 13. Тепловая карта

13. Выполнено индивидуальное задание 1.

```
[1]: import numpy as np
import matplotlib.pyplot as plt

# 1. Данные:
time = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
atoms = np.array([100, 85, 72, 60, 50, 42, 35, 30, 26, 22, 19])

# 2. Построение графика:
plt.figure(figsize=(8, 6))

# Строим график на обычном масштабе (для сравнения)
plt.plot(time, atoms, marker='o', linestyle='-', label='Линейный масштаб')

# 3. Настройка графика:
plt.xlabel('Время (часы)')
plt.ylabel('Количество атомов (млн)')
plt.title('Распад радиоактивного вещества')
plt.grid(True)
plt.legend()

# 4. Логарифмический масштаб по оси Y:
plt.yscale('log') # Переключаем Y в логарифмический масштаб
plt.ylim(10, 110) # Настраиваем пределы оси Y, чтобы данные хорошо отображались в логарифмическом масштабе
plt.show()
```

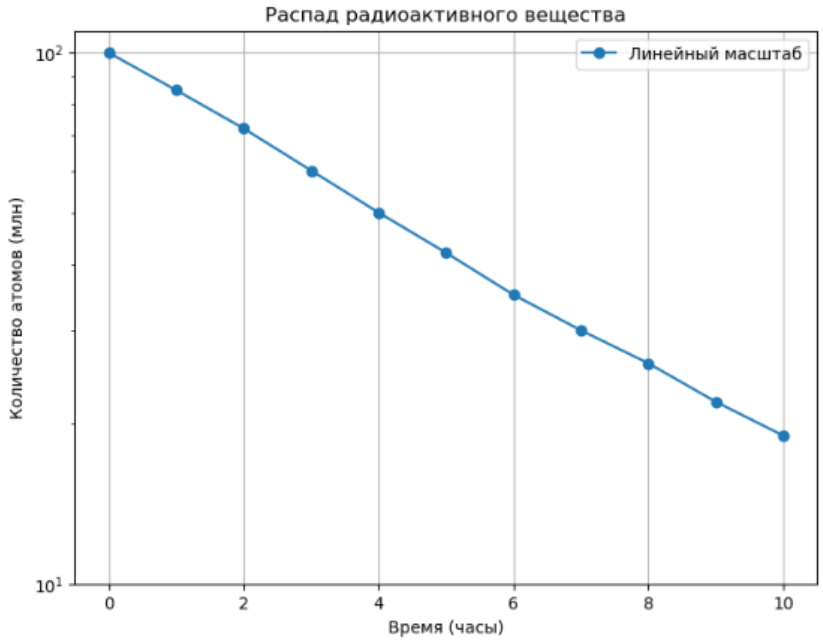

Рисунок 14. Линейный график

14. Выполнено индивидуальное задание 2.

```python
import matplotlib.pyplot as plt

# 1. Данные:
countries = ['США', 'Германия', 'Франция', 'Япония', 'Россия']
study_years = [4, 3.5, 3, 4.5, 4]

# 2. Создание горизонтальной столбчатой диаграммы:
plt.figure(figsize=(8, 6)) # Опционально: задаем размер графика

plt.barh(countries, study_years, color='skyblue', edgecolor='black') # barh() для горизонтальных столбцов

# 3. Настройка графика:
plt.xlabel('Среднее время обучения (лет)') # Подпись оси X
plt.ylabel('Страна') # Подпись оси Y
plt.title('Сравнение среднего времени обучения в университетах разных стран')
plt.grid(True, axis='x') # Отображаем сетку только по оси X

# Альтернативный вариант: показать значения непосредственно на столбцах
# for i, v in enumerate(study_years):
#     plt.text(v + 0.1, i, str(v), color='black', va='center') # Добавляем текст с количеством лет рядом со столбцом

# 4. Отображение графика:
plt.show()
```

