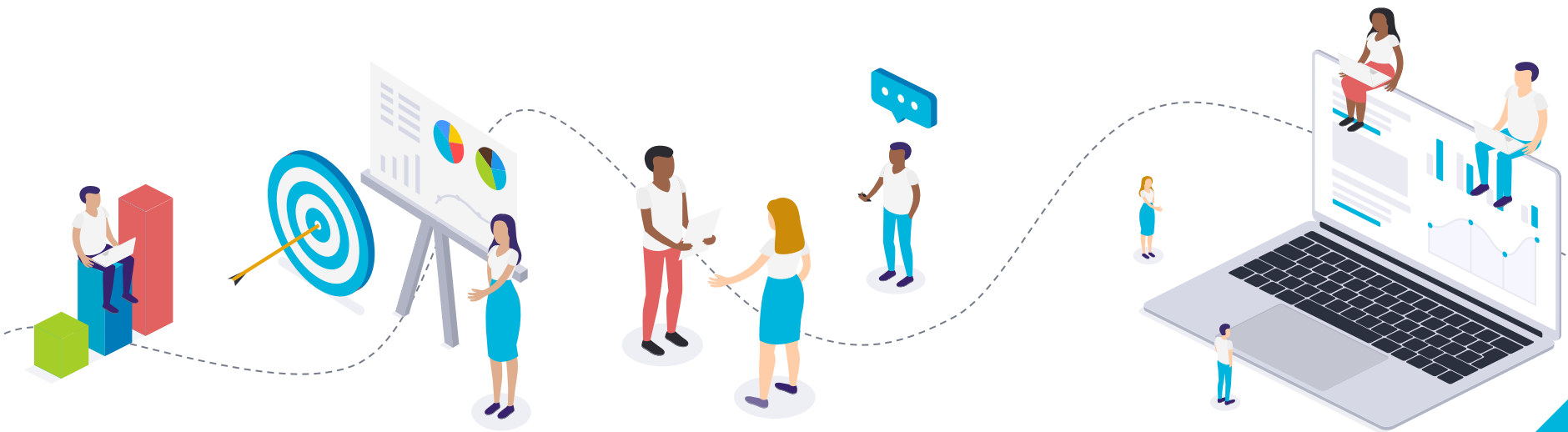


МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»



Программа профессиональной переподготовки «Технологии искусственного интеллекта, визуализации и анализа данных на Python»



Создание отличных графиков – это наука и искусство

Тема 1.4. Визуализация данных

Часть 1. Визуализация данных с помощью библиотеки Matplotlib



Визуализация данных с помощью библиотеки Matplotlib

Matplotlib – библиотека языка Python, содержащая функции для визуализации данных.

Matplotlib имеет иерархическую структуру и использует принципы объектно-ориентированного программирования.

Базовые функции для создания графиков расположены в модуле **matplotlib.pyplot**.

Импорт библиотек, модулей и встроенных команд:

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

Часть 1.1. Основные компоненты matplotlib.

Иерархическая структура рисунка

Пользовательская работа подразумевает операции с разными уровнями рисунка:
Figure (Окно рисунка) -> **Axes** (Область рисования) -> **Axis** (Координатная ось)

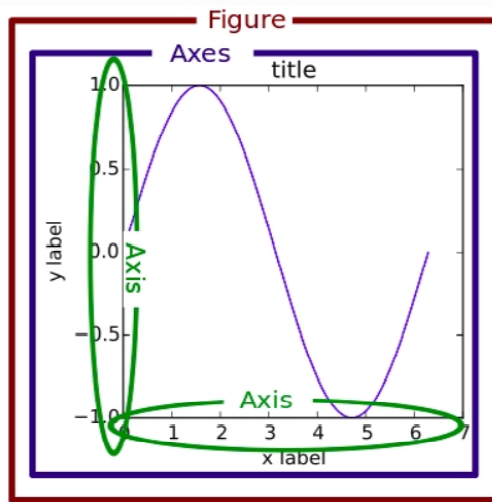


Рисунок 1. – Изображение иерархии

Основные компоненты matplotlib. Иерархическая структура рисунка

Верхний уровень иерархии – окно рисунка (**Figure**). На нем располагаются:

- одна или несколько областей (панелей) рисования – **Axes**;
- элементы рисунка – **Artists** (заголовки, легенда и т.д.);
- основа-холст – **Canvas**.

Панели (области рисования) являются объектами класса **Axes** – это та область, на которую наносятся координатные оси, линии графиков и столбцы диаграмм, легенды к ним и т.д.

Координатные оси являются объектами класса **Axis** и определяют область изменения данных. На них наносятся:

- деления ticks;
- подписи к делениям ticklabels.

Расположение делений определяется объектом Locator, а подписи делений обрабатывает объект Formatter.

Практически все, что отображается на рисунке, является элементом рисунка (**Artist**), даже объекты Figure, Axes и Axis. Элементы рисунка Artists включают в себя следующие объекты:

- текст (Text);
- плоская линия (Line2D);
- фигура (Patch) и другие.

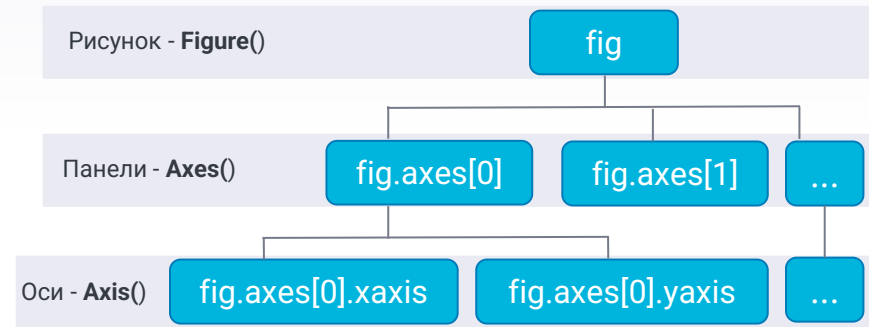


Рисунок 2. – Схема иерархии объектов, служащих для создания рисунка в библиотеке Matplotlib

