

Исследование библиотеки matplotlib с помощью языка программирования Python

Ёлгин Роман Николаевич

Приамурский государственный университет имени Шолом-Алейхема
Студент

Добрынин Александр Андреевич

Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация

В данной статье рассмотрен метод исследования библиотеки matplotlib с помощью языка программирования Python. Результатом исследования является визуализация наборов данных.

Ключевые слова: matplotlib, Python, Pycharm, Jupyter Notebook.

Exploring the Matplotlib library using the Python programming language

Yolgin Roman Nikolaevich

Sholom-Aleichem Priamursky State University
Student

Dobrynin Alexander Andreevich

Sholom-Aleichem Priamursky State University
Student

Abstract

This article discusses a method for researching the matplotlib library using the Python programming language. The result of the research is the visualization of data sets.

Keywords: matplotlib, Python, Pycharm, Jupyter Notebook.

1 Введение

1.1 Актуальность

Библиотека matplotlib позволяет визуализировать данные для наглядного и упрощенного подсчёта элементов в базе данных.

1.2 Обзор исследований

В своей работе Е.В. Ценина и Е.В. Слепенкова представили алгоритм, направленный на повышение точности прогнозирования потребности в готовой продукции [1]. В.А. Мельникова и Д.А. Медведев объясняют, почему Python является наиболее распространённым языком для анализа данных,

приводя ряд примеров и аргументов в поддержку этого утверждения [2]. Л.А. Ельшин, А.М. Гильманов и В.В. Бандеров разработали модель машинного обучения, способную прогнозировать курс криптовалюты на один день вперёд с минимальными ошибками [3]. А.В. Петрухин и А.С. Стешенко исследуют тему «Компьютерная визуализация биржевых данных о динамике фондового рынка» [4], уделяя особое внимание эффективным методам построения модулей визуализации данных. В.В. Волков в своей статье «Использование программных средств поддержки планирования эксперимента для оптимизации параметров генетического алгоритма аппроксимации» [5] описывает процесс экспериментальной оптимизации скорости сходимости генетического алгоритма, осуществлённый с помощью специализированных научных программ для поддержки планирования экспериментов.

1.3 Цель исследования

Цель исследования – рассмотреть библиотеку matplotlib, которая была создана для визуализации данных, наглядного и упрощённого подсчёта элементов в базе данных.

2 Материалы и методы

Исследование Библиотеки matplotlib основано на языке программирования Python. При помощи программы Pycharm написан код, предоставляющий пользователю комплекс средств для написания кода и визуальный отладчик.

3 Результаты и обсуждения

3.1 Работа с библиотекой Matplotlib

Функция plot создает 2D график данных в Y по сравнению с соответствующими значениями в X (рис.1).

```
import matplotlib.pyplot as plt  
plt.plot([1, 2, -6, 0, 4])  
plt.show()
```

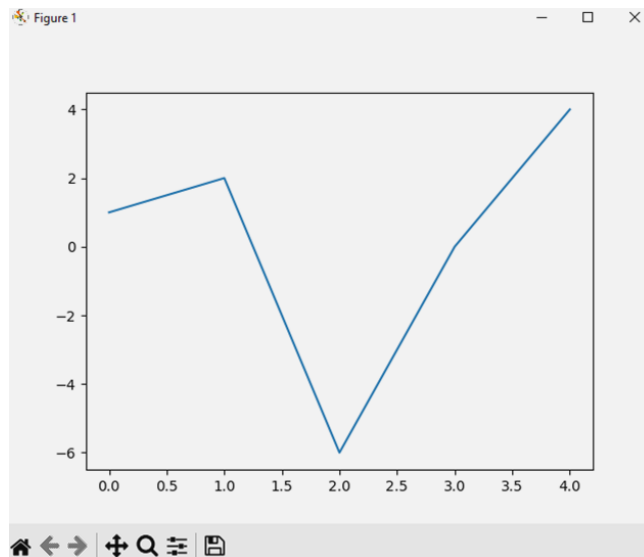
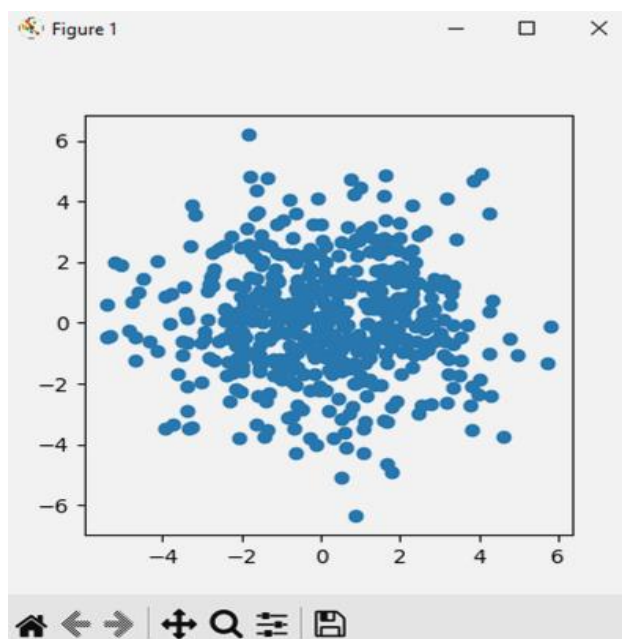


Рисунок 1. График функции plot

Функция `scatter` создает график рассеивания с круговыми маркерами в местоположениях, заданных векторами `x` и `y`. (рис.2).

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(4, 4))
ax = fig.add_subplot()
x = np.random.normal(0, 2, 500)
y = np.random.normal(0, 2, 500)
ax.scatter(x, y)
plt.show()
```

Рисунок 2. График функции `scatter` «рассеяния с маркерами»

Функция `bar` создает столбчатый график с одной панелью для каждого элемента в `y` (рис.3).

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(6, 4))
ax = fig.add_subplot()
y = np.random.normal(0, 2, 500)
ax.grid()
x = ['H{i+1}' for i in range(10)]
y = np.random.randint(1, 5, len(x))
ax.bar(x, y)
plt.show()
```

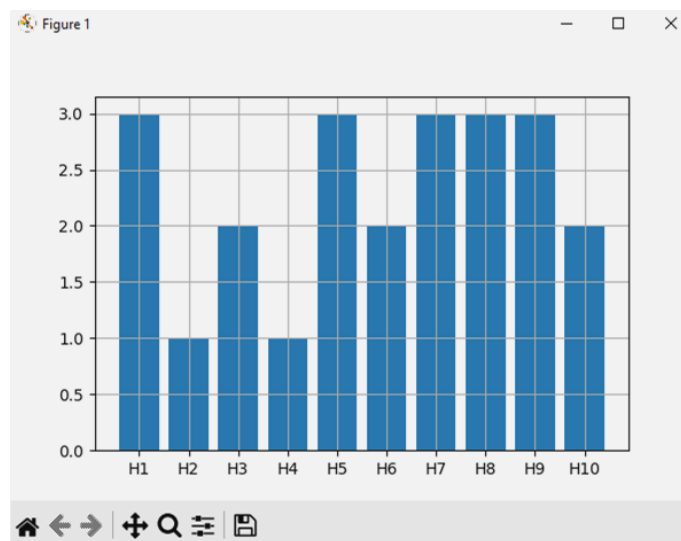


Рисунок 3. График функции `bar`

Функция `stem` отображает последовательность данных, `Y`, как основы, которые простираются от базовой линии вдоль оси `X`. Значения данных обозначаются кругами, завершающими каждый стержень (рис.4).