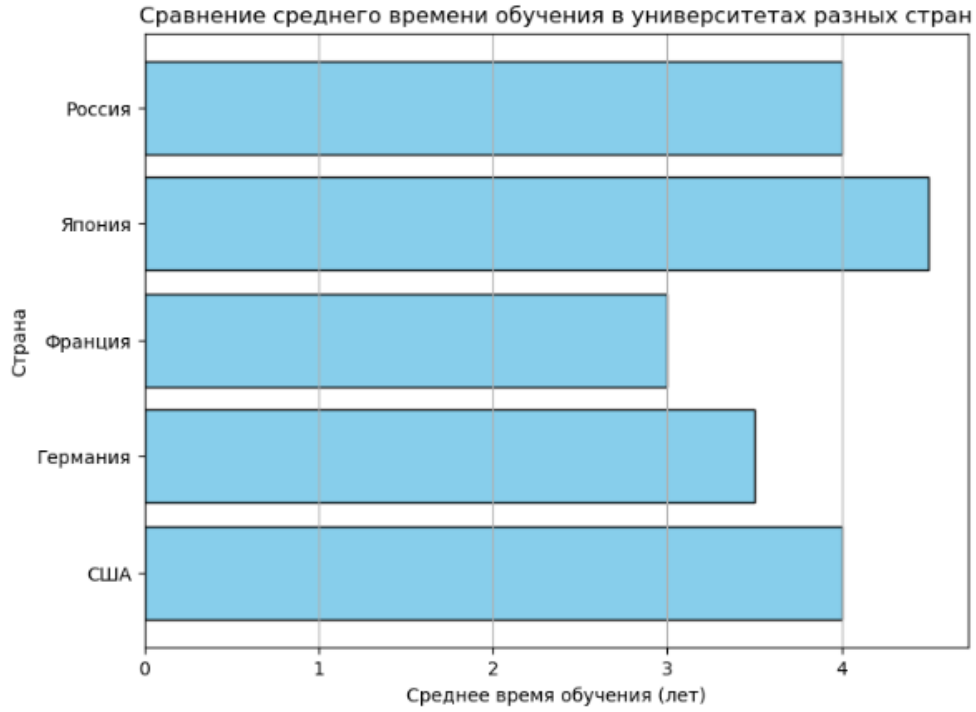

Рисунок 15. Столбчатая диаграмма

15. Выполнено индивидуальное задание 3.

```python
import numpy as np
from scipy import integrate

# 1. Определение функции:
def f(x):
    """Вычисляет значение функции f(x) = 3x^4 - 2x^3 + x^2 - 5"""
    return 3*x**4 - 2*x**3 + x**2 - 5

# 2. Вычисление интеграла:
a = -1   # Нижний предел интегрирования
b = 1    # Верхний предел интегрирования

# Используем функцию quad из scipy.integrate:
result = integrate.quad(f, a, b)

# 3. Вывод результата:
print("Значение интеграла:", result[0])

# 4. Вывод ошибки (опционально):
print("Оценка погрешности:", result[1])
```

```
Значение интеграла: -8.133333333333333
Оценка погрешности: 9.089316508790763e-14
```

Рисунок 16. Вычисление определенного интеграла

16. Выполнено индивидуальное задание 4.

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

# 1. Создание сетки координат:
x = np.linspace(-2*np.pi, 2*np.pi, 100)  # Увеличил число точек для более плавного графика
y = np.linspace(-2*np.pi, 2*np.pi, 100)
X, Y = np.meshgrid(x, y)

# 2. Вычисление значений Z:
Z = np.sin(X**2 + Y**2)

# 3. Создание 3D графика:
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# 4. Построение поверхности:
surf = ax.plot_surface(X, Y, Z, cmap=cm.viridis, linewidth=0, antialiased=False)

# Добавляем контурный график на плоскость z=0 (опционально):
cset = ax.contourf(X, Y, Z, zdir='z', offset=-1, cmap=cm.viridis)  # Добавляем контурный график

# 5. Настройка графика:
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('График функции f(x, y) = sin(x² + y²)')
fig.colorbar(surf) # Добавляем цветовую шкалу
ax.set_zlim(-1, 1)  # Устанавливаем пределы оси Z, чтобы видеть всю функцию