Часть 1.2. Pyplot – интерфейс для построения графиков функций

Некоторые функции отрисовки

```
plt.scatter(x, y, params)      # – нарисовать точки с координатами из x по горизонтальной
                               # оси и из y по вертикальной оси;
plt.plot(x, y, params)         # – нарисовать график по точкам с координатами из x по
                               # горизонтальной оси и из y по вертикальной оси.
                               # Точки будут соединяться в том порядке, в котором они
                               # указаны в этих массивах;
plt.fill_between(x, y1, y2, params) # – закрасить пространство между y1 и y2 по координатам из x;
plt.pcolormesh(x1, x1, y, params)  # – закрасить пространство в соответствии с интенсивностью y;
plt.contour(x1, x1, y, lines)      # – нарисовать линии уровня. Затем нужно применить plt.clabel.
```

Вспомогательные функции

```
plt.figure(figsize=(x, y)) # – создать график размера (x, y);
plt.show()                 # – показать график;
plt.subplot(...)           # – добавить подграфик;
plt.xlim(x_min, x_max)     # – установить пределы графика по горизонтальной оси;
plt.ylim(y_min, y_max)     # – установить пределы графика по вертикальной оси;
plt.title(name)            # – установить имя графика;
plt.xlabel(name)           # – установить название горизонтальной оси;
plt.ylabel(name)           # – установить название вертикальной оси;
plt.legend(loc=...)        # – сделать легенду в позиции loc;
plt.grid()                 # – добавить сетку на график;
plt.savefig(filename)      # – сохранить график в файл.
```

Pyplot – интерфейс для построения графиков функций

Для того, чтобы посмотреть все параметры функций в matplotlib, можно воспользоваться справкой `plt.plot?`
Появится полная информация по всем параметрам:

Signature: `plt.plot(*args, scalex=True, scaley=True, data=None, **kwargs)`

Docstring:

Plot y versus x as lines and/or markers.

Call signatures::

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

The coordinates of the points or line nodes are given by `*x*`, `*y*`.

The optional parameter `*fmt*` is a convenient way for defining basic formatting like color, marker and linestyle. It's a shortcut string notation described in the `*Notes*` section below.

```
>>> plot(x, y)           # plot x and y using default line style and color
>>> plot(x, y, 'bo')      # plot x and y using blue circle markers
>>> plot(y)              # plot y using x as index array 0..N-1
```

В matplotlib работает правило «текущей области» («current axes»), которое означает, что все графические элементы наносятся на **текущую область** рисования.

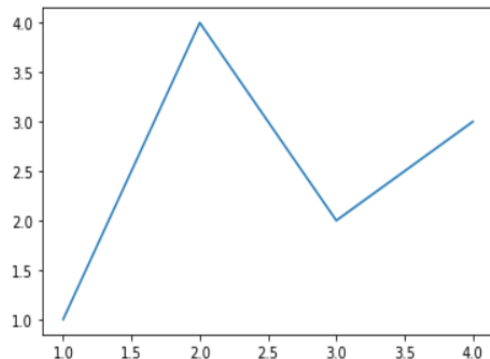
Часть 1.3. Простые графики

Построение графиков начинают с создания области рисунка. Самый простой способ создать рисунок – использовать оболочку **pyplot.subplots()**, затем – **Axes.plot()** для рисования графика:

Пример 1

```
fig, ax = plt.subplots()
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])
```

```
[<matplotlib.lines.Line2D at 0x2906fee6288>]
```



Списки *x* и *y* в функциях – координаты точек. Точки соединяются прямыми.

Создать рисунок (figure) также позволяет метод **plt.figure()**.

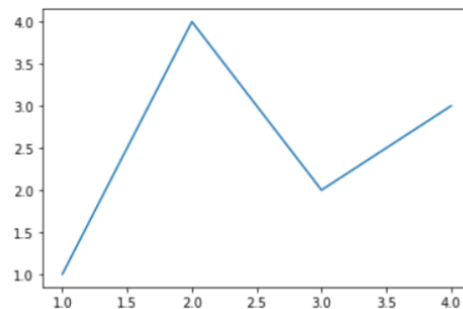
Чтобы текущее состояние рисунка отразилось на экране, можно воспользоваться командой **plt.show()**.

Пример 2

Предыдущий код (пример 1) можно сделать короче, если воспользоваться методом **plt.plot()**:

```
plt.plot([1, 2, 3, 4], [1, 4, 2, 3])
```

```
[<matplotlib.lines.Line2D at 0x2906ff70f88>]
```

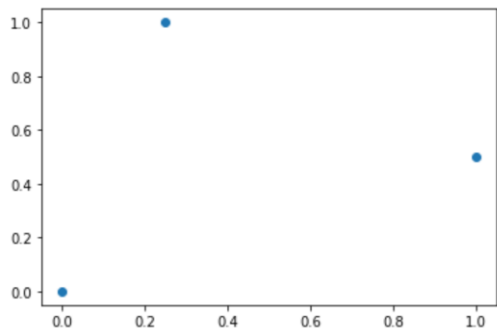


Простые графики

Пример 3

Функция **scatter** из модуля **pyplot** просто рисует точки, не соединяя их линиями.

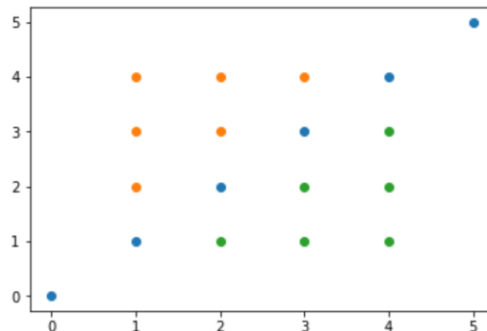
```
# Создать рисунок
plt.figure()
# Нарисовать отдельные точки с координатами (x, y) с помощью функции scatter
plt.scatter([0, 0.25, 1], [0, 1, 0.5])
# Отобразить рисунок на экране
plt.show()
```



Пример 4

Если имеется несколько множеств точек, то все их можно построить на одном графике:

```
plt.scatter([0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5])
plt.scatter([1, 2, 3, 1, 2, 1], [2, 3, 4, 3, 4, 4])
plt.scatter([2, 3, 4, 3, 4, 4], [1, 2, 3, 1, 2, 1])
plt.show()
```

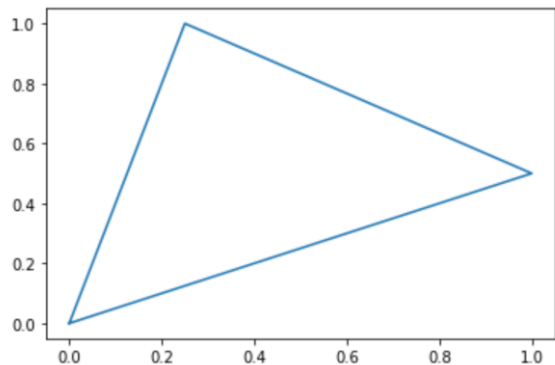


Простые графики

Пример 5

Построим замкнутый многоугольник.

```
plt.figure()  
plt.plot([0, 0.25, 1, 0], [0, 1, 0.5, 0])  
plt.show()
```



Часть 1.4. Использование массивов библиотеки NumPy

Библиотека **matplotlib** основана на массивах библиотеки **NumPy (np)** и спроектирована в расчете на работу с библиотекой **SciPy (sp)**.