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(4, 4))
ax = fig.add_subplot()
x = np.arange(-np.pi, np.pi, 0.3)
ax.stem(x, np.cos(x))
plt.show()
```

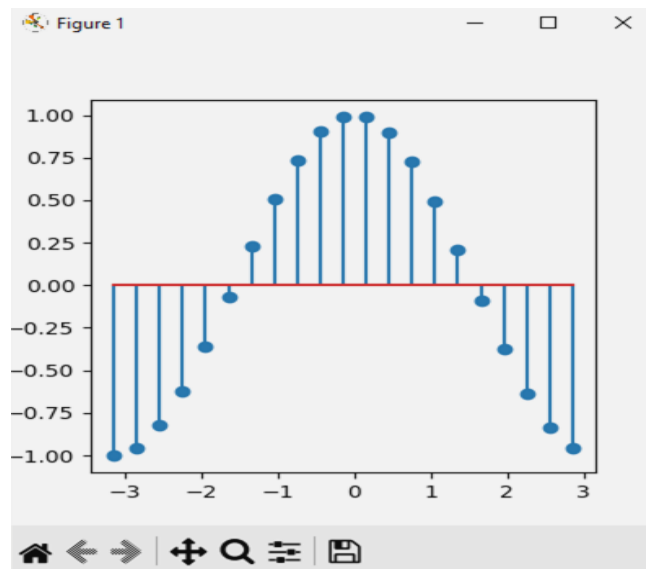


Рисунок 4. График функции stem

Функция `step` автоматически определяет временные шаги и длительность симуляции на основе системной динамики (рис.5).

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(4, 4))
ax = fig.add_subplot()
x = np.arange(0, 10)
ax.step(x, x)
ax.grid()
plt.show()
```

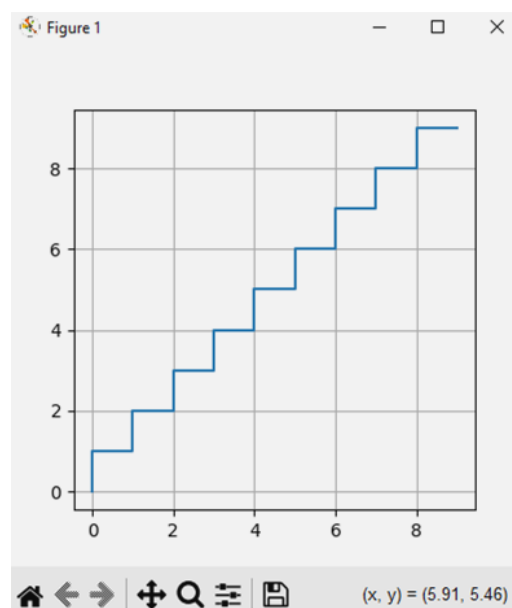


Рисунок 5. График функции step

Функция `fill_between` заполняет область между двумя горизонтальными кривыми. Кривые определяются точками x и y . Это создает один или несколько многоугольников, описывающих заполненную область (рис.6).

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(-2*np.pi, 2*np.pi, 0.1)
y = np.cos(x)
plt.plot(x, y)
plt.fill_between(x, y)
plt.show()
```

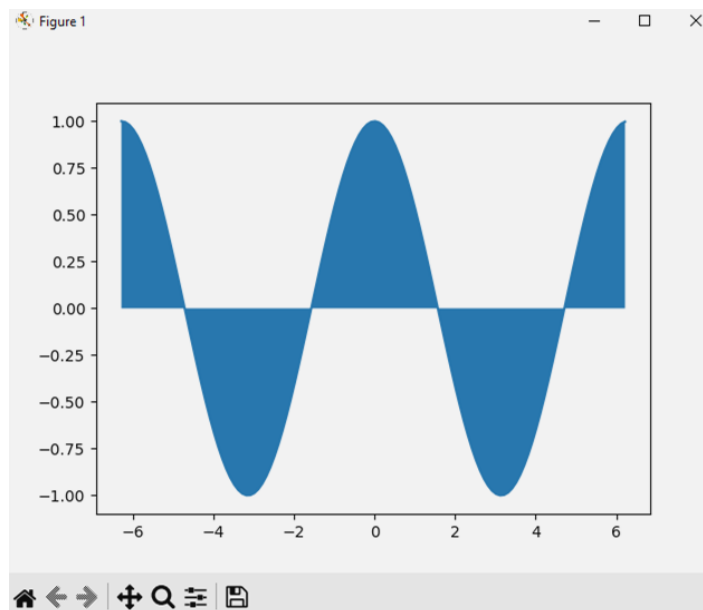


Рисунок 6. График функции `fill_between`

Функция `imshow` отображает изображение, хранящееся в каталоге нашей программы (рис.7).

```
from PIL import Image
img = Image.open('img.png')
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(6, 4))
ax = fig.add_subplot()
ax.imshow(img)
plt.show()
```

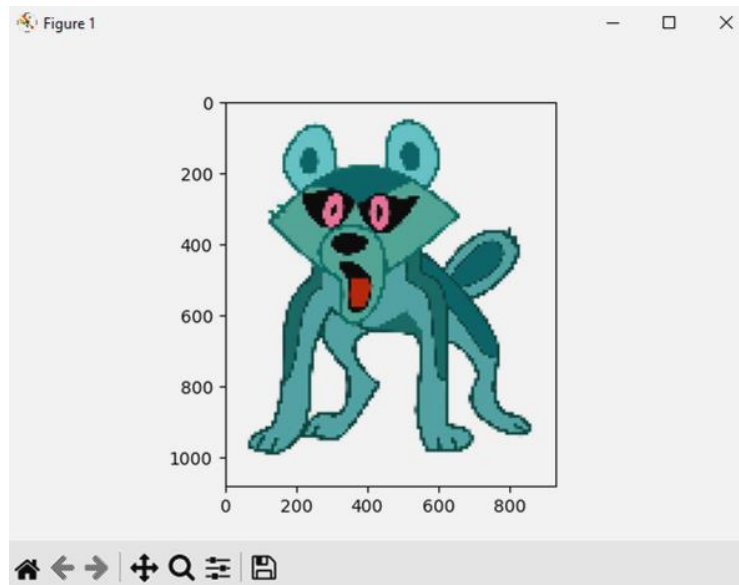


Рисунок 7. График функции imshow показ изображения «img.png»

Функция pcolormesh создаёт цветной график прямоугольной сетки, x и y используются для определения углов четырехугольников. (рис.8).

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(6, 4))
ax = fig.add_subplot()
data = np.random.randint(0, 255, (10, 10))
ax.pcolormesh(data)
plt.show()
```

