Визуализация данных в Matplotlib

Для графического отображения данных в Python широко применяется библиотека matplotlib. Она предоставляет множество инструментов для создания двумерных графиков различного типа - от простых линейных до сложных диаграмм и гистограмм. Matplotlib отличается простотой использования и позволяет получать качественные и наглядные изображения.

```
import matplotlib.pyplot as plt
```

1 Диаграммы

Для построения линейного графика используется функция *plot()*, со следующей сигнатурой:

plot([x], y, [fmt], *, data=None, **kwargs)
plot([x], y, [fmt], [x2], y2, [fmt2], …, **kwargs)

Если вызвать функцию *plot()* с одним аргументом – вот так: *plot(y)*, то мы получим график, у которого по оси ординат (ось *y*) будут отложены значения из переданного списка, по по оси абсцисс (ось *x*) – индексы элементов массива.

Рассмотрим аргументы функции *plot()*:

– x, x2, …: array. Набор данных для оси абсцисс первого, второго и т.д. графика.

– y, y2, …: array. Набор данных для оси ординат первого, второго и т.д. графика.

– fmt: str. Формат графика, задается в виде строки: ‘[marker][line][color]’.

– **kwargs – свойства класса *Line2D*, которые предоставляют доступ к большому количеству настроек внешнего вида графика, отметим наиболее полезные (табл.1).

Таблица 1 – Настройки внешнего вида графика

Свойство	Тип	Описание
alpha	float	Прозрачность
color или c	color	Цвет
fillstyle	{‘full’, ‘left’, ‘right’, ‘bottom’, ‘top’, ‘none’}	Стиль заливки
label	object	Текстовая метка
linestyle или ls	{‘-‘, ‘--‘, ‘-.’, ‘:’, ‘’, (offset, on-off-seq), …}	Стиль линии
linewidth или lw	float	Толщина линии
marker	matplotlib.markers	Стиль маркера
markeredgecolor или mec	color	Цвет границы маркера

markeredgewidth или mew	float	Толщина границы маркера
markerfacecolor или mfc	color	Цвет заливки маркера
markersize или ms	float	Размер маркера

Построение графика в matplotlib можно начать с задания набора точек. Для этого нужно отдельно указать координаты по осям x и y. Поскольку matplotlib хорошо взаимодействует с массивами из numpy, удобно использовать именно их для хранения координат.

```
x = np.arange(5, 16)
```

Рисунок 1 – Задаем агрумент

Построим, например, график функции $y=x2$ y=x2. В этом случае массив из координат точек на оси y будет состоять из квадратов значений координат на оси x (рис. 2):

Рисунок 2 – Вычисление значений функции

Передадим эти два массива в функцию plt.plot чтобы получить желаемый график (рис. 3):

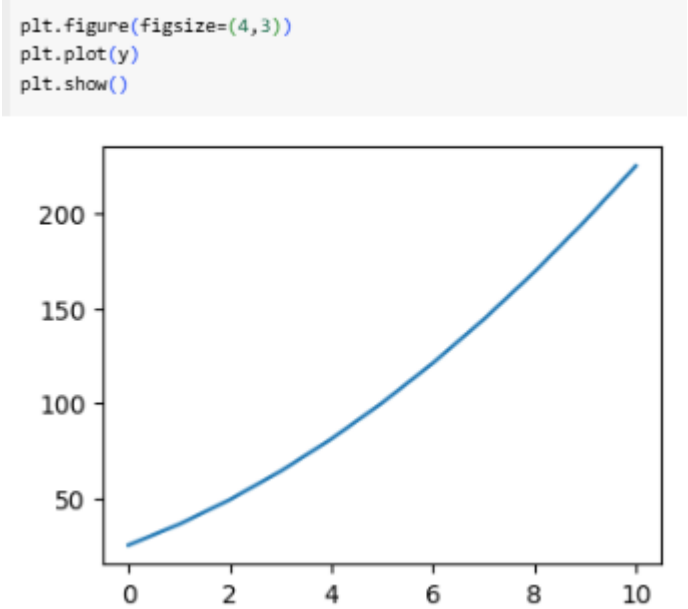

Рисунок 53 – Построение графика

Кстати, если координаты по оси x представляют собой последовательность натуральных чисел, которая начинается с 0, то значение x в функцию можно и не передавать.

Графики, которые мы построили здесь, называются *линейными диаграммами*: они получены путём соединения точек прямыми линиями.