Функция **linspace()** возвращает одномерный массив **NumPy** из указанного количества элементов, значения которых равномерно распределены внутри заданного интервала, и создает массив координат. Синтаксис вызова функции:

```
numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
```

Параметры:

- **start** – число, которое является началом последовательности.
- **stop** – число, которое является концом последовательности, если `endpoint=True`.
Если `endpoint=False`, то данное число не включается в интервал, при этом значение шага между элементами последовательности изменяется.
- **num** – целое положительное число (необязательный параметр). Определяет количество элементов последовательности. По умолчанию `num = 50`.
- **endpoint** – True или False (необязательный).
Если `endpoint = True`, то значение `stop` включается в интервал и является последним. В противном случае – `stop` не входит в интервал. По умолчанию `endpoint = True`.
- **retstep** – True или False (необязательный). Если `retstep = True`, то будет возвращено значение шага между элементами.
- **dtype** – тип данных NumPy (необязательный). Определяет тип данных выходного массива. Если этот параметр не указан, то он будет определен автоматически на основе других параметров.

Использование массивов библиотеки NumPy

Пример 6. Поработаем с функцией `linspace()`

6.1. По умолчанию количество элементов последовательности выводится 50.

```
# Создаем массив координат – по умолчанию 50 точек,  
# равномерно распределенных в диапазоне от 0 до 1  
np.linspace(0, 1)
```

```
array([0.          , 0.02040816, 0.04081633, 0.06122449, 0.08163265,  
       0.10204082, 0.12244898, 0.14285714, 0.16326531, 0.18367347,  
       0.20408163, 0.2244898 , 0.24489796, 0.26530612, 0.28571429,  
       0.30612245, 0.32653061, 0.34693878, 0.36734694, 0.3877551 ,  
       0.40816327, 0.42857143, 0.44897959, 0.46938776, 0.48979592,  
       0.51020408, 0.53061224, 0.55102041, 0.57142857, 0.59183673,  
       0.6122449 , 0.63265306, 0.65306122, 0.67346939, 0.69387755,  
       0.71428571, 0.73469388, 0.75510204, 0.7755102 , 0.79591837,  
       0.81632653, 0.83673469, 0.85714286, 0.87755102, 0.89795918,  
       0.91836735, 0.93877551, 0.95918367, 0.97959184, 1.        ])
```

6.2.

```
np.linspace(0, 1, num = 5)
```

```
array([0. , 0.25, 0.5 , 0.75, 1.  ])
```

6.3.

```
np.linspace(0, 1, num = 5, endpoint = False)
```

```
array([0. , 0.2, 0.4, 0.6, 0.8])
```

Если `endpoint = False`, то значение `stop` не входит в интервал.
По умолчанию `endpoint = True`.

6.4.

```
np.linspace(0, 1, num = 5, endpoint = False, retstep = True)
```

```
(array([0. , 0.2, 0.4, 0.6, 0.8]), 0.2)
```

Если `retstep = True`, то возвращено значение шага между элементами, равное 0.2.

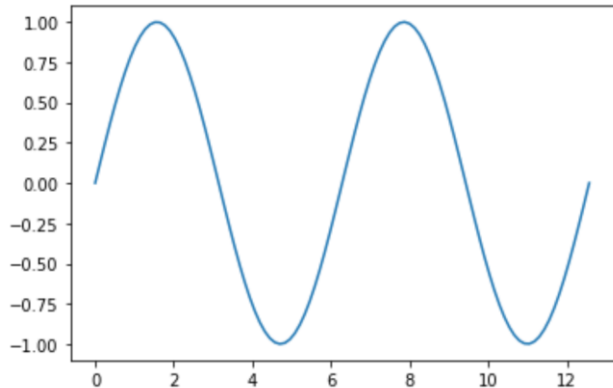
Использование массивов библиотеки NumPy

Пример 7

Построим ломаную с использованием функции `linspace()`:

```
x = np.linspace(0, 4*np.pi, 100)

plt.figure()
plt.plot(x, np.sin(x))
plt.show()
```

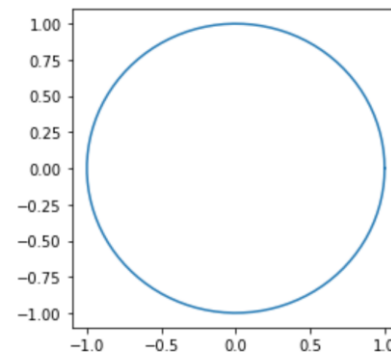


Пример 8

Построим параметрическую линию $x=x(t)$, $y=y(t)$:

```
t = np.linspace(0, 2*np.pi, 100)

plt.figure(figsize=(4,4))
plt.plot(np.cos(t), np.sin(t))
plt.show()
```



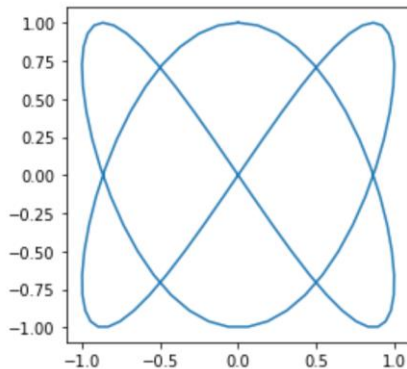
Использование массивов библиотеки NumPy

Пример 9

Построим **фигуры Лиссажу** – траектории, прочерчиваемые точкой, совершающей одновременно два гармонических колебания в двух взаимно перпендикулярных направлениях:

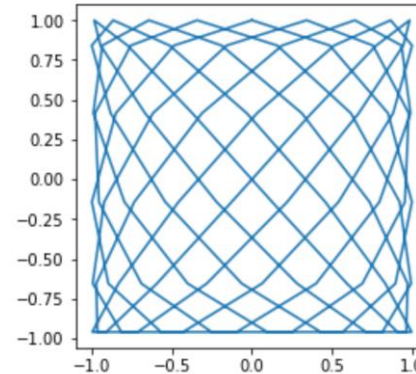
9.1.

```
plt.figure(figsize=(4, 4))  
plt.plot(np.sin(2*t), np.cos(3*t))  
plt.show()
```



9.2. Изменим параметры: a=8, b=9

```
plt.figure(figsize=(4, 4))  
plt.plot(np.sin(8*t), np.cos(9*t))  
plt.show()
```



Часть 1.5. Расположение нескольких кривых на одном графике

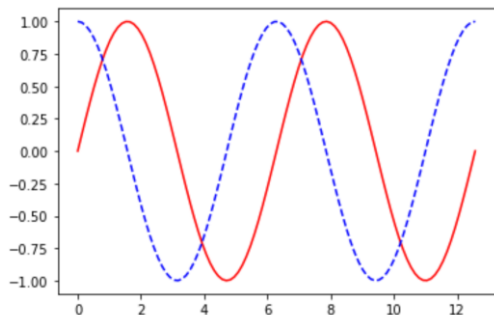
Каждая кривая задаётся парой массивов координат – x и y .

Для регулировки цветов и типов линий после пары x и y координат вставляется форматная строка: первая буква определяет цвет ('r' – **красный**, 'b' – **синий**, 'g' – **зелёный** и т.д.), дальше задаётся тип линии ('-' – сплошная, '--' – пунктирная, '-.' – штрих-пунктирная и т.д.).

Пример 10

```
x = np.linspace(0, 4 * np.pi, 100)

plt.figure()
plt.plot(x, np.sin(x), 'r-')
plt.plot(x, np.cos(x), 'b--')
plt.show()
```

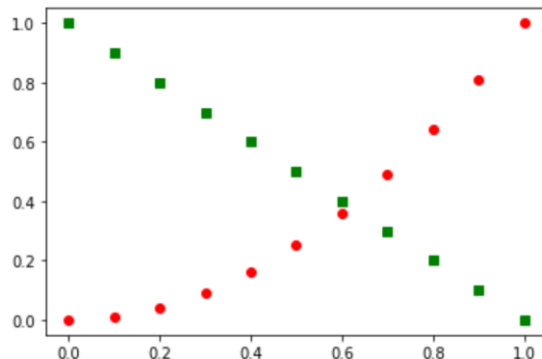


Пример 11

Если в качестве «типа линии» указано 'o', то это означает рисовать точки кружочками и не соединять их линиями; аналогично, 's' означает квадратики.

```
x = np.linspace(0, 1, 11)

plt.figure()
plt.plot(x, x ** 2, 'ro')
plt.plot(x, 1 - x, 'gs')
plt.show()
```



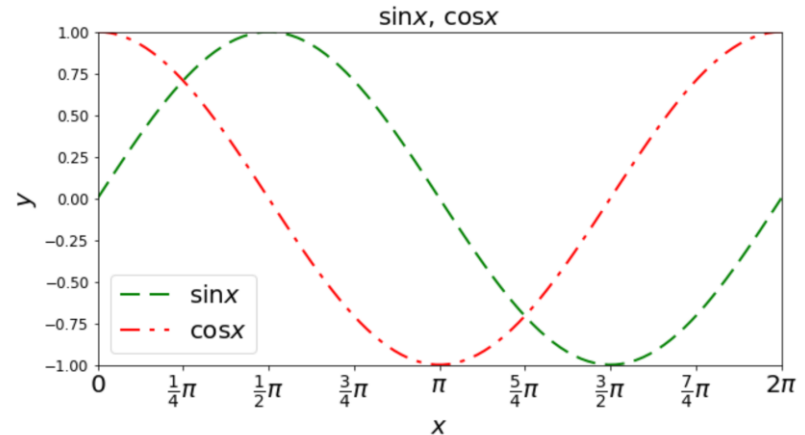
Часть 1.6. Настройка графиков

- Как задать последовательность засечек на осях x и y и сделать подписи к ним.
- Как задать подписи осей x и y и заголовок графика.
- Как задать в текстовых элементах размер шрифта.
- Как задать толщину линий и штрихи.
- Как задать подписи к кривым (legend), размещение которых можно регулировать.

Пример 12

```
x = np.linspace(0, 2*np.pi, 100)

plt.figure(figsize=(10, 5))
plt.plot(x, np.sin(x), linewidth=2, color='g', dashes=[8, 4], label=r'$\sin x$')
plt.plot(x, np.cos(x), linewidth=2, color='r', dashes=[8, 4, 2, 4], label=r'$\cos x$')
plt.axis([0, 2*np.pi, -1, 1])
plt.xticks(np.linspace(0, 2*np.pi, 9), # Где сделать отметки
            ('0', r'$\frac{1}{4}\pi$', r'$\frac{1}{2}\pi$', # Как подписать
             r'$\frac{3}{4}\pi$', r'$\pi$', r'$\frac{5}{4}\pi$',
             r'$\frac{3}{2}\pi$', r'$\frac{7}{4}\pi$', r'$2\pi$'),
            fontsize=20)
plt.yticks(fontsize=12)
plt.xlabel(r'$x$', fontsize=20)
plt.ylabel(r'$y$', fontsize=20)
plt.title(r'$\sin x$, $\cos x$', fontsize=20)
plt.legend(fontsize=20, loc=0)
plt.show()
```



Настройка графиков

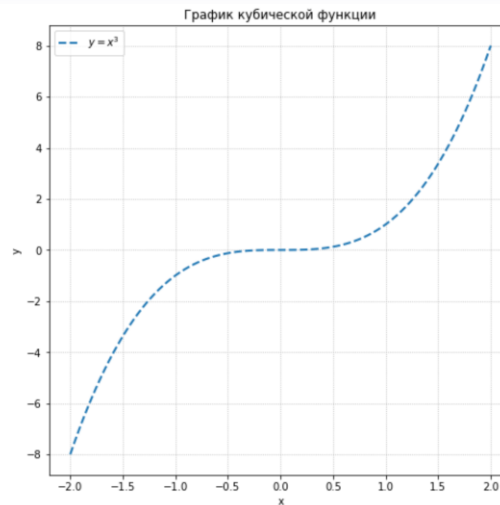
Пример 13

Построим пунктирный график функции $y=x^3$. Подпишем оси, заголовок и легенду. Нанесем сетку.

```
x = np.linspace(-2, 2, 100)

plt.figure(figsize=(8, 8))
plt.plot(x, x**3, linestyle='--', lw=2, label='$y=x^3$')
plt.xlabel('x'), plt.ylabel('y')
plt.legend()
plt.title('График кубической функции')
plt.grid(ls=':')
plt.show()
```

Примечание. Если `linestyle=''`, то точки не соединяются линиями. Сами точки могут рисоваться маркерами разных типов.