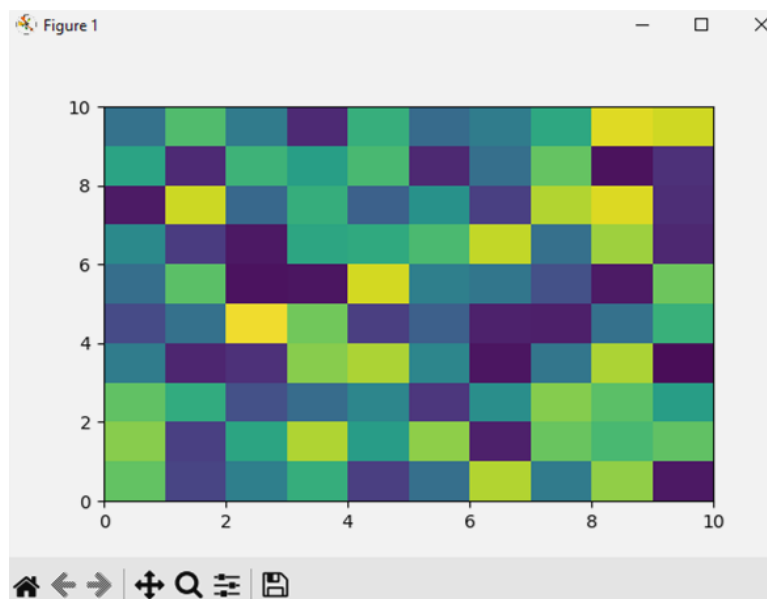


Рисунок 8. График функции pcolormesh «цветовая сетка»

Функция `contour` рисует двумерные линии уровня для массива данных Z , определяющих поверхность в трехмерном пространстве без учета диапазона изменения координат x и y (рис.9).

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(7, 4))
ax_3d = fig.add_subplot(projection='3d')
x = np.arange(-2 * np.pi, 2 * np.pi, 0.2)
y = np.arange(-2 * np.pi, 2 * np.pi, 0.2)
xgrid, ygrid = np.meshgrid(x, y)
zgrid = np.sin(xgrid) * np.sin(ygrid) / (xgrid * ygrid)
ax[0].contour(xgrid, ygrid, zgrid)
```

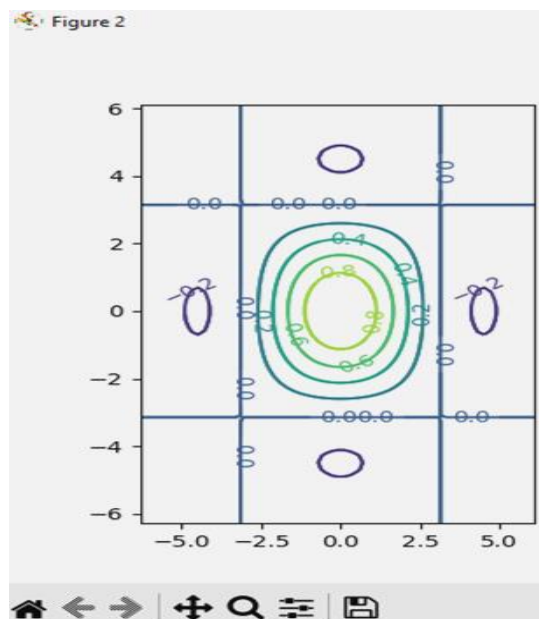


Рисунок 9. График функции `contour`

Функция `contourf` создает заполненный контурный график, содержащий изолинии матричного Z , где Z содержит значения высоты на x и y плоскость (рис.10).

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(7, 4))
ax_3d = fig.add_subplot(projection='3d')
x = np.arange(-2 * np.pi, 2 * np.pi, 0.2)
y = np.arange(-2 * np.pi, 2 * np.pi, 0.2)
xgrid, ygrid = np.meshgrid(x, y)
zgrid = np.sin(xgrid) * np.sin(ygrid) / (xgrid * ygrid)
ax[1].contourf(xgrid, ygrid, zgrid)
```

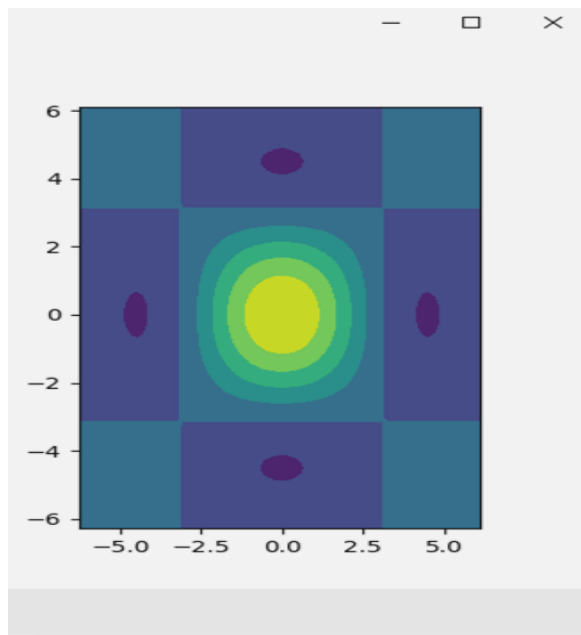


Рисунок 10. График функции contourf

Функция barbs строит двухмерное поле ветровых зубцов (рис.11).

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-5, 5, 5)
X, Y = np.meshgrid(x, x)
U, V = 12 * X, 12 * Y
data = [(-1.5, .5, -6, -6),
        (1, -1, -46, 46),
        (-3, -1, 11, -11),
        (1, 1.5, 80, 80),
        (0.5, 0.25, 25, 15),
        (-1.5, -0.5, -5, 40)]
data = np.array(data, dtype=[('x', np.float32), ('y', np.float32),
                              ('u', np.float32), ('v', np.float32)])
fig1, axs1 = plt.subplots(nrows=2, ncols=2)
axs1[0, 0].barbs(X, Y, U, V)
axs1[0, 1].barbs(
    data['x'], data['y'], data['u'], data['v'], length=8, pivot='middle')
axs1[1, 0].barbs(
    X, Y, U, V, np.sqrt(U ** 2 + V ** 2), fill_empty=True, rounding=False,
    sizes=dict(emptybarb=0.25, spacing=0.2, height=0.3))
axs1[1, 1].barbs(data['x'], data['y'], data['u'], data['v'], flagcolor='r',
    barbcOLOR=['b', 'g'], flip_barb=True,
    barb_increments=dict(half=10, full=20, flag=100))
masked_u = np.ma.masked_array(data['u'])
masked_u[4] = 1000
masked_u[4] = np.ma.masked
```