# 6. Отображение графика:
plt.show()
```

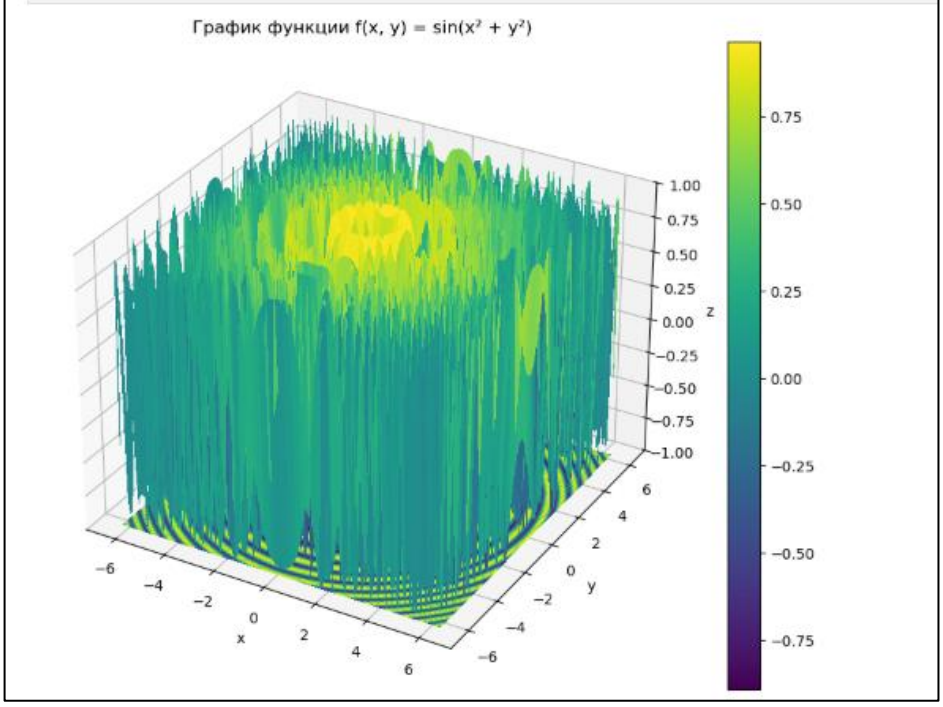

Рисунок 17. 3D график

17. Отправил изменения на репозиторий

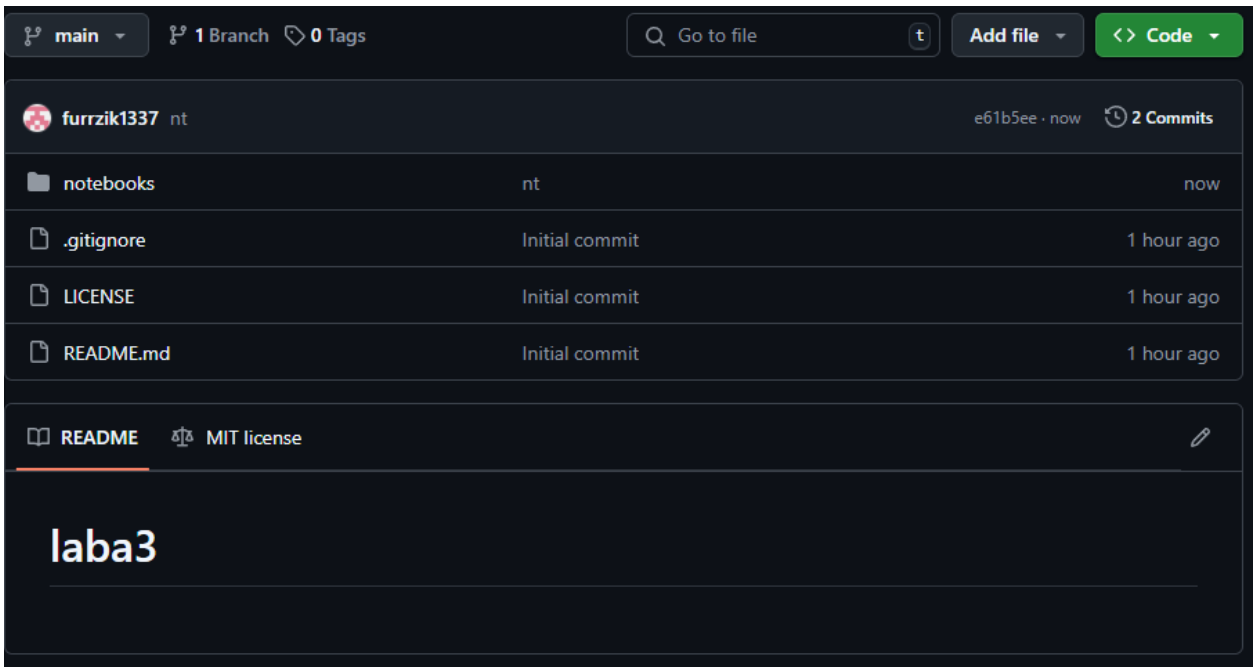

Рисунок 18. Репозиторий

Ответы на контрольные вопросы:

1. Как осуществляется установка пакета matplotlib?

Установка пакета matplotlib в Python обычно осуществляется с помощью менеджера пакетов pip (или pip3, если у вас установлено несколько версий Python).

2. Какая "магическая" команда должна присутствовать в ноутбуках Jupyter для корректного отображения графиков matplotlib

Для корректного отображения графиков matplotlib в ноутбуках Jupyter необходимо использовать команду: %matplotlib inline

3. Как отобразить график с помощью функции plot ?

Чтобы отобразить график с помощью функции plot из библиотеки Matplotlib, выполните следующие шаги:

1. Импортируйте необходимые библиотеки:

   import matplotlib.pyplot as plt

2. Подготовьте данные для графика:

   x = [1, 2, 3, 4, 5]

   y = [2, 3, 5, 7, 11]

3. Используйте функцию plot для построения графика:

```
plt.plot(x, y)
```

4. Добавьте заголовок и метки осей (по желанию):

```
plt.title('Пример графика')
plt.xlabel('Ось X')
plt.ylabel('Ось Y')
```

5. Отобразите график:

```
plt.show()
```

В результате вы получите график, отображающий данные из списков x и y.

4. Как отобразить несколько графиков на одном поле?

Чтобы отобразить несколько графиков на одном поле с помощью Matplotlib, вы можете использовать функцию plot несколько раз перед вызовом plt.show().

5. Какой метод Вам известен для построения диаграмм категориальных данных?

1. Столбчатая диаграмма (Bar Chart):

Столбчатые диаграммы хорошо подходят для отображения категориальных данных. Вы можете использовать plt.bar() для создания столбчатой диаграммы.

2. Гистограмма (Histogram):

Гистограммы используются для отображения распределения числовых данных, но их также можно адаптировать для категориальных данных, если у вас есть количество наблюдений в каждой категории.

3. Круговая диаграмма (Pie Chart):

Круговые диаграммы полезны для отображения долей категорий в общем объеме.

4. Скаттер-плот (Scatter Plot):

Хотя он чаще используется для числовых данных, его можно применять и для категориальных данных в случае, если вы хотите показать взаимосвязь между двумя категориальными переменными.

6. Какие основные элементы графика Вам известны?

1. Оси (Axes):

• Ось X: Горизонтальная ось, обычно отображает независимую переменную или категориальные данные.

• Ось Y: Вертикальная ось, обычно отображает зависимую переменную или значения.

2. Метки осей (Axis Labels):

• Названия, которые описывают, что представляют собой оси. Например, "Время" для оси X и "Температура" для оси Y.

3. Заголовок (Title):

• Краткое описание графика, которое помогает понять его содержание и цель.

4. Легенда (Legend):

• Объясняет символы, цвета или линии на графике, особенно если представлено несколько наборов данных.

5. Сетка (Grid):

• Линии, которые помогают визуально ориентироваться на графике и облегчают чтение значений.

6. Точки данных (Data Points):

• Конкретные значения, представленные на графике, которые могут быть обозначены маркерами или линиями.

7. Линии (Lines):

• Используются для соединения точек данных в линейных графиках или для обозначения трендов.

8. Шкала (Scale):

• Определяет диапазон значений, отображаемых на осях. Может быть линейной или логарифмической.

9. Аннотации (Annotations):

• Дополнительные заметки или комментарии на графике, которые помогают объяснить определенные аспекты данных.

10. Фон (Background):

• Цвет или текстура фона графика, который может улучшать визуальное восприятие

7. Как осуществляется управление текстовыми надписями на графике?

1. Добавление заголовка и меток осей:

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3], [4, 5, 6])
plt.title('Заголовок графика')  # Заголовок графика
plt.xlabel('Ось X')             # Метка оси X
plt.ylabel('Ось Y')             # Метка оси Y
plt.show()
```