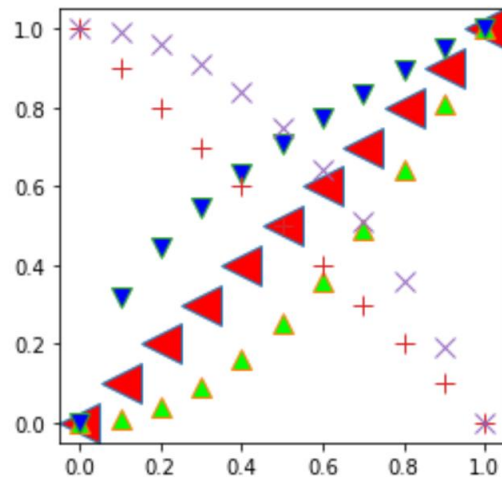


Настройка графиков

Пример 14

```
x = np.linspace(0, 1, 11)

plt.figure(figsize=(4, 4))
plt.plot(x, x, linestyle='', marker='<', markersize=20, markerfacecolor='#FF0000')
plt.plot(x, x ** 2, linestyle='', marker='^', markersize=10, markerfacecolor='#00FF00')
plt.plot(x, x ** (1/2), linestyle='', marker='v', markersize=10, markerfacecolor='#0000FF')
plt.plot(x, 1 - x, linestyle='', marker='+', markersize=10, markerfacecolor='#0F0F00')
plt.plot(x, 1 - x ** 2, linestyle='', marker='x', markersize=10, markerfacecolor='#0F000F')
plt.axis([-0.05, 1.05, -0.05, 1.05])
plt.show()
```

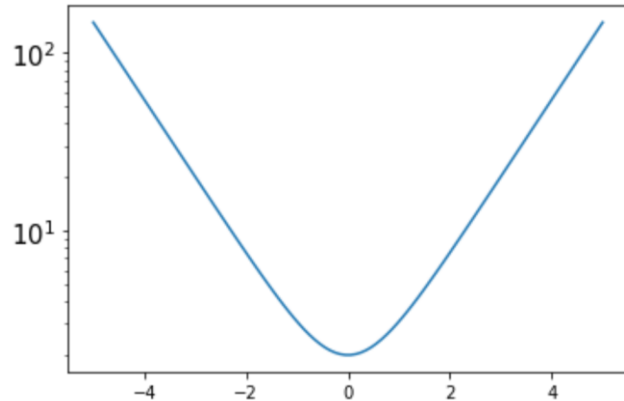


Настройка графиков

Пример 15

```
x = np.linspace(-5, 5, 100)

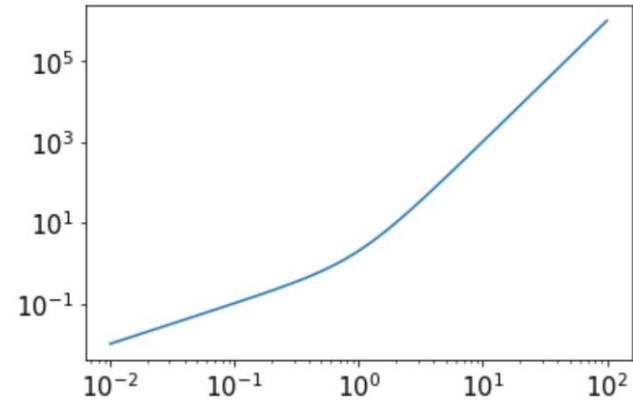
plt.figure()
plt.plot(x, np.exp(x) + np.exp(-x))
plt.yscale('log')
plt.yticks(fontsize=15)
plt.show()
```





Пример 16

```
x = np.logspace(-2, 2, 100)

plt.figure()
plt.plot(x, x + x**3)
plt.xscale('log'), plt.xticks(fontsize=15)
plt.yscale('log'), plt.yticks(fontsize=15)
plt.show()
```



- 
- ▶ **Конец Части 1 Лекции 1.4: «Визуализация данных с помощью библиотеки Matplotlib».**
 - ▶ **Смотрите продолжение: Часть 2 Лекции 1.4.**
- 

Часть 1.7. Сложные графики.

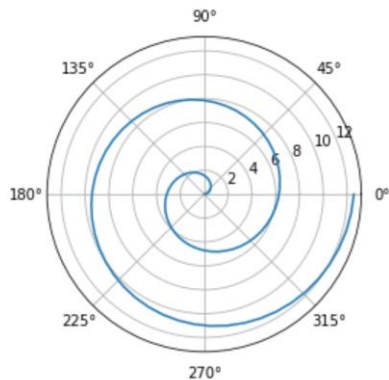
1.7.1. Полярные координаты

Пример 17

17.1. Рисунок спирали: первый массив – φ , второй – r

```
t = np.linspace(0, 4*np.pi, 100)

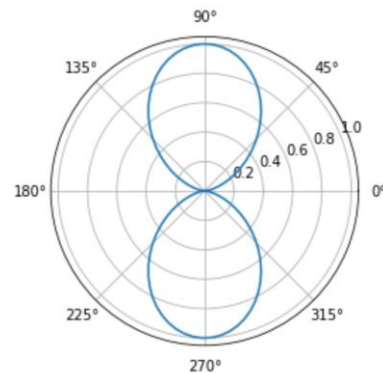
plt.figure()
plt.polar(t, t)
plt.show()
```



17.2. Угловое распределение пионов (субатомная частица) в e^+e^- аннигиляции:

```
phi = np.linspace(0, 2*np.pi, 100)

plt.figure()
plt.polar(phi, np.sin(phi)**2)
plt.show()
```



Часть 1.7. Сложные графики.