```
fig2, ax2 = plt.subplots()
ax2.barbs(data['x'], data['y'], masked_u, data['v'], length=8, pivot='middle')
plt.show()
```

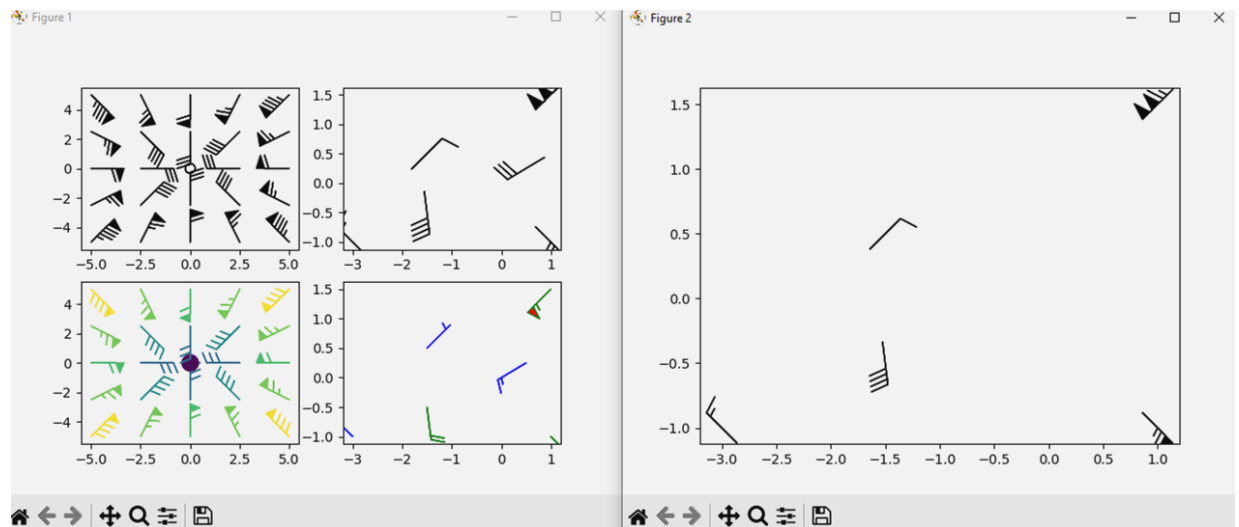


Рисунок 11. График функции barbs «Ветровые шипы»

Функция `quiver` строит двумерное поле стрелок. `X` и `Y` определяют местоположение стрелок, `U`, `V` определяют направления стрелок, а `C` опционально задает цвет (рис.12).

```
import matplotlib.pyplot as plt
import numpy as np
X = np.arange(-10, 10, 1)
Y = np.arange(-10, 10, 1)
U, V = np.meshgrid(X, Y)
fig, ax = plt.subplots()
q = ax.quiver(X, Y, U, V)
ax.quiverkey(q, X=0.3, Y=1.1, U=10,
             label='Quiver key, length = 10', labelpos='E')
plt.show()
```

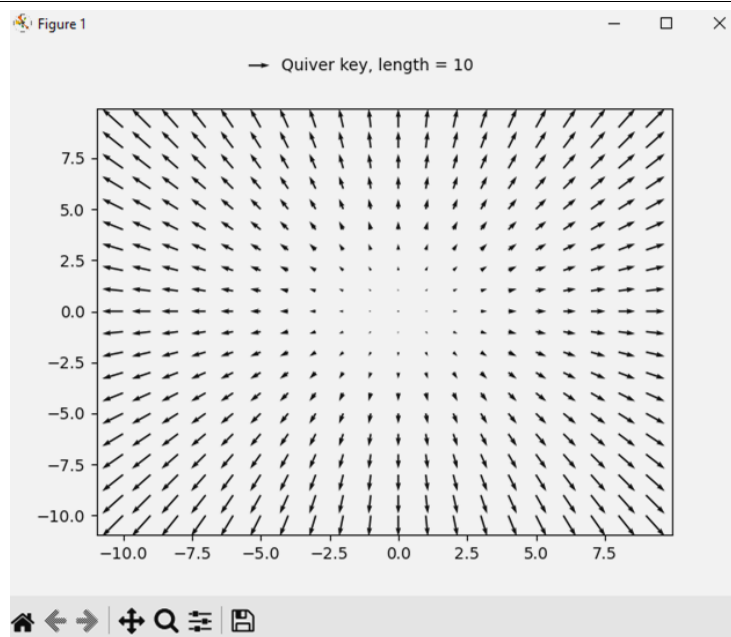


Рисунок 12. График функции quiver «Двумерное поле стрелок»

Функция streamplot это тип двумерного графика, который в основном используется для отображения потока жидкости и градиентов двумерного поля. На рисунках изображены: «Изменение цвета вдоль линии потока», «Изменение плотности линий тока», «Изменение ширины линии вдоль линии тока», «Управление начальными точками линий тока», «Оптимизирует пропуск замаскированных областей и значений NaN», «Непрерывные линии потока даже при превышении лимита линий в пределах одной ячейки сетки» (рис.13).

```
import matplotlib.pyplot as plt
import numpy as np
w = 3
Y, X = np.mgrid[-w:w:100j, -w:w:100j]
U = -1 - X**2 + Y
V = 1 + X - Y**2
speed = np.sqrt(U**2 + V**2)
fig, axs = plt.subplots(3, 2, figsize=(7, 9), height_ratios=[1, 1, 2])
axs = axs.flat
axs[0].streamplot(X, Y, U, V, density=[0.5, 1])
axs[0].set_title('Varying Density')
strm = axs[1].streamplot(X, Y, U, V, color=U, linewidth=2, cmap='autumn')
fig.colorbar(strm.lines)
axs[1].set_title('Varying Color')
lw = 5*speed / speed.max()
axs[2].streamplot(X, Y, U, V, density=0.6, color='k', linewidth=lw)
axs[2].set_title('Varying Line Width')
seed_points = np.array([[-2, -1, 0, 1, 2, -1], [-2, -1, 0, 1, 2, 2]])
strm = axs[3].streamplot(X, Y, U, V, color=U, linewidth=2,
```

```

cmap='autumn', start_points=seed_points.T)
fig.colorbar(strm.lines)
axs[3].set_title('Controlling Starting Points')

axs[3].plot(seed_points[0], seed_points[1], 'bo')
axs[3].set(xlim=(-w, w), ylim=(-w, w))
mask = np.zeros(U.shape, dtype=bool)
mask[40:60, 40:60] = True
U[:20, :20] = np.nan
U = np.ma.array(U, mask=mask)
axs[4].streamplot(X, Y, U, V, color='r')
axs[4].set_title('Streamplot with Masking')
axs[4].imshow(~mask, extent=(-w, w, -w, w), alpha=0.5, cmap='gray',
               aspect='auto')
axs[4].set_aspect('equal')
axs[5].streamplot(X, Y, U, V, broken_streamlines=False)
axs[5].set_title('Streamplot with unbroken streamlines')
plt.tight_layout()
plt.show()

```

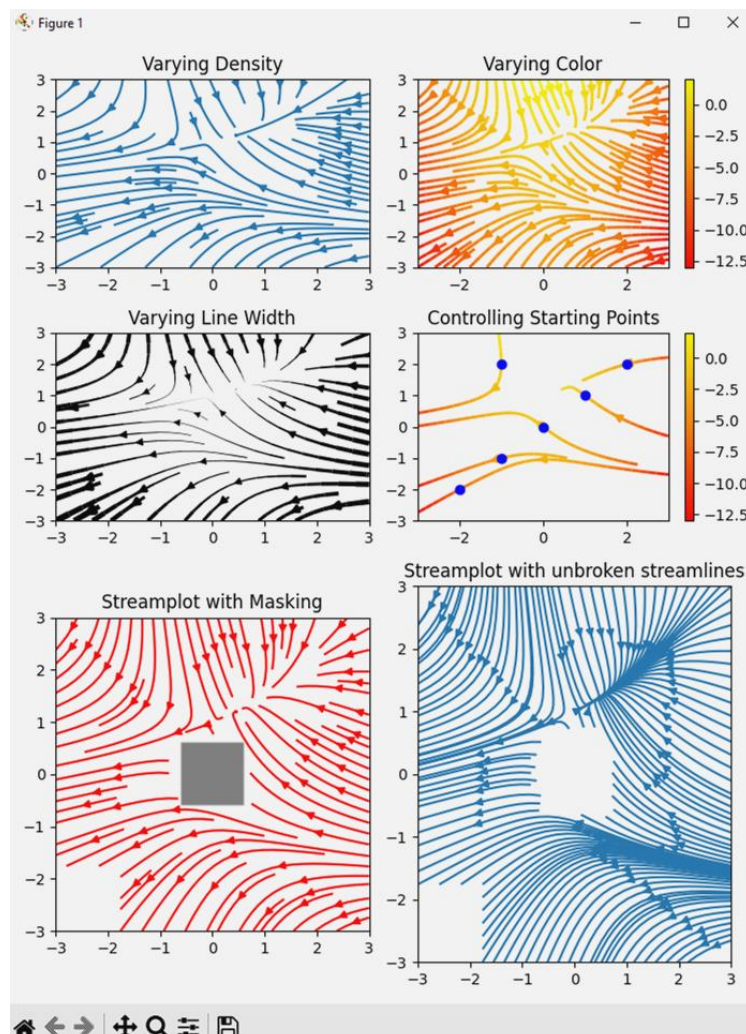


Рисунок 13. Графики функции streamplot

Функция `hist` группирует данные в `x` и подсчитывает количество значений в каждой группировке, выводит гистограмму исходя из данных значений. (рис.14).

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(6, 4))
ax = fig.add_subplot()
y = np.random.normal(0, 2, 500)
ax.hist(y)
ax.grid()
plt.show()
```