2. Добавление легенды:

```
plt.plot([1, 2, 3], [4, 5, 6], label='Линия 1')
plt.plot([1, 2, 3], [6, 5, 4], label='Линия 2')
plt.legend()                    # Отображение легенды
plt.show()
```

3. Добавление аннотаций:

```
plt.plot([1, 2, 3], [4, 5, 6])
plt.text(2, 5, 'Аннотация', fontsize=12)  # Добавление текста на график
plt.show()
```

4. Настройка шрифта и цвета:

```
plt.title('Заголовок', fontsize=14, color='blue')  # Настройка заголовка
plt.xlabel('Ось X', fontsize=12, color='green')    # Настройка метки оси X
plt.ylabel('Ось Y', fontsize=12, color='red')      # Настройка метки оси Y
```

1. Добавление заголовка и меток осей:

```
plot([1, 2, 3], [4, 5, 6]);
title('Заголовок графика'); % Заголовок графика
xlabel('Ось X');          % Метка оси X
ylabel('Ось Y');          % Метка оси Y
```

2. Добавление легенды:

```
plot([1, 2, 3], [4, 5, 6], 'DisplayName', 'Линия 1');
hold on;
plot([1, 2, 3], [6, 5, 4], 'DisplayName', 'Линия 2');
legend show;              % Отображение легенды
hold off;
```

3. Добавление аннотаций

```
plot([1, 2, 3], [4, 5, 6]);
text(2, 5, 'Аннотация', 'FontSize', 12); % Добавление текста на график
```

4. Настройка шрифта и цвета:

```
title('Заголовок', 'FontSize', 14, 'Color', 'b'); % Настройка заголовка
xlabel('Ось X', 'FontSize', 12, 'Color', 'g');     % Настройка метки оси X
ylabel('Ось Y', 'FontSize', 12, 'Color', 'r');     % Настройка метки оси Y
```

8. Как осуществляется управление легендой графика?

1. Добавление легенды:

Используйте параметр label в функции plot() для каждой линии, а затем вызовите plt.legend() для отображения легенды.

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3], [4, 5, 6], label='Линия 1')
plt.plot([1, 2, 3], [6, 5, 4], label='Линия 2')
plt.legend()  # Отображение легенды
plt.show()
```

2. Настройка положения:

Вы можете указать положение легенды с помощью параметра loc:

```
plt.legend(loc='upper left')  # Положение в верхнем левом углу
```

3. Настройка внешнего вида:

Можно изменять шрифт, цвет и другие параметры через аргументы функции legend():

plt.legend(fontsize=10, frameon=False) # Настройка шрифта и отключение рамки

1. Добавление легенды:

Используйте параметр 'DisplayName' в функции plot() и затем вызовите legend show.