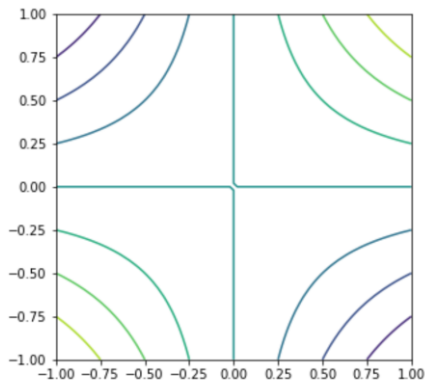
1.7.2. Контурные графики

Пример 18

18.1. Изучим поверхность $z=xy$.
Построим ее линии уровня:

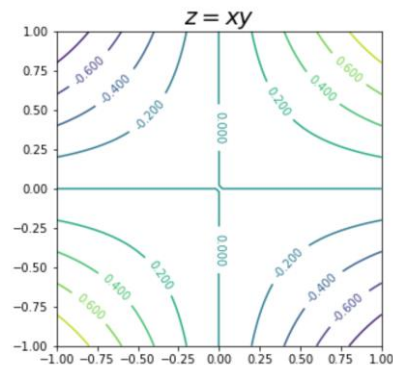
```
x = np.linspace(-1, 1, 50)
y = x
z = np.outer(x, y)

plt.figure(figsize=(5,5))
plt.contour(x, y, z)
plt.show()
```



18.2. Сделаем больше линий уровней и подпишем их.

```
plt.figure(figsize=(5,5))
curves = plt.contour(x, y, z, np.linspace(-1, 1, 11))
plt.clabel(curves)
plt.title(r'$z=xy$', fontsize=20)
plt.show()
```

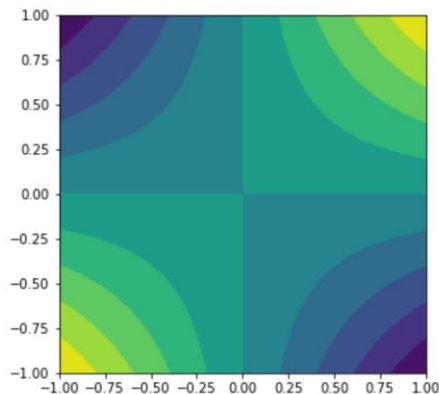


Часть 1.7. Сложные графики.

1.7.2. Контурные графики

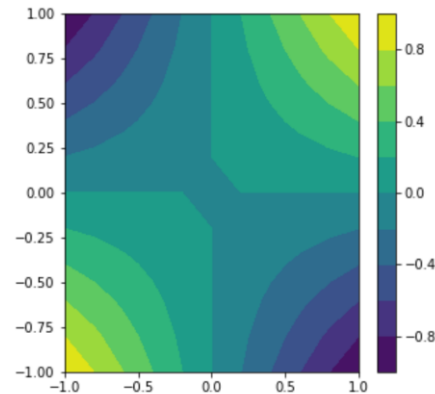
18.3. Заменяем функцию `contour` на `contourf`:

```
plt.figure(figsize=(5,5))
plt.contourf(x, y, z, np.linspace(-1, 1, 11))
# plt.colorbar()
plt.show()
```



18.4. Зададим собственную шкалу с помощью функции `colorbar()`.
Функция `colorbar` показывает соответствие цветов и значений `z`.

```
plt.figure(figsize=(5, 5))
plt.contourf(x, y, z, np.linspace(-1, 1, 11))
plt.colorbar()
plt.show()
```



Часть 1.7. Сложные графики.

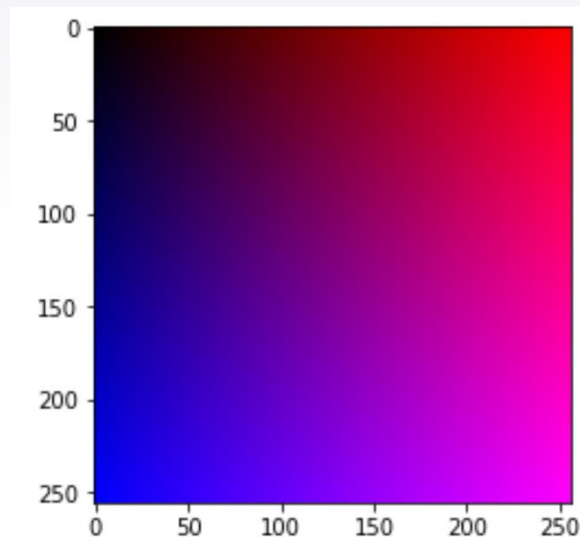
1.7.3. Images (пиксельные картинки)

Пример 19

19.1.

```
n = 256
u = np.linspace(0, 1, n)
x, y = np.meshgrid(u, u)
z = np.zeros((n, n, 3))
z[:, :, 0] = x
z[:, :, 2] = y

plt.figure()
plt.imshow(z)
plt.show()
```



`np.zeros((n, n, 3))` – функции для создания специальной нулевой матрицы.

В качестве аргумента в нее передается кортеж, первое число – это количество строк, второе – столбцов.

Картинка задаётся методом `plt.imshow()` и массивом `z: z[i,j]` – это цвет пикселя `i,j`.

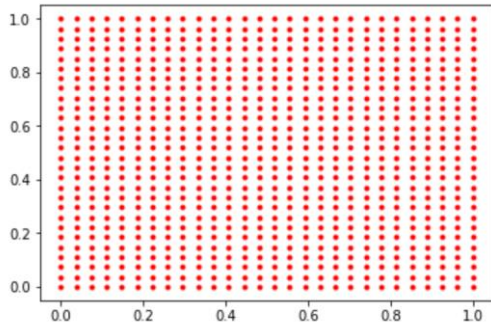
Часть 1.7. Сложные графики.