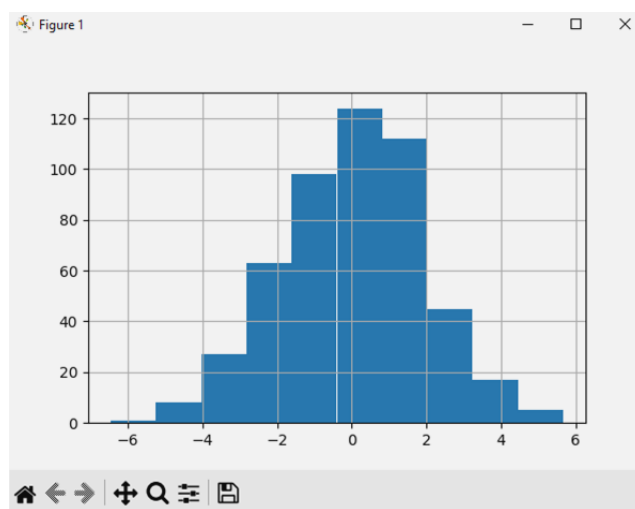


Рисунок 14. График функции `hist`

Функция `boxplot` это ящичный график, также известный как график Вискера, создается для отображения сводной информации о наборе значений данных, имеющих такие свойства, как минимум, первый квартиль, медиана, третий квартиль и максимум (рис.15).

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cbook
np.random.seed(19680801)
data = np.random.randn(20, 3)
fig, (ax1, ax2) = plt.subplots(1, 2)
# single boxplot call
ax1.boxplot(data, tick_labels=['A', 'B', 'C'],
             patch_artist=True, boxprops={'facecolor': 'bisque'})
# separate calculation of statistics and plotting
stats = cbook.boxplot_stats(data, labels=['A', 'B', 'C'])
ax2.bxp(stats, patch_artist=True, boxprops={'facecolor': 'bisque'})
plt.show()
```

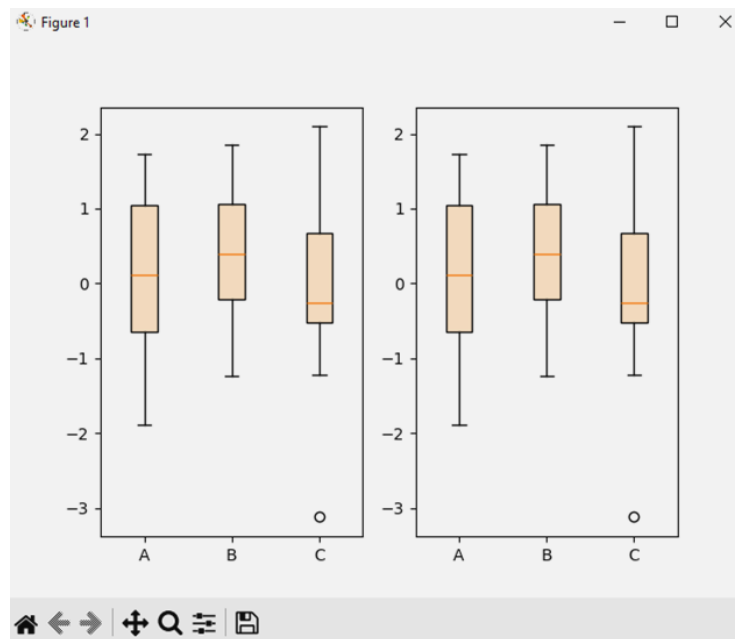
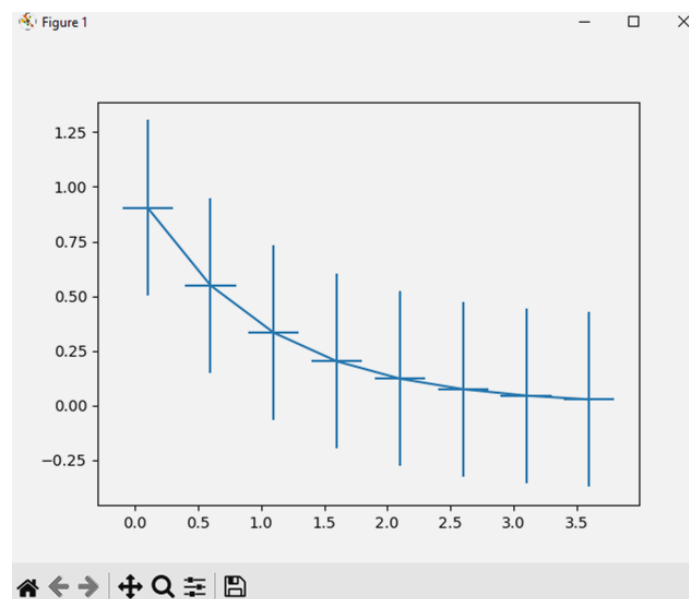


Рисунок 15. График функции boxplot

Функция `errorbar` строит график зависимости y от x в виде линий и/или маркеров с прикрепленными планками погрешностей. (рис.16).

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0.1, 4, 0.5)
y = np.exp(-x)
fig, ax = plt.subplots()
ax.errorbar(x, y, xerr=0.2, yerr=0.4)
plt.show()
```

Рисунок 16. График функции `errorbar`

Функция `violinplot` Создаёт график скрипки для каждого столбца набора данных или каждого вектора в последовательном наборе данных . Каждая заполненная область расширяется, чтобы представить весь диапазон данных, с дополнительными линиями на среднем значении, медиане, минимуме, максимуме и указанных пользователем квантилях. (рис.17).

```
import matplotlib.pyplot as plt
import numpy as np
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(9, 4))
np.random.seed(19680801)
all_data = [np.random.normal(0, std, 100) for std in range(6, 10)]
axs[0].violinplot(all_data,
                  showmeans=False,
                  showmedians=True)
axs[0].set_title('Violin plot')
for ax in axs:
    ax.yaxis.grid(True)
    ax.set_xticks([y + 1 for y in range(len(all_data))],
                  labels=['x1', 'x2', 'x3', 'x4'])
    ax.set_xlabel('Four separate samples')
    ax.set_ylabel('Observed values')
plt.show()
```