```
plot([1, 2, 3], [4, 5, 6], 'DisplayName', 'Линия 1');
hold on;
plot([1, 2, 3], [6, 5, 4], 'DisplayName', 'Линия 2');
legend show;  % Отображение легенды
hold off;
```

2. Настройка положения:

Вы можете установить положение легенды с помощью legend('Location', 'northeast'):

```
legend('Location', 'northeast');  % Положение в правом верхнем углу
```

3. Настройка внешнего вида:

Можно изменять шрифт и другие свойства через параметры legend:

```
legend('FontSize', 12, 'TextColor', 'b');  % Настройка шрифта и цвета текста
```

9. Как задать цвет и стиль линий графика?

1. Цвет линии:

Вы можете задать цвет линии, указывая его в формате RGB или используя предопределенные названия цветов:

```
plot([1, 2, 3], [4, 5, 6], 'Color', [1, 0, 0]); % Красный цвет в формате RGB
plot([1, 2, 3], [6, 5, 4], 'g'); % Зеленый цвет
```

2. Стиль линии:

Стиль линии можно указать в строке форматирования:

• '-': сплошная линия

• '--': пунктирная линия

• '-.': пунктирно-штриховая линия

• ':': точечная линия

Пример:

plot([1, 2, 3], [4, 5, 6], 'r--'); % Пунктирная красная линия

10. Как выполнить размещение графика в разных полях?

Для размещения графиков в разных полях (подграфиках) используется функция subplot.

11. Как выполнить построение линейного графика с помощью matplotlib?

1. Импортируем библиотеку: import matplotlib.pyplot as plt.

2. Подготавливаем данные для осей.

3. Используем функци. Plt.plot() для создания графика.

12. Как выполнить заливку области между графиком и осью? Между двумя графиками?

1. Заливка области между графиком и осью:

Используется fill_between(x,y), где x - значения по оси X, а y – значения по оси Y.

2. Заливка области между двумя графиками:

Используется fill_between(x, y1, y2), где y1 и y2 – значения двух графиков. Можно применять условие where для выбора области заливки.

13. Как выполнить выборочную заливку, которая удовлетворяет некоторому условию?

Используется fill_between(): Передаются массивы x, y1, y2 и логический массив where в функцию fill_between().

Можно дополнительно настроить цвет и прозрачность заливки с помощью параметров color и alpha.

14. Как выполнить двухцветную заливку?

1. **Определите условия**: Создайте два логических массива, каждый из которых будет указывать, какие области должны быть залиты разными цветами.

2. **Первый вызов** fill_between(): Залейте первую область с одним цветом, используя первое условие.

3. **Второй вызов** fill_between(): Залейте вторую область с другим цветом, используя второе условие.

4. **Настройте визуализацию**: Вы можете настроить цвета и прозрачность для каждой заливки.

15. Как выполнить маркировку графиков?

1. **Добавление заголовка**: Используйте plt.title() для задания заголовка графика.

2. **Подписи осей**: Используйте plt.xlabel() и plt.ylabel() для добавления подписей к осям X и Y соответственно.

3. **Легенда**: Для отображения легенды используйте plt.legend(), чтобы обозначить различные линии или данные на графике.

4. **Подписи точек данных**: Для индивидуальной маркировки точек данных используйте plt.text() или plt.annotate() для добавления текста рядом с конкретными точками на графике.

5. **Сетка**: Включите сетку с помощью plt.grid(), чтобы улучшить читаемость графика.

16. Как выполнить обрезку графиков?

Для обрезки графиков в Matplotlib используются методы set_xlim() и set_ylim() для задания пределов осей X и Y соответственно. Также можно использовать plt.axis() для одновременной настройки всех пределов.

17.Как построить ступенчатый график? В чем особенность ступенчатого графика?