Рассмотрим также другой тип диаграмм - *точечные диаграммы* (или *диаграммы разброса*) (рис.54):

```
plt.figure(figsize=(4,3))
plt.scatter(x, y)
plt.show()
```

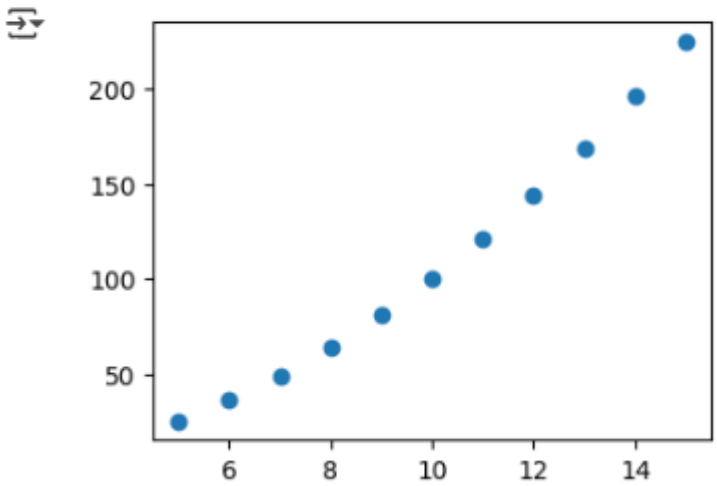

Рисунок 4 – Построение точечной диаграммы

2 Масштаб

На всех построенных нами диаграммах был использован *линейный масштаб*. При таком масштабе расстояния между точками на оси пропорциональны разностям между значениями в этих точках. Однако, бывают случаи, когда рассматриваемые значения отличаются на порядки. В этом случае, линейный масштаб не всегда позволяет уловить разницу между более близкими значениями.

Например, рассмотрим несколько списков (рис.5):

```
In [8]: planets = ['Mercury', 'Venus', 'Earth', 'Mars',
                   'Jupiter', 'Saturn', 'Uranus', 'Neptune']

        masses = [0.055274, 0.815, 1.0, 0.107,
                  317.8, 95.0, 14.6, 17.147]
```

Рисунок 5 – Массы планет по отношеню к Земле

В первом списке указаны имена планет Солнечной системы, а во втором - их массы относительно массы планеты Земля.

Построим график этих значений:

```
In [9]: plt.plot(planets, masses)

        plt.show()
```

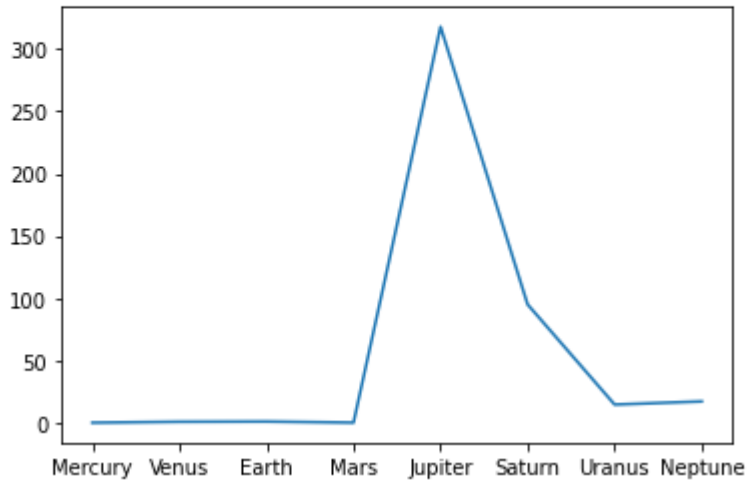

Рисунок 6 – График масс планет

Поскольку относительные массы первых четырёх планет отличаются друг от друга не так сильно, как, например, от относительной массы Юпитера, зависимость между их относительными массами уловить по этому графику невозможно. В этом случае, полезным оказывается *логарифмический масштаб*. Применим его по оси y (рис.7):

```
In [10]: plt.plot(planets, masses)
         plt.yscale(value="log")

         plt.show()
```

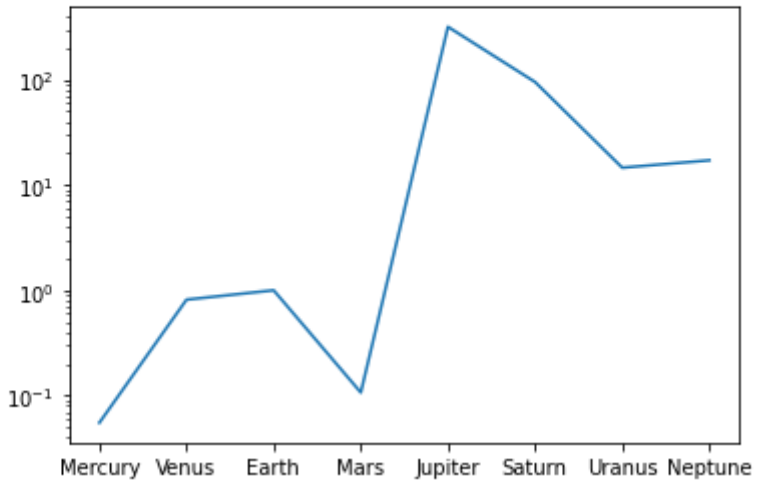

Рисунок 7 – Логарифмический масштаб

Логарифмический масштаб означает, что на выбранной оси (в нашем случае, на оси y) соседние откладываемые деления будут отличаться в 10 раз.

Например, при таком масштабе расстояние между числами 10 и 100 будет таким же, как расстояние между числами 100 и 1000.

Кроме массивов numpy, библиотека matplotlib также поддерживает структуры данных из библиотеки pandas: Series и DataFrame.

Ещё один типом диаграммы - *столбчатая диаграмма* (рис.8):

```
In [13]: planets_info.plot(kind="bar")

         plt.show()
```

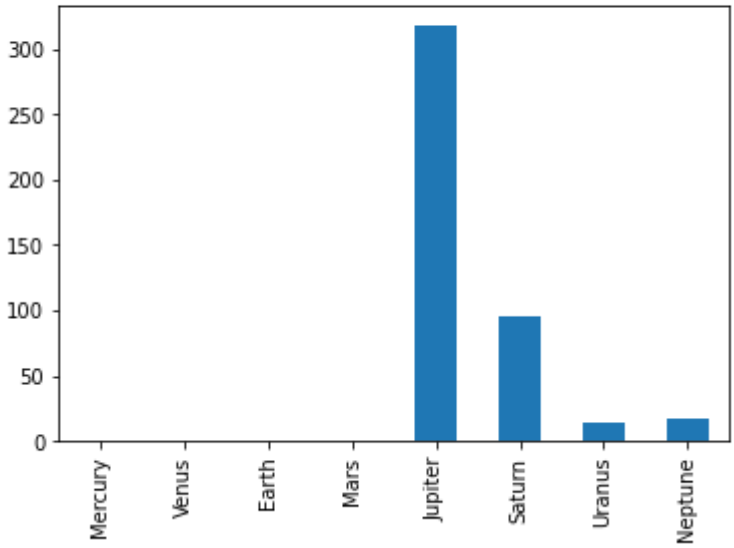

Рисунок 8 – Столбчатая диаграмма

Гистограммы позволяют визуализировать, каким образом распределена некоторая величина в выборке (9).

```
In [16]: print(y)

         hist_info = plt.hist(y, bins=3)

         plt.show()
```

```
[  0   1   4   9  16  25  36  49  64  81 100]
```

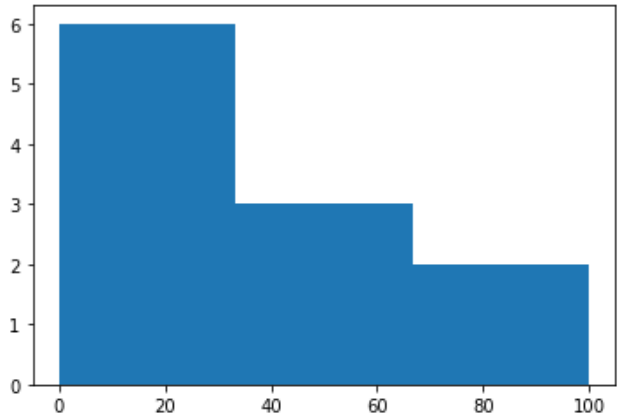

Рисунок 9 – Гистограмма

В предыдущем примере использовалась функцию plt.hist, но и сохранили её результат в переменной hist_info. Рассмотрим, что возвращает функция plt.hist:

```
In [17]: print(hist_info)

(array([6., 3., 2.]), array([  0.        ,  33.33333333,  66.66666667, 100.
]), <BarContainer object of 3 artists>)
```

Рисунок 10 – Функция hist_info

Как мы видим, она возвращает tuple из трёх объектов. Первый объект - это массив из значений каждого бина. Тут содержится число элементов поданного массива, которые содержатся в каждом из бинов: 6 элементов в первом, 3 во втором, 2 в третьем.

Второй массив содержит границы каждого бина. Например, границы первого бина - от 0 до 33.3, второго - от 33.3 до 66.6 и т.д.

Вот несколько полезных параметров функции plt.hist:

– edgecolor - цвет границы бинов. Полезен, если несколько соседних бинов имеют одинаковую высоту.

– ec - синоним edgecolor.

– orientation - ориентация гистограммы. С помощью этого параметра можно расположить гистограмму горизонтально.

4 Выведение дополнительной информации на график

Построим график функции y=x^2:

```
x = np.linspace(-5, 5, 101)
y = x ** 2

plt.plot(x, y)

plt.show()
```

Заголовок и названия осей. С помощью функции plt.title можно добавить заголовок к графику. Данная функция принимает параметры для управления шрифтом заголовка:

– fontsize - размер текста

– fontweight - насыщенность текста

– color - цвет текста

– family - семейство шрифтов

В качестве параметра color можно подавать как название цвета на английском (например, black или red), так и код цвета а формате HEX (например, #808080).

```
plt.plot(x, y)

plt.title(
    "График функции",
    fontsize=16,
    fontweight="bold",
    color="#808080",
    family="serif",
)

plt.show()
```

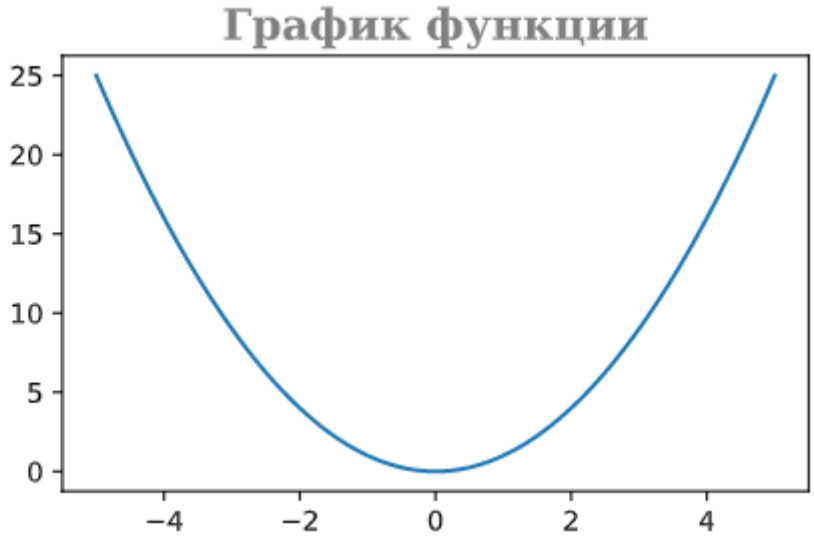

Рисунок 11 – Вывод заголовка графика

Все эти параметры можно подать также и с помощью словаря, используя аргумент fontdict:

```
title_font = {
    "fontsize": 16,
    "fontweight": "bold",
    "color": "#808080",
    "family": "serif",
}

plt.plot(x, y)

plt.title("График функции", fontdict=title_font)

plt.show()
```

Также в функции plt.title доступно и другое форматирование заголовка. Например, с помощью параметра loc можно задать расположение заголовка:

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

```
plt.title("График функции", fontdict=title_font, loc="left")

plt.show()
```

Аналогично работают функции plt.xlabel и plt.ylabel, с помощью которых можно управлять названиями осей:

```
label_font = {
    "fontsize": 9,
    "family": "serif",
}

plt.plot(x, y)

plt.title("График функции", fontdict=title_font)
plt.xlabel("Переменная x", fontdict=label_font)
plt.ylabel("Функция f(x)", fontdict=label_font)

plt.show()
```

Легенда. Добавим к нашему графику график функции $y=x^3$. Когда matplotlib строит график, он автоматически подбирает масштаб осей так, чтобы уместить все заданные точки. Иногда это приводит к искажению пропорций, особенно если значения по одной оси значительно превышают значения по другой. Чтобы задать конкретные границы для осей, можно воспользоваться функцией plt.axis, передав ей список с нужными пределами для осей x и y. Чтобы пояснить, какой график относится к какой функции, можно добавить легенду. Для этого используется функция plt.legend, которая отображает подписи к графикам на изображении (рис.12):

```
plt.plot(x, y)
plt.plot(x, y2)

plt.title("Графики функций", fontdict=title_font)
plt.xlabel("Переменная x", fontdict=label_font)
plt.ylabel("Функция f(x)", fontdict=label_font)

plt.axis([-5, 5, -8, 8])
plt.legend(labels=["x^2", "x^3"])

plt.show()
```

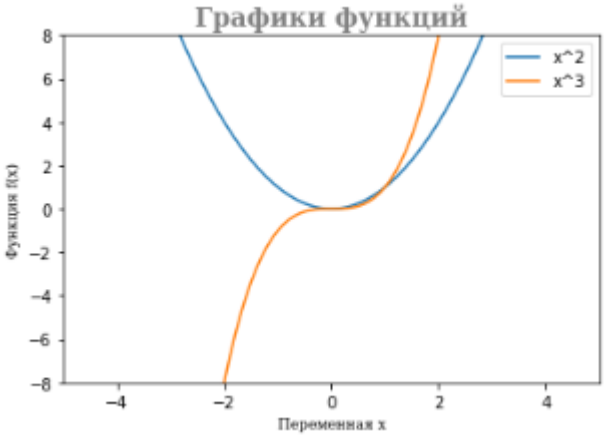

Рисунок 12 – Легенда

Оформление графиков и сетка. В функцию plt.plot можно также подавать параметры, связанные с оформлением графика:

– color - цвет
– linestyle - стиль линии

Функция plt.legend поддерживает настройку внешнего вида, в том числе форматирование текста. Однако параметры форматирования здесь могут отличаться от тех, что используются, например, для подписей осей. Если вы хотите изменить стиль шрифта, необходимо передать словарь с нужными настройками через специальный аргумент, но использовать при этом немного другие ключи, чем в других функциях оформления (рис.13).

```
legend_font = {
    "size": 7,
    "family": "serif",
}

plt.plot(x, y, label="x^2", color="green", linestyle="dotted")
plt.plot(x, y2, label="x^3", color="red", linestyle="dashed")

plt.title("Графики функций", fontdict=title_font)
plt.xlabel("Переменная x", fontdict=label_font)
plt.ylabel("Функция f(x)", fontdict=label_font)

plt.axis([-5, 5, -8, 8])

legend = plt.legend(loc="lower right", prop=legend_font)
plt.setp(legend.get_texts(), color="DarkBlue")

plt.show()
```

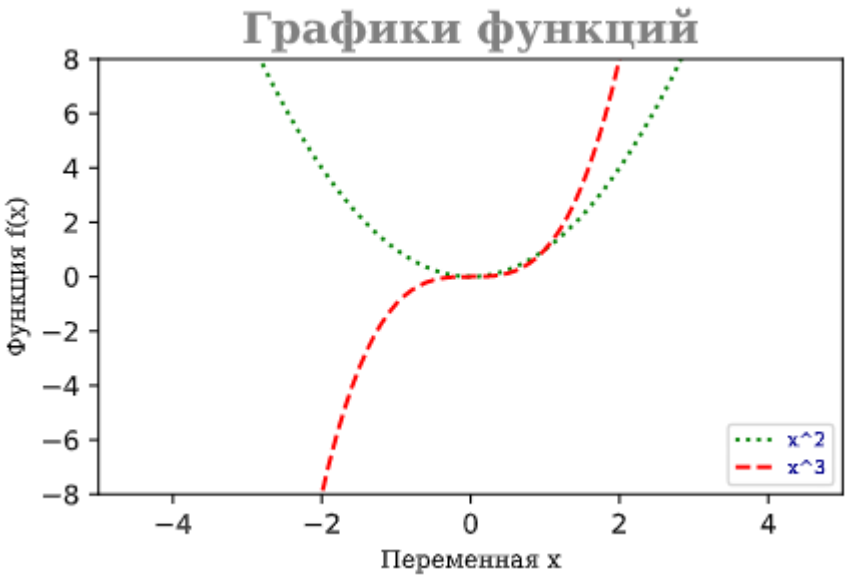

Рисунок 13 – Оформление графиков

Для улучшения восприятия данных на графике можно включить отображение сетки с помощью функции plt.grid. Сетка помогает точнее ориентироваться по осям и оценивать значения. Чтобы добавить её, достаточно включить соответствующий вызов в существующий код.

```
plt.grid()
```

5 Объекты библиотеки Matplotlib

До настоящего момента применялся процедурный подход к построению графиков, заключающийся в последовательном вызове функций для создания и настройки визуализации. Этот метод прост и нагляден, однако ограничен при реализации сложных визуализаций. Для более гибкой работы в matplotlib реализована объектно-ориентированная модель, основанная на двух основных объектах: Figure (область для размещения графиков) и Axes (отдельная система координат для каждого графика).

В процедурном подходе эти объекты создаются автоматически, и обычно используется одна фигура с одним графиком. Для размещения нескольких графиков в разных областях изображения используется функция plt.subplots, позволяющая задать количество строк и столбцов в фигуре. В результате возвращаются объекты Figure и массив Axes, каждый из которых может содержать отдельный график. Например, для отображения функций $y=x^2$ и $y=x^3$ на отдельных графиках, plt.subplots создаёт одну фигуру с двумя осями, размещёнными в одной строке.

```
In [133]: fig, ax = plt.subplots(nrows=1, ncols=2)

          ax[0].plot(x, y)
          ax[1].plot(x, y2)

Out[133]: [<matplotlib.lines.Line2D at 0x1c835ee83d0>]
```

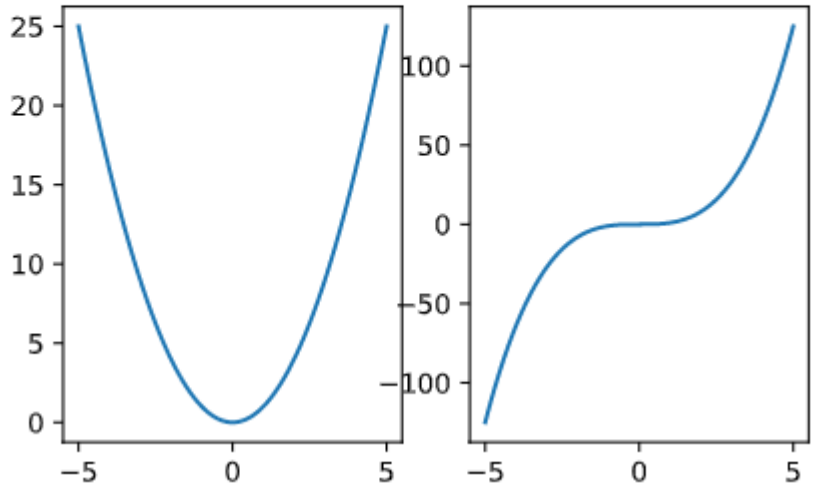

Рисунок 14 – Отельные графики

Поскольку размер фигуры заранее задан, размещение нескольких графиков на ней может привести к их неестественному масштабированию. Однако, обладая доступом к объекту Figure, можно напрямую изменять параметры фигуры: задавать её размер и регулировать межграфиковые отступы. Параметр **wspace** метода .**subplots_adjust()** определяет относительное расстояние между графиками по ширине, а **hspace** — по высоте; оба значения задаются в долях от средней ширины или высоты осей. При компоновке с одной строкой или одним столбцом массив ax остаётся одномерным. Для удобства индексации можно использовать метод .**flatten()** из numpy, преобразующий массив осей в одномерный.

```
ax = ax.flatten()  # "Разворачиваем" массив в одномерный
```

Это особенно полезно, когда структура осей может меняться, но требуется единообразный доступ к ним по индексу.

```
ar = np.array([[1, 2],
               [3, 4]])

ar.flatten()
```

Рассмотрим расширение примера с созданием фигуры, содержащей четыре графика, организованных в сетку размером 2×2. В этом случае возвращаемый массив осей ax будет иметь двумерную структуру, где оси располагаются по строкам и столбцам сетки. Для удобства обработки такого

массива можно применить метод .**flatten()**, который преобразует двумерный массив в одномерный, упорядочивая элементы сначала по строкам слева направо, а затем переходя к следующей строке (см. рис. 15). Такой подход облегчает последовательный доступ к каждому объекту Axes при работе с множественными подграфиками.

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

# Создаём фигуру с 2 строками и 2 столбцами
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(10, 8))
ax = ax.flatten()  # Преобразуем в одномерный массив

# Генерируем данные
x = np.linspace(0, 10, 100)
functions = [np.sin(x), np.cos(x), x**0.5, np.exp(x*0.2)]
titles = ['y = sin(x)', 'y = cos(x)', 'y = √x', 'y = e^(0.2x)']

# Строим графики в цикле
for i in range(4):
    ax[i].plot(x, functions[i])
    ax[i].set_title(titles[i])
    ax[i].grid(True)

plt.tight_layout()  # Автоматически регулируем отступы
plt.show()
```

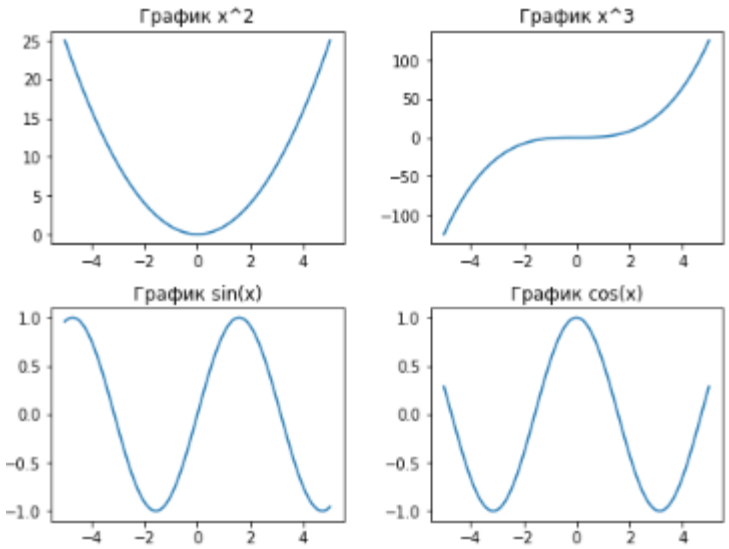

Рисунок 15 – Графики функций

6 Построение 3D графиков

До этого момента все графики, которые мы строили были двумерные, *Matplotlib* позволяет строить *3D* графики. Импортируем необходимые модули для работы с *3D* (рис. 16):

```
In [4]: import numpy as np
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D
```

Рисунок 16 – Подключение библиотек

В библиотеке доступны инструменты для построения различных типов графиков. Рассмотрим некоторые из них более подробно.

Линейный график. Для построения линейного графика используется функция *plot()*.

```
Axes3D.plot(self, xs, ys, *args, zdir='z', **kwargs)
```

– *xs*: *x* координаты.

– *ys*: *y* координаты.

– *zs*: *z* координаты. Если передан скаляр, то он будет присвоен всем точкам графика.

– *zdir: {'x', 'y', 'z'}* Определяет ось, которая будет принята за *z* направление, значение по умолчанию: *'z'*.

– ***kwargs* Дополнительные аргументы, аналогичные тем, что используются в функции *plot()* для построения двумерных графиков. Пример (рис.17):

```
In [5]: x = np.linspace(-np.pi, np.pi, 50)
        y = x
        z = np.cos(x)
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.plot(x, y, z, label='parametric curve')
Out[5]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x22fc6cffe20>]
```

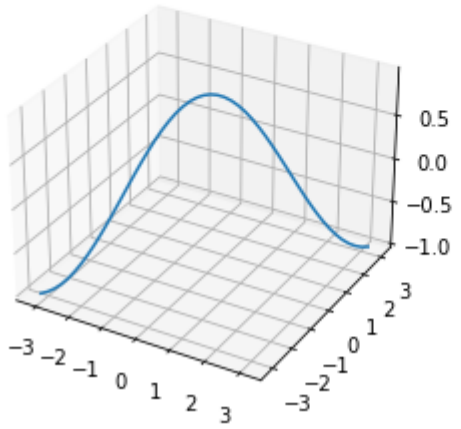

Рисунок 17 – Линейный график

Точечный график. Для построения точечного графика используется функция *scatter()*.

```
Axes3D.scatter(self, xs, ys, zs=0, zdir='z', s=20, c=None,
depthshade=True, *args, **kwargs)
```

– *xs, ys*: массив, Координаты точек по осям *x* и *y*.

– *zs*: *float* или массив, *optional*. Координаты точек по оси *z*. Если передан скаляр, то он будет присвоен всем точкам графика. Значение по умолчанию: 0.

– *zdir: {'x', 'y', 'z', '-x', '-y', '-z'}, optional*. Определяет ось, которая будет принята за *z* направление, значение по умолчанию: *'z'*

– *s*: скаляр или массив, *optional*. Размер маркера. Значение по умолчанию: 20.

– *c*: *color*, массив, массив значений цвета, *optional*

- Цвет маркера. Возможные значения:
  - Строковое значение цвета для всех маркеров.
  - Массив строковых значений цвета.
  - Массив чисел, которые могут быть отображены в цвета через функции *cmap* и *norm*.
  - *2D* массив, элементами которого являются *RGB* или *RGBA*.

– *depthshade: bool, optional*. Затенение маркеров для придания эффекта глубины.

– ***kwargs*. Дополнительные аргументы, аналогичные тем, что используются в функции *scatter()* для построения двумерных графиков. Пример (рис.18):

```
In [8]: np.random.seed(123)
        x = np.random.randint(-5, 5, 40)
        y = np.random.randint(0, 10, 40)
        z = np.random.randint(-5, 5, 40)
        s = np.random.randint(10, 100, 40)
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(x, y, z, s=s)
Out[8]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x22fc6e104f0>
```

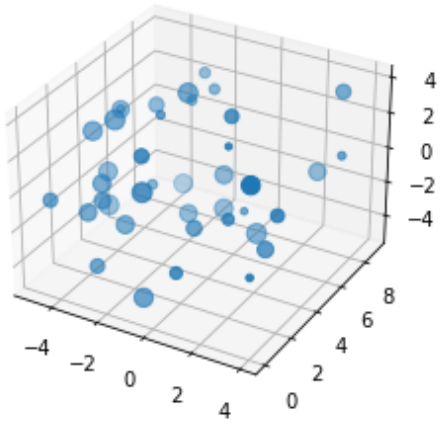

Рисунок 18 – Тотетный 3D график

Каркасная поверхность. Для построения каркасной поверхности используется функция *plot_wireframe()*.

```
plot_wireframe(self, X, Y, Z, *args, **kwargs)
```

– *X, Y, Z: 2D* массивы. Данные для построения поверхности.

– *rcount, ccount: int*. Максимальное количество элементов каркаса, которое будет использовано в каждом из направлений. Значение по умолчанию: 50.

– *rstride, cstride: int*. Параметры определяют величину шага, с которым будут браться элементы строки / столбца из переданных массивов. Параметры *rstride*, *cstride* и *rcount*, *ccount* являются взаимоисключающими.

– ***kwargs*. Дополнительные аргументы, определяемые *Line3DCollection*.

Пример (рис.19):

```
In [12]: u, v = np.mgrid[0:2*np.pi:20j, 0:np.pi:10j]
         x = np.cos(u)*np.sin(v)
         y = np.sin(u)*np.sin(v)
         z = np.cos(v)
         fig = plt.figure()
         ax = fig.add_subplot(111, projection='3d')
         ax.plot_wireframe(x, y, z)
Out[12]: <mpl_toolkits.mplot3d.art3d.Line3DCollection at 0x22fc7908280>
```

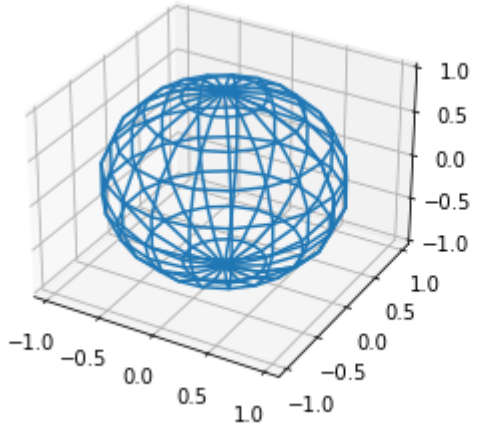

Рисунок 19 – Каркасная поверхность

Поверхность. Для построения поверхности используйте функцию *plot_surface()*.

```
plot_surface(self, X, Y, Z, *args, norm=None, vmin=None,
vmax=None, lightsource=None, **kwargs)
```

– *X, Y, Z : 2D* массивы. Данные для построения поверхности.

– *rcount, ccount : int*

– см. *rcount, ccount* в “Каркасная поверхность“.

– *rstride, cstride : int* .см.*rstride, cstride* в “Каркасная поверхность“.

– *color: color*. Цвет для элементов поверхности.

– *cmap*: *Colormap*. *Colormap* для элементов поверхности.

– *facecolors*: массив элементов *color*. Индивидуальный цвет для каждого элемента поверхности.

– *norm*: *Normalize*. Нормализация для *colormap*.

– *vmin, vmax: float*. Границы нормализации.

– *shade: bool*. Использование тени для *facecolors*. Значение по умолчанию: *True*.

– *lightsource*: *LightSource*. Объект класса *LightSource* – определяет источник света, используется, только если *shade = True*.

– ***kwargs*. Дополнительные аргументы, определяемые *Poly3DCollection*.

Пример (рис.20):

```
In [11]: u, v = np.mgrid[0:2*np.pi:20j, 0:np.pi:10j]
         x = np.cos(u)*np.sin(v)
         y = np.sin(u)*np.sin(v)
         z = np.cos(v)
         fig = plt.figure()
         ax = fig.add_subplot(111, projection='3d')
         ax.plot_surface(x, y, z, cmap='inferno')
```

```
Out[11]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x22fc7861430>
```

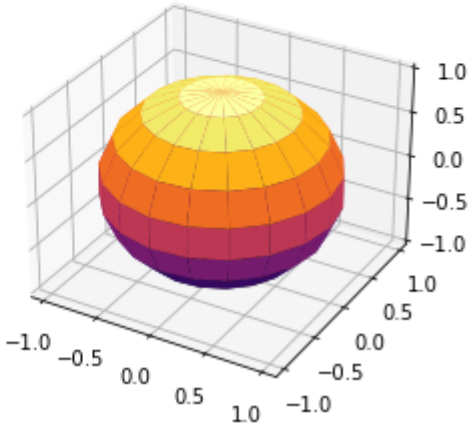

Рисунок 20 – Поверхность