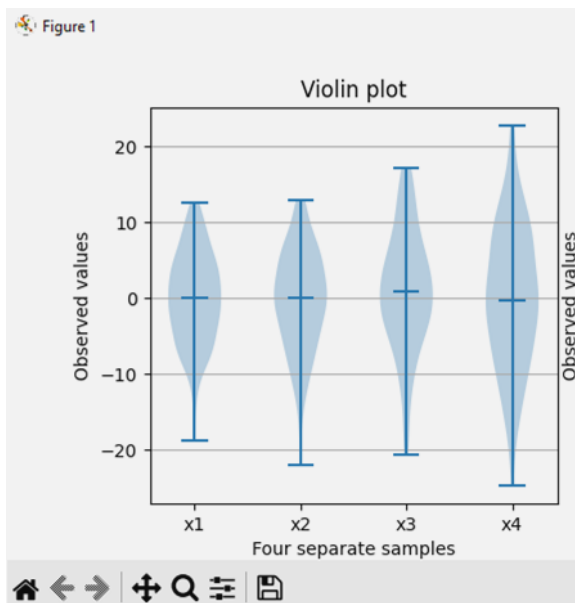


Рисунок 17. График функции `violinplot`

Функция `eventplot` стоит одинаковые параллельные линии в указанных позициях. Этот тип диаграммы обычно используется в нейронауке для представления нейронных событий, где ее обычно называют пиковым растром, точечным растром или растровой диаграммой. (рис.18).

```

import matplotlib.pyplot as plt
import numpy as np
import matplotlib
matplotlib.rcParams['font.size'] = 8.0
np.random.seed(19680801)
data1 = np.random.random([6, 50])
colors1 = [f'C{i}' for i in range(6)]
lineoffsets1 = [-15, -3, 1, 1.5, 6, 10]
linelengths1 = [5, 2, 1, 1, 3, 1.5]
fig, axs = plt.subplots(2, 2)
axs[0, 0].eventplot(data1, colors=colors1, lineoffsets=lineoffsets1,
                    linelengths=linelengths1)
axs[1, 0].eventplot(data1, colors=colors1, lineoffsets=lineoffsets1,
                    linelengths=linelengths1, orientation='vertical')
data2 = np.random.gamma(4, size=[60, 50])
colors2 = 'black'
lineoffsets2 = 1
linelengths2 = 1
axs[0, 1].eventplot(data2, colors=colors2, lineoffsets=lineoffsets2,
                    linelengths=linelengths2)
axs[1, 1].eventplot(data2, colors=colors2, lineoffsets=lineoffsets2,
                    linelengths=linelengths2, orientation='vertical')
plt.show()

```

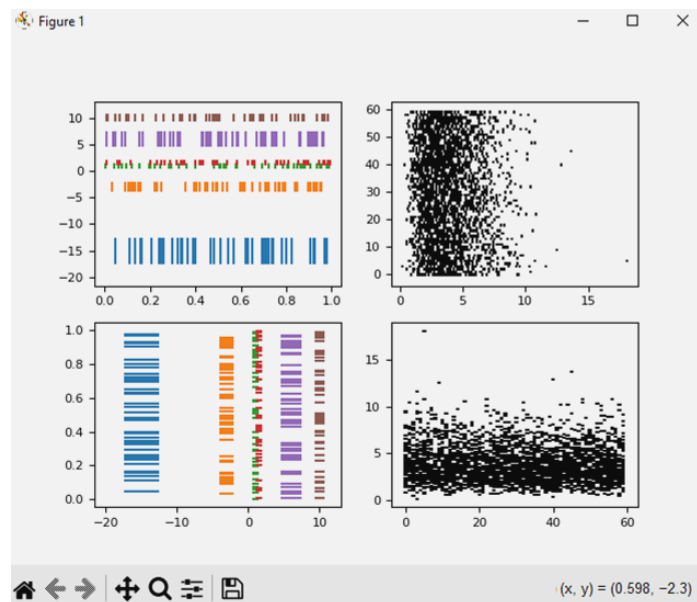


Рисунок 18. График функции eventplot

Функция `hist2d` строит двумерную гистограмму образцов x и y . Значения в x строятся вдоль первого измерения, а значения в y строятся вдоль второго измерения. (рис.19).

```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('_mpl-gallery-nogrid')
np.random.seed(1)
x = np.random.randn(5000)
y = 1.2 * x + np.random.randn(5000) / 3
fig, ax = plt.subplots()
ax.hist2d(x, y, bins=(np.arange(-3, 3, 0.1), np.arange(-3, 3, 0.1)))
ax.set(xlim=(-2, 2), ylim=(-3, 3))
plt.show()
```

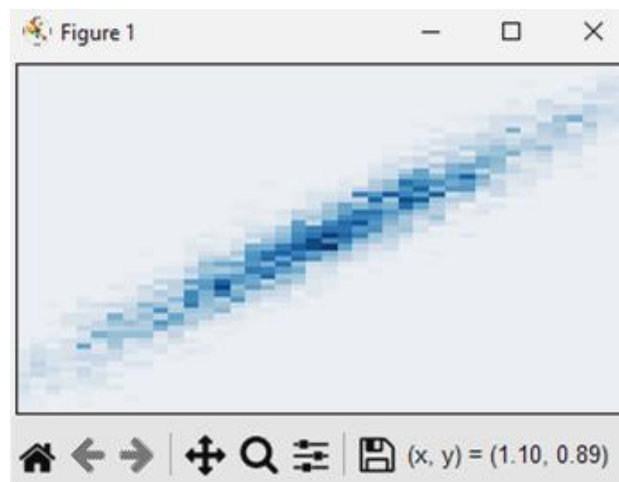


Рисунок 19. График функции hist2d

Функция hexbin представляет собой двумерную гистограмму, в которой ячейки представляют собой шестиугольники, а цвет представляет собой количество точек данных в каждой ячейке (рис.20).

```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('_mpl-gallery-nogrid')
np.random.seed(1)
x = np.random.randn(5000)
y = 1.2 * x + np.random.randn(5000) / 3
fig, ax = plt.subplots()
ax.hexbin(x, y, gridsize=20)
ax.set(xlim=(-2, 2), ylim=(-3, 3))
plt.show()
```