Ступенчатый график в Matplotlib строится с помощью функции plt.step(). Особенность ступенчатого графика заключается в том, что он отображает изменения значений на горизонтальных отрезках, что позволяет лучше

визуализировать дискретные изменения во времени или других переменных. Это особенно полезно для представления кумулятивных данных или для отображения функций с резкими переходами.

18. Как построить стековый график? В чем особенность стекового графика?

Стековый график в Matplotlib строится с помощью функции plt.stackplot(). Особенность стекового графика заключается в том, что он отображает несколько наборов данных, складывая их друг на друга, что позволяет визуализировать общую величину и относительное значение каждого набора данных по сравнению с другими. Это удобно для анализа изменений во времени и распределения значений между категориями.

19. Как построить stem-график? В чем особенность stem-графика?

Stem-график (стебельчатый график) в Matplotlib строится с помощью функции plt.stem(). Особенность stem-графика заключается в том, что он отображает дискретные данные, представляя каждую точку как вертикальную линию (стебель), исходящую от оси X, с пометкой на верхней части, что позволяет легко видеть значения и их распределение. Это полезно для визуализации последовательностей данных и их изменений.

20. Как построить точечный график? В чем особенность точечного графика?

Точечный график (scatter plot) строится с помощью функции plt.scatter() в Matplotlib. Особенность точечного графика заключается в том, что он отображает индивидуальные точки данных на плоскости, позволяя визуализировать взаимосвязи между двумя переменными. Это помогает выявлять паттерны, тренды и корреляции в данных.

21. **Как осуществляется построение столбчатых диаграмм с помощью matplotlib?**

Построение столбчатых диаграмм (bar charts) в Matplotlib осуществляется с помощью функции plt.bar() или plt.barh() (для горизонтальных столбцов) из модуля matplotlib.pyplot.

22. **Что такое групповая столбчатая диаграмма? Что такое столбчатая диаграмма с errorbar элементом?**

Групповая столбчатая диаграмма — это способ визуализации данных, который позволяет сравнивать несколько групп категорий одновременно

Столбчатая диаграмма с полосами погрешностей— это тип столбчатой диаграммы, который отображает неопределенность или изменчивость данных с помощью "полос" (обычно линий) над каждым столбцом. Эти полосы показывают диапазон возможных значений вокруг среднего значения, представленного высотой столбца.

23. **Как выполнить построение круговой диаграммы средствами matplotlib?**

Для построения круговой диаграммы в Matplotlib используется функция plt.pie()

Основные шаги для построение:

Подготовить данные

Вызов функции plt.pie()

Настройка графика

Отображение графика

24. Что такое цветовая карта? Как осуществляется работа с цветовыми картами в matplotlib?

Цветовая карта — это отображение, которое связывает числа из заданного диапазона с цветами.

Как работает цветовая карта:

1. Данные: У вас есть набор данных, где каждый элемент представляет собой некоторое значение.

2. Нормализация: Значения данных обычно нормализуются в диапазон от 0 до 1 (или другой подходящий диапазон).

3. Отображение: Каждый нормализованный элемент данных отображается в цвет из цветовой карты. Цветовая карта определяет, какой цвет соответствует каждому значению от 0 до 1.

4. Визуализация: Полученные цвета используются для отображения данных

25. Как отобразить изображение средствами matplotlib?

Отображение изображения из массива NumPy:

```
import matplotlib.pyplot as plt
import numpy as np
# Создадим случайный массив для примера (замените на ваш массив изображения)
image_array = np.random.rand(100, 100, 3)  # 100x100 пикселей, 3 канала (RGB)
plt.imshow(image_array)  # Отображает массив как изображение
plt.axis('off')  # Убирает оси координат (по желанию)
plt.title("Изображение из массива NumPy")
plt.show()
```

26. Как отобразить тепловую карту средствами matplotlib?

Отображение тепловой карты с помощью imshow():

```
import matplotlib.pyplot as plt
import numpy as np
# Создадим случайные данные для тепловой карты (замените на ваши данные)
data = np.random.rand(10, 10)
plt.imshow(data, cmap='viridis', interpolation='nearest')  # Отображаем данные как тепловую карту
plt.colorbar(label='Значение')  # Добавляем цветовую шкалу
plt.title('Тепловая карта с imshow()')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

27. Как выполнить построение линейного 3D-графика с помощью matplotlib?

Для построения линейного 3D-графика в Matplotlib необходимо использовать модуль mplot3d

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
# 1. Создание данных
# Генерируем случайные данные для x, y и z координат
num_points = 100
z = np.linspace(0, 10, num_points)  # z координаты от 0 до 10
x = np.sin(z)  # x координаты, зависящие от z
y = n.cos(z)  # y координаты, зависящие от z
# 2. Создание фигуры и 3D-подграфика
fig = plt.figure(figsize=(10, 8))  # Создаем фигуру (окно)
ax = fig.add_subplot(projection='3d')  # Добавляем 3D-подграфик на фигуру
# 3. Построение графика
ax.plot(x, y, z, label='Линейный 3D-график')  # Строим линейный график
# 4. Настройка графика (необязательно)
ax.set_xlabel('X')  # Подпись оси X
ax.set_ylabel('Y')  # Подпись оси Y
ax.set_zlabel('Z')  # Подпись оси Z
ax.set_title('Пример линейного 3D-графика')  # Заголовок графика
ax.legend()  # Отображение легенды
#ax.view_init(elev=20, azim=45) # Настройка угла обзора (угол возвышения, азимут)
# 5. Отображение графика
plt.show()
```

28. Как выполнить построение точечного 3D-графика с помощью matplotlib?

Для построения точечного 3D-графика в matplotlib используется функция ax.scatter()

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
# 1. Создание данных
# Генерируем случайные данные для x, y и z координат
num_points = 100
x = np.random.rand(num_points)
y = np.random.rand(num_points)
z = np.random.rand(num_points)
# Дополнительно: задаем размеры и цвета точек
sizes = np.random.rand(num_points) * 100  # Случайные размеры точек (от 0 до 100)
colors = np.random.rand(num_points)  # Случайные значения для цветов (от 0 до 1)
# Или задаем массив с конкретными цветами, если это необходимо
# colors = ['red', 'blue', 'green'] * (num_points // 3)  # Повторяем список цветов

# 2. Создание фигуры и 3D-подграфика
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(projection='3d')
# 3. Построение точечного графика
# s - размер точки (scale)
# c - цвет точки (color)
# marker - форма маркера (например, 'o', '^', 's')
ax.scatter(x, y, z, s=sizes, c=colors, marker='o')
```

```
# 4. Настройка графика (необязательно)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Пример точечного 3D-графика')
# 5. Отображение графика
plt.show()
```

29. Как выполнить построение каркасной поверхности с помощью matplotlib?

Для построения каркасной поверхности в matplotlib используется функция ax.plot_wireframe()

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
# 1. Создание данных
# Создадим сетку координат
num_points = 50
x = np.linspace(-5, 5, num_points)
y = np.linspace(-5, 5, num_points)
X, Y = np.meshgrid(x, y)
# Определим функцию поверхности
Z = np.sin(np.sqrt(X**2 + Y**2))
# 2. Создание фигуры и 3D-подграфика
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(projection='3d')
# 3. Построение каркасной поверхности
ax.plot_wireframe(X, Y, Z, color='black')
# 4. Настройка графика (необязательно)
ax.set_xlabel('X')
ax.set_ylabel('Y')
```

```
ax.set_zlabel('Z')
ax.set_title('Каркасная поверхность')
# 5. Отображение графика
plt.show()
```

30. Для построения трехмерной поверхности с помощью matplotlib используется функция ax.plot_surface()

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
from matplotlib import cm  # Для цветовых карт
# 1. Создание данных
# Создадим сетку координат
num_points = 100
x = np.linspace(-5, 5, num_points)
y = np.linspace(-5, 5, num_points)
X, Y = np.meshgrid(x, y)
# Определим функцию поверхности
Z = np.sin(np.sqrt(X**2 + Y**2))
# 2. Создание фигуры и 3D-подграфика
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(projection='3d')

# 3. Построение поверхности
# cmap - цветовая карта
# rstride - шаг по строкам
# cstride - шаг по столбцам
ax.plot_surface(X, Y, Z, cmap=cm.viridis, rstride=1, cstride=1)
# 4. Настройка графика (необязательно)
ax.set_xlabel('X')
ax.set_ylabel('Y')
```

```
ax.set_zlabel('Z')
ax.set_title('3D Поверхность')
# 5. Добавление цветовой шкалы (colorbar)
fig.colorbar(ax.plot_surface(X, Y, Z, cmap=cm.viridis))
# 6. Отображение графика
plt.show()
```

Вывод: в ходе лабораторной работы были приобретены навыки работы с базовыми возможностями библиотеки matplotlib

Ссылка на GitHub: https://github.com/furrzik1337/laba2