

## Библиотека matplotlib



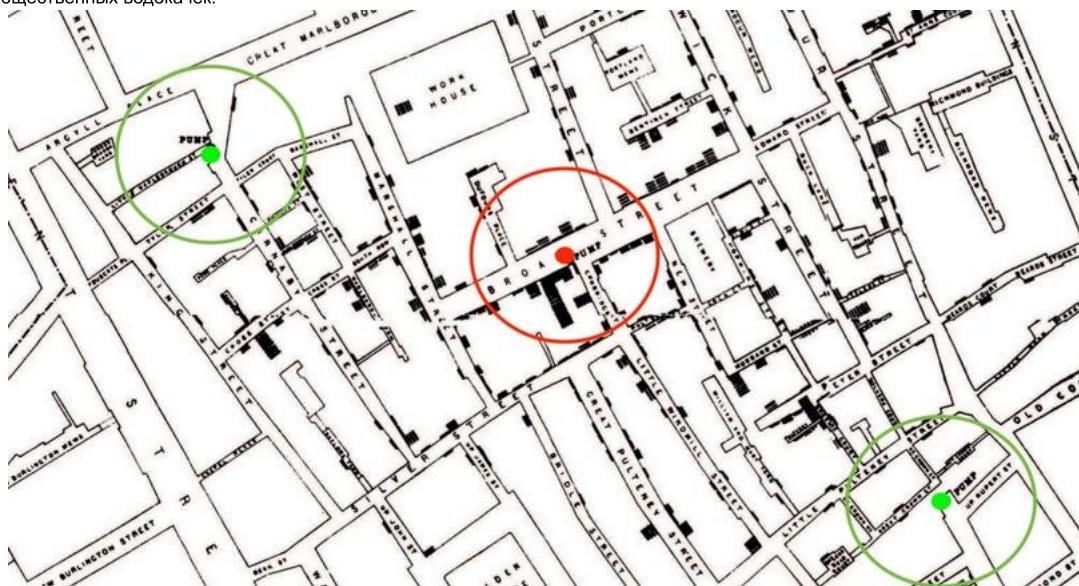
### Введение

Зачем нужна визуализация данных?

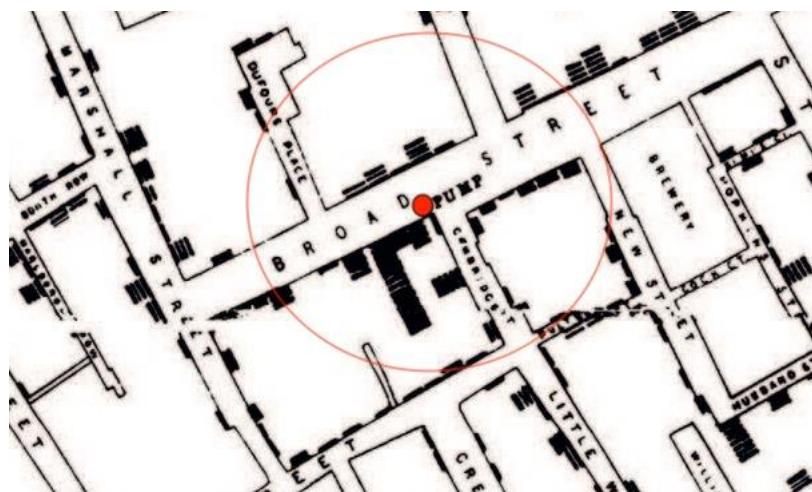
История про джона сноу, как пример зачем нужна визуализация <https://sysblok.ru/visual/dzhon-snou-vs-holera/> (<https://sysblok.ru/visual/dzhon-snou-vs-holera/>)

В 1854 году в Лондоне в районе Сохо вспыхнула эпидемия холеры — болезни смертельной и с отвратительными симптомами. «Эта экзотическая захватчица превратила просвещенных европейцев в орду дикарей» (историк Ричард Эванс).