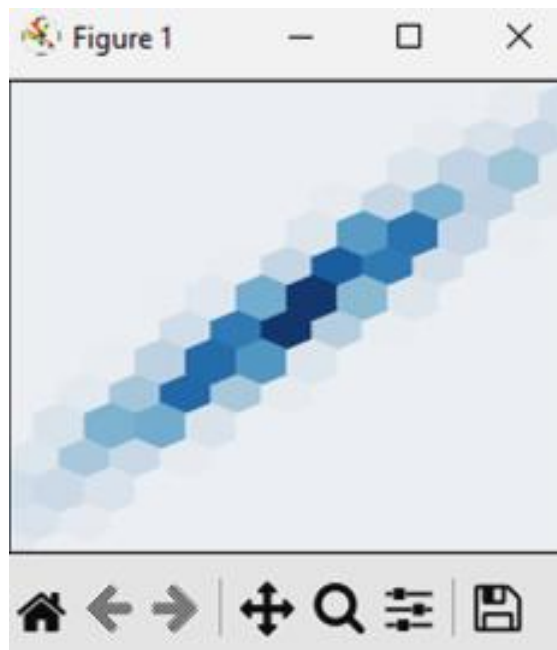
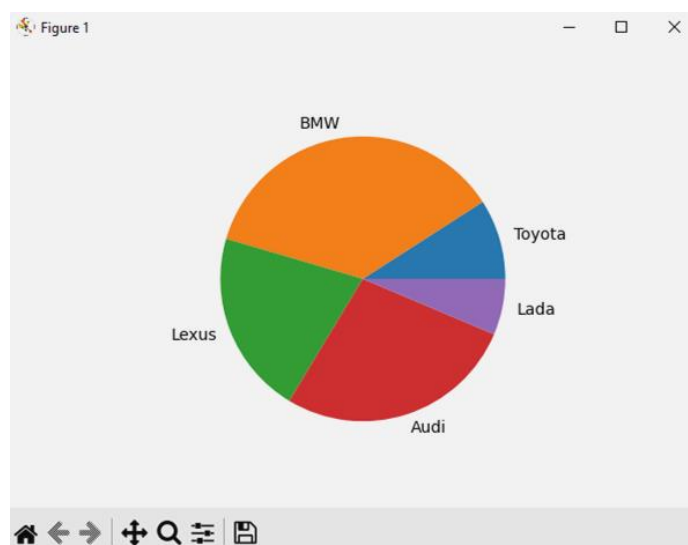


Рисунок 20. График функции hexbin

Функция `pie` используется для рисования круговых диаграмм, добавляя метки `labels` на круговую диаграмму (рис.21).

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(6, 4))
ax = fig.add_subplot()
y = np.random.normal(0, 2, 500)
ax.grid()
vals = [10, 40, 23, 30, 7]
labels = ['Toyota', 'BMW', 'Lexus', 'Audi', 'Lada']
ax.pie(vals, labels=labels)
plt.show()
```

Рисунок 21. График функции `pie`

Функция `tricontour` создаёт рисунок контурных линий случайных значений в диапазоне $[-2\pi; 2\pi]$ (рис.22).

```
import numpy as np
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
x = np.random.rand(100) * 4*np.pi - 2*np.pi
y = np.random.rand(100) * 4*np.pi - 2*np.pi
z = np.sin(x) * np.sin(y) / (1+np.abs(x * y))
xgrid, ygrid = np.meshgrid(x, y)
zgrid = np.sin(xgrid) * np.sin(ygrid) / (xgrid * ygrid)
c1 = ax.tricontour(x, y, z, cmap='plasma')
c1.clabel(colors='k', fmt='%0.2f')
plt.show()
```



Рисунок 22. График функции `tricontour`

Функция `tricontourf` создаёт рисунок заполненных контуров случайных значений в диапазоне $[-2\pi; 2\pi]$ (рис.23).

```
import numpy as np
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
x = np.random.rand(100) * 4*np.pi - 2*np.pi
y = np.random.rand(100) * 4*np.pi - 2*np.pi
z = np.sin(x) * np.sin(y) / (1+np.abs(x * y))
xgrid, ygrid = np.meshgrid(x, y)
zgrid = np.sin(xgrid) * np.sin(ygrid) / (xgrid * ygrid)
c1 = ax.tricontourf(x, y, z, cmap='plasma')
plt.show()
```

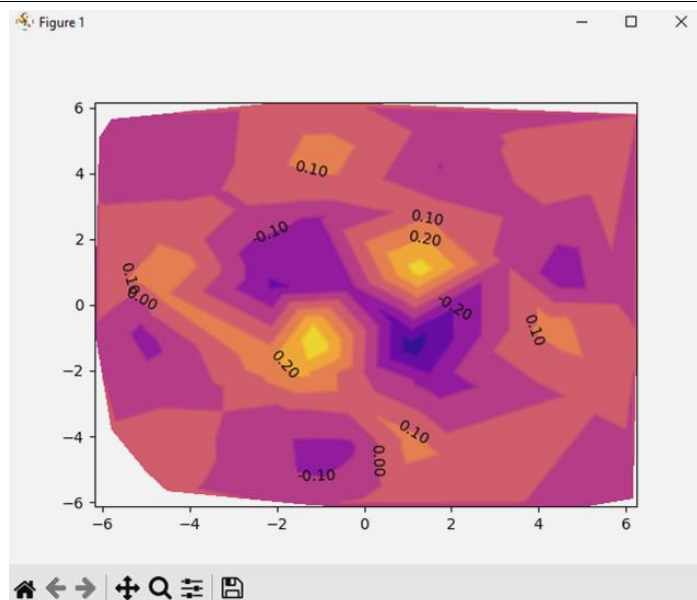


Рисунок 23. График функции tricontourf

Функция `tricolor` Создаёт псевдоцветной график неструктурированной треугольной сетки с заданными значениями x и y (рис.24).

```
import matplotlib.pyplot as plt
import numpy as np
xy = np.asarray([
    [-0.101, 0.872], [-0.080, 0.883], [-0.069, 0.888], [-0.054, 0.890],
    [-0.045, 0.897], [-0.057, 0.895], [-0.073, 0.900], [-0.087, 0.898],
    [-0.090, 0.904], [-0.069, 0.907], [-0.069, 0.921], [-0.080, 0.919],
    [-0.073, 0.928], [-0.052, 0.930], [-0.048, 0.942], [-0.062, 0.949],
    [-0.054, 0.958], [-0.069, 0.954], [-0.087, 0.952], [-0.087, 0.959],
    [-0.080, 0.966], [-0.085, 0.973], [-0.087, 0.965], [-0.097, 0.965],
    [-0.097, 0.975], [-0.092, 0.984], [-0.101, 0.980], [-0.108, 0.980],
    [-0.104, 0.987], [-0.102, 0.993], [-0.115, 1.001], [-0.099, 0.996],
    [-0.101, 1.007], [-0.090, 1.010], [-0.087, 1.021], [-0.069, 1.021],
    [-0.052, 1.022], [-0.052, 1.017], [-0.069, 1.010], [-0.064, 1.005],
    [-0.048, 1.005], [-0.031, 1.005], [-0.031, 0.996], [-0.040, 0.987],
    [-0.045, 0.980], [-0.052, 0.975], [-0.040, 0.973], [-0.026, 0.968],
    [-0.020, 0.954], [-0.006, 0.947], [ 0.003, 0.935], [ 0.006, 0.926],
    [ 0.005, 0.921], [ 0.022, 0.923], [ 0.033, 0.912], [ 0.029, 0.905],
    [ 0.017, 0.900], [ 0.012, 0.895], [ 0.027, 0.893], [ 0.019, 0.886],
    [ 0.001, 0.883], [-0.012, 0.884], [-0.029, 0.883], [-0.038, 0.879],
    [-0.057, 0.881], [-0.062, 0.876], [-0.078, 0.876], [-0.087, 0.872],
    [-0.030, 0.907], [-0.007, 0.905], [-0.057, 0.916], [-0.025, 0.933],
    [-0.077, 0.990], [-0.059, 0.993]])
x, y = np.rad2deg(xy).T
triangles = np.asarray([
    [67, 66, 1], [65, 2, 66], [ 1, 66, 2], [64, 2, 65], [63, 3, 64],
    [60, 59, 57], [ 2, 64, 3], [ 3, 63, 4], [ 0, 67, 1], [62, 4, 63],
```