1.7.3. Images (пиксельные картинки)

Пример 19

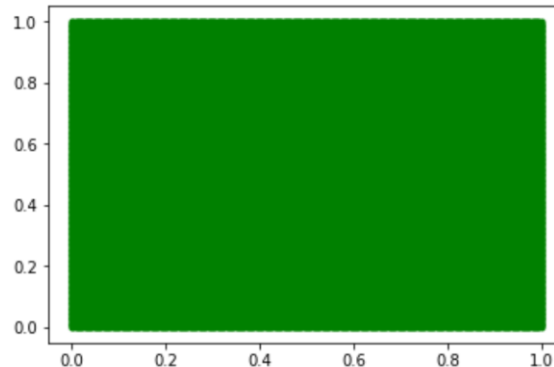
19.2. Построим массивы значений x и y при $n = 28$, чтобы убедиться, что они представляют собой сетку:

```
n = 28
u = np.linspace(0, 1, n)
x, y = np.meshgrid(u, u)
plt.plot(x, y, marker='.', color='r', linestyle='none')
plt.show()
```



19.3. Увеличим n : $n = 256$ и цвет сделаем зеленым ('g')

```
n = 256
u = np.linspace(0, 1, n)
x, y = np.meshgrid(u, u)
plt.plot(x, y, marker='.', color='g', linestyle='none')
plt.show()
```



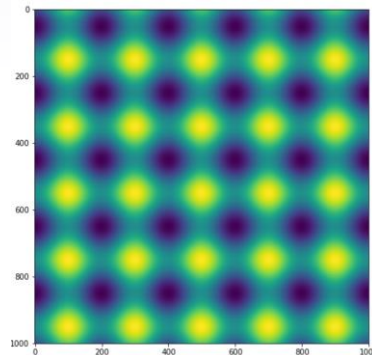
Часть 1.7. Сложные графики.

1.7.3. Images (пиксельные картинки)

Пример 19

19.4. Нередко, бывает очень полезным отобразить в виде изображения график некоторой трехмерной функции $z = f(x, y)$, на котором каждому значению z соответствует определенный цвет:

```
x, y = np.mgrid[-5*np.pi:5*np.pi:1000j,  
                -5*np.pi:5*np.pi:1000j]  
  
z = np.sin(x) + np.cos(y)  
  
fig, ax = plt.subplots()  
ax.imshow(z)  
  
fig.set_figwidth(8)    # ширина и  
fig.set_figheight(8)   # высота "Figure"  
  
plt.show()
```



Пример 20. Загрузим картинку из файла:

```
picture = plt.imread('Прописывается путь, например, G:/ДПО/МАТPLOTLIB/Логотип Питона.jpeg')  
print(type(picture), picture.shape)  
  
plt.imshow(picture)  
plt.axis('off')  
plt.show()
```



Часть 1.8. Трёхмерная графика

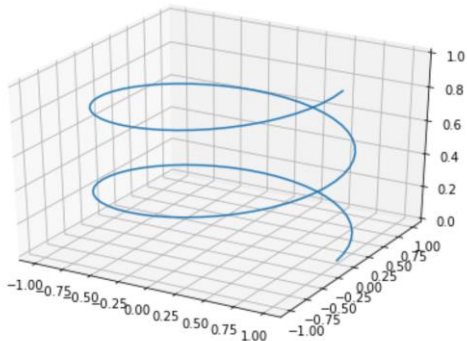
Трёхмерная графика задаётся параметрически: $x=x(t)$, $y=y(t)$, $z=z(t)$ с использованием объекта класса `Axes3D` из пакета `mpl_toolkits.mplot3d`.

Создаём в нём объект `ax`, потом используем его методы.

Пример 21

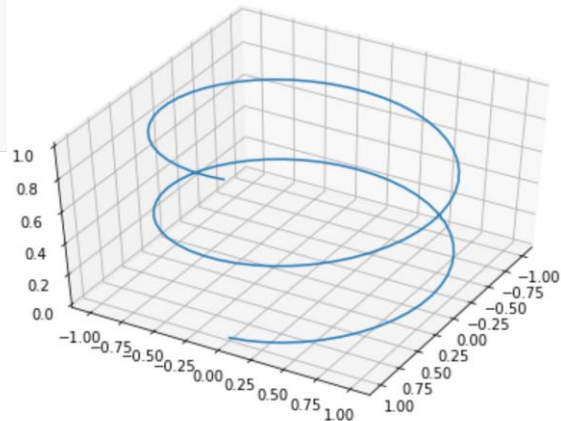
21.1.

```
t = np.linspace(0, 4*np.pi, 100)
x = np.cos(t)
y = np.sin(t)
z = t/(4*np.pi)
fig = plt.figure()
from mpl_toolkits.mplot3d import Axes3D
ax = Axes3D(fig)
ax.plot(x, y, z)
plt.show()
```



21.2. Трёхмерную картинку нельзя вертеть мышкой, но можно задать, с какой стороны ее смотреть.

```
fig = plt.figure()
ax = Axes3D(fig)
ax.elev, ax.azim = 45, 30
ax.plot(x, y, z)
plt.show()
```



Часть 1.9. Поверхности

Поверхности задаются параметрические: $x=x(u,v)$, $y=y(u,v)$, $z=z(u,v)$.

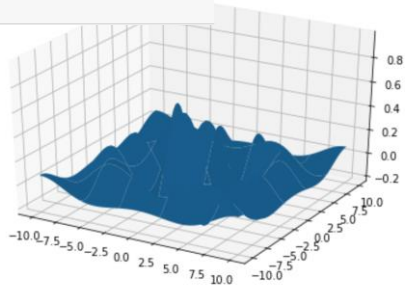
Если мы хотим построить явную поверхность $z=z(x,y)$, то удобно создать массивы $x=u$ и $y=v$ функцией `meshgrid`.

Пример 22

22.1.