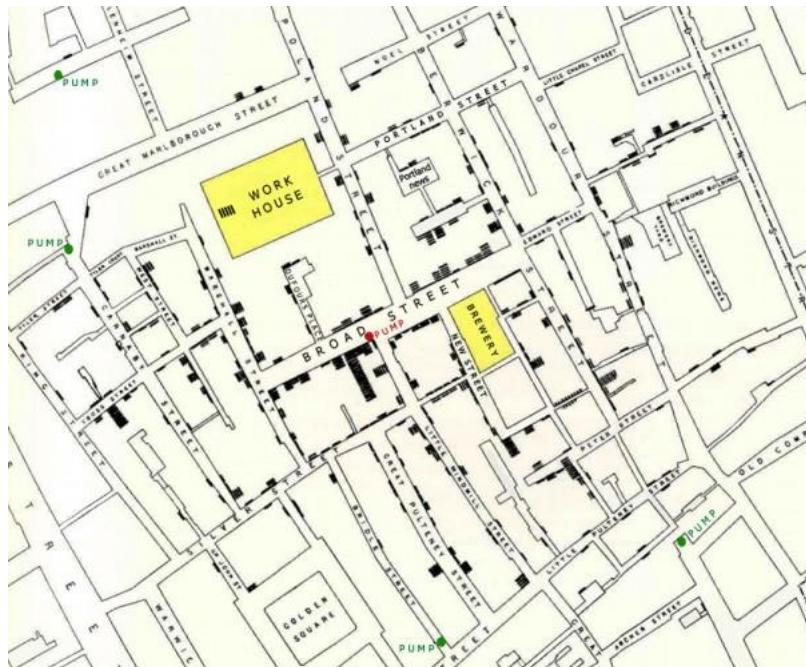
Д-р Джон Сноу — британский врач, именно ему первому в голову пришла идея, что распространение холеры связано с водой. Но чтобы доказать свою теорию правительству, ему пришлось обойти все дома в районе пандемии, составить статистику количества зараженных в каждом доме и сопоставить с расположением общественных водокачек:



Столбики внутри здания — число зараженных, круги — области вокруг водокачек, точки — сами водокачки.



Глядя на карту, можно увидеть, что основной очаг заражения локализуется вокруг центрального насоса, в то время как в домах рядом с насосами на северо-западе и юго-востоке случаев заражения сильно меньше.



Единственным вопросом оставалось, почему в местной пивоварне и общежитии для рабочих случаи заражения были единичными. Ответ прост: рабочие, жившие в общежитии, работали преимущественно в пивоварне, хозяин которой не только имел собственную водонапорную скважину в подвале, но и разрешал пить производимое пиво в течение рабочего дня.

Составив эту подробную карту, Джону Сноу удалось убедить правительство в своей правоте. Рукоять центрального насоса сломали, и эпидемия вскоре остановилась. В честь Джона Сноу в Лондоне установлен памятник в виде водопадки и открыт паб, но главное — его исследования и метод в последующем использовались для обнаружения очагов заболевания в других странах при вспышках холеры.

Источник: <https://sysblok.ru/visual/dzhon-snou-vs-holera/> (<https://sysblok.ru/visual/dzhon-snou-vs-holera/>)

#### Визуализация данных

- исследование закономерностей данных (существует ли в данных закономерность?)
- подготовка моделей (шумы, признаки, гипотезы)
- результаты работы моделей (мониторинг, количественные графики и т.д.)

#### Что можно визуализировать?

- количественные характеристики
- изменение количественных характеристик во времени
- изменение количественных характеристик в пространстве
- распределения характеристик
- прочее

## § 1 Назначение библиотеки

По данным учёных (причём не только британских!), более 80% информации человек получает с помощью зрения. В процессе анализа данных мы перерабатываем и создаём огромное количество информации, но чтобы донести эту информацию до окружающих, важно уметь делать так, чтобы информация радовала глаз.

**Графики**, диаграммы, тепловые карты и другие приёмы визуализации данных не просто **украшают любой отчёт**, но и помогают нам сделать результаты анализа