```

[57, 59, 56], [59, 58, 56], [61, 60, 69], [57, 69, 60], [ 4, 62, 68],
[ 6, 5, 9], [61, 68, 62], [69, 68, 61], [ 9, 5, 70], [ 6, 8, 7],
[ 4, 70, 5], [ 8, 6, 9], [56, 69, 57], [69, 56, 52], [70, 10, 9],
[54, 53, 55], [56, 55, 53], [68, 70, 4], [52, 56, 53], [11, 10, 12],
[69, 71, 68], [68, 13, 70], [10, 70, 13], [51, 50, 52], [13, 68, 71],
[52, 71, 69], [12, 10, 13], [71, 52, 50], [71, 14, 13], [50, 49, 71],
[49, 48, 71], [14, 16, 15], [14, 71, 48], [17, 19, 18], [17, 20, 19],
[48, 16, 14], [48, 47, 16], [47, 46, 16], [16, 46, 45], [23, 22, 24],
[21, 24, 22], [17, 16, 45], [20, 17, 45], [21, 25, 24], [27, 26, 28],
[20, 72, 21], [25, 21, 72], [45, 72, 20], [25, 28, 26], [44, 73, 45],
[72, 45, 73], [28, 25, 29], [29, 25, 31], [43, 73, 44], [73, 43, 40],
[72, 73, 39], [72, 31, 25], [42, 40, 43], [31, 30, 29], [39, 73, 40],
[42, 41, 40], [72, 33, 31], [32, 31, 33], [39, 38, 72], [33, 72, 38],
[33, 38, 34], [37, 35, 38], [34, 38, 35], [35, 37, 36]])
xmid = x[triangles].mean(axis=1)
ymid = y[triangles].mean(axis=1)
x0 = -5
y0 = 52
zfaces = np.exp(-0.01 * ((xmid - x0) * (xmid - x0) +
                          (ymid - y0) * (ymid - y0)))
fig3, ax3 = plt.subplots()
ax3.set_aspect('equal')
tpc = ax3.tripcolor(x, y, triangles, facecolors=zfaces, edgecolors='k')
fig3.colorbar(tpc)
ax3.set_title('tripcolor of user-specified triangulation')
ax3.set_xlabel('Longitude (degrees)')
ax3.set_ylabel('Latitude (degrees)')
plt.show()

```

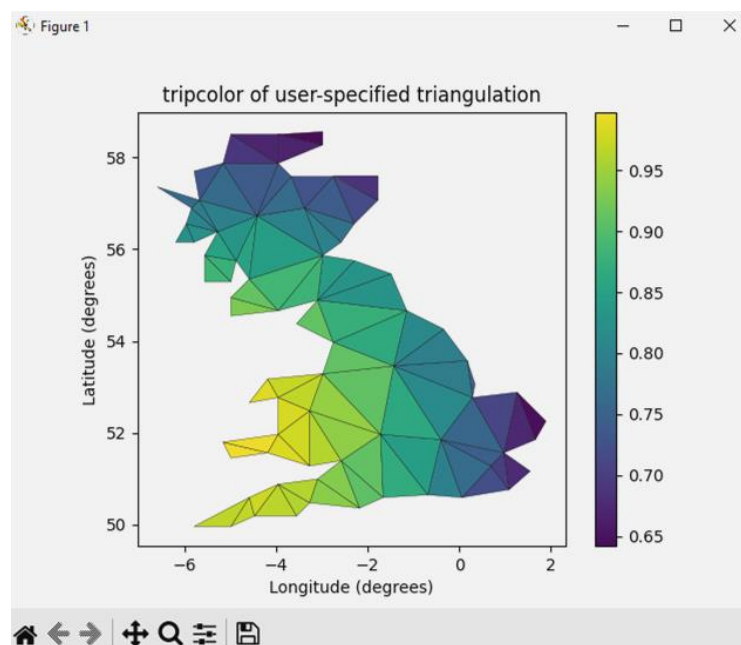


Рисунок 24. График функции tripcolor

Функция triplot создаёт неструктурированную треугольную сетку в виде линий и/или маркеров в случайном диапазоне значений. (рис.25):

```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('_mpl-gallery-nogrid')
np.random.seed(1)
x = np.random.uniform(-3, 3, 256)
y = np.random.uniform(-3, 3, 256)
z = (1 - x/2 + x**5 + y**3) * np.exp(-x**2 - y**2)
fig, ax = plt.subplots()
ax.triplot(x, y)
ax.set(xlim=(-3, 3), ylim=(-3, 3))
plt.show()
```

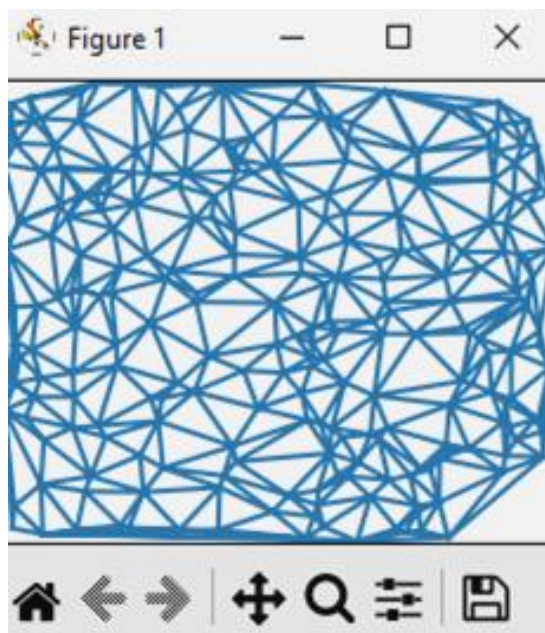


Рисунок 25. График функции triplot неструктурированных треугольных сеток

4 Выводы

Таким образом, с помощью языка программирования Python исследована библиотеки matplotlib, которая позволяет визуализировать данные для наглядного и упрощенного подсчёта элементов в базе данных.

Библиографический список

1. Ценина Е.В., Слепенкова Е.В. Разработка алгоритма машинного обучения для прогнозирования потребности в запасах готовой продукции // Риск: ресурсы, информация, снабжение, конкуренции. 2023. №. 3. С. 57-65.
2. Мельникова В. А., Медведев Д. А. Анализ больших данных с использованием Python //Труды Братского государственного университета.

- Серия: Естественные и инженерные науки. 2019. Т. 1. С. 46-49.
3. Ельшин Л.А., Гильманов А.М., Бандеров В.В. Разработка прогноза динамики курса криптовалюты на основе теории машинного обучения // Финансовая аналитика: проблемы и решения. 2020. Т. 13. №. 1(351). С. 97-113.
 4. Петрухин А.В., Стешенко А.С. Компьютерная визуализация биржевых данных о динамике фондового рынка // Известия Волгоградского государственного технического университета 2015. №6. С. 124-129.
 5. Волков В.В. Использование программных средств поддержки планирования эксперимента для оптимизации параметров генетического алгоритма аппроксимации // Вестник Донского государственного технического университета. 2010. №10. С. 487-491.