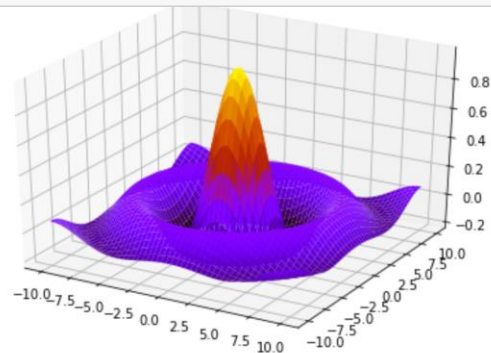
```
X = 10
N = 50
u = np.linspace(-X, X, N)
x, y = np.meshgrid(u, u)
r = np.sqrt(x**2 + y**2)
z = np.sin(r)/r

fig = plt.figure()
ax = Axes3D(fig)
ax.plot_surface(x, y, z, rstride=10, cstride=10)
plt.show()
```



22.2. Раскраска поверхностей. В методе `gnuplot` цвет зависит от высоты Z:

```
fig = plt.figure()
ax = Axes3D(fig)
ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap='gnuplot')
plt.show()
```

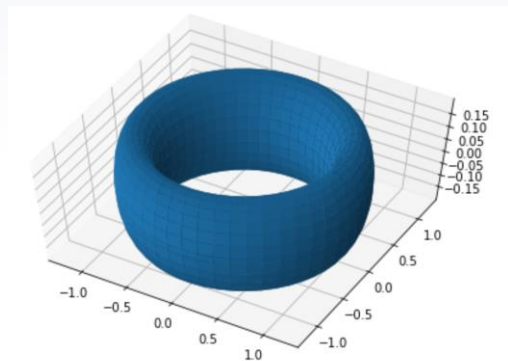


Поверхности

22.3. Построим бублик – параметрическую поверхность с параметрами θ , φ .

```
t = np.linspace(0, 2*np.pi, 50)
th, ph = np.meshgrid(t, t)
r = 0.2
x, y, z = (1 + r*np.cos(ph))*np.cos(th), (1 + r*np.cos(ph))*np.sin(th), r*np.sin(ph)

fig = plt.figure()
ax = Axes3D(fig)
ax.elev = 60
ax.plot_surface(x, y, z, rstride=2, cstride=1)
plt.show()
```



Замечание.

Чтобы сохранить получившийся рисунок можно воспользоваться методом `plt.savefig()` – сохранение текущей конфигурации текущего рисунка в графический файл с некоторым расширением (`png`, `jpeg`, `pdf` и др.), который можно задать через параметр `fmt`. Например, `plt.savefig('{ }.{}'.format(name, fmt), fmt='png')`.

Его нужно вызывать в конце исходного кода, после вызова всех других команд. Если в python-скрипте создать несколько рисунков `figure` и попытаться сохранить их одной командой `plt.savefig()`, то будет сохранён только последний рисунок `figure`.

Заключение

1. По ссылке <https://matplotlib.org/stable/gallery/index.html> Вы попадете в галерею, которая содержит примеры многих действий, которые вы можете делать в Matplotlib. Галерея демонстрирует разнообразие способов создания фигур. Для этого достаточно щелкнуть любое изображение, чтобы увидеть полное изображение и исходный код.

Просмотрите галерею и вскоре вы, как шеф-повар, будете смешивать и сочетать компоненты для создания своего шедевра!

2. **Matplotlib** позволяет строить двумерные графики практически всех нужных типов, с достаточно гибкой регулировкой их параметров; он также поддерживает основные типы трехмерных графиков, но для серьезной трехмерной визуализации данных лучше пользоваться более мощными специализированными системами.

При подготовке использованы материалы

1. [Matplotlib.org](https://matplotlib.org) – веб-сайт проекта – ресурс для документации библиотеки. Содержит галереи примеров, ответы на часто задаваемые вопросы, документацию по API и учебные пособия.
2. https://nbviewer.jupyter.org/github/matplotlib/AnatomyOfMatplotlib/blob/master/AnatomyOfMatplotlib-Part1-Figures_Subplots_and_layouts.ipynb
3. Как с помощью Matplotlib рисовать линии уровня: <https://jenyay.net/Matplotlib/Contour>
4. <https://inp.nsk.su/~grozin/python/>
5. https://mipt-stats.gitlab.io/courses/python/06_matplotlib.html

Рекомендуемая литература

1. Официальная документация

1. Библиотека matplotlib. URL: <http://matplotlib.org>.
2. Библиотека NumPy. URL: <http://www.numpy.org>.

2. Python

1. Библиотека Pandas. URL: <http://pandas.pydata.org>.
2. Библиотека Seaborn. URL: <http://stanford.edu/~mwaskom/software/seaborn>.
3. Шабанов П.А. Научная графика в Python.
URL: https://github.com/whitehorn/Scientific_graphics_in_python.
1. Обзор архитектуры библиотеки matplotlib.
URL: <http://rus-linux.net/MyLDP/BOOKS/Architecture-Open-Source-Applications/Vol-2/matplotlib-02.html>.
1. Визуализация данных с использованием matplotlib.
URL: <http://itnovella.ru/itnovella/2013/10/21/matplotlib.html>.
1. Работа с данными среднего размера в Python. Pandas и Seaborn. URL: <https://habrahabr.ru/post/266289/>.
2. Основы NumPy в Python. URL: <http://pythonworld.ru/numpy>.
3. Python Data Visualizations (pandas, matplotlib, seaborn).
URL: <https://www.kaggle.com/benhamner/d/uciml/iris/python-data-visualizations>.

1. Прочее

1. Визуализация данных. URL: https://ru.wikipedia.org/wiki/Визуализация_данных.
2. Диаграмма. URL: <https://ru.wikipedia.org/wiki/Диаграмма>.
3. Matplotlib. URL: <https://ru.wikipedia.org/wiki/Matplotlib>.
4. Как выбрать диаграмму?! URL: <http://www.fdfgroup.ru/?id=279>.
5. Типы диаграмм.
URL: http://www.plam.ru/bisliit/govori_na_jazyke_diagramm_posobie_po_vizualnym_kommunikacijam/p3.php.
1. Типы диаграмм. URL: <https://support.office.com/ru-ru/article/Типы-диаграмм-10b5a769-100d-4e41-9b0f-20df0544a683>.
2. Рост хоккеистов: анализируем данные всех чемпионатов мира в текущем веке. URL: <https://habrahabr.ru/post/301340/>.

Задание

1. Для закрепления пройденного материала необходимо выполнить:
Самостоятельную работу по теме 1.4. **Визуализация с помощью библиотеки Matplotlib:**
 - Файл [Самостоятельная работа-1\(Тема-1.4\).doc](#) содержит программные коды для построения рассмотренных в лекции графиков.
2. Прослушать видео Практических занятий по теме 1.4, которые содержат подробные материалы по визуализации данных с помощью библиотеки **Matplotlib**.
3. Выполнить лабораторную работу по разделу 1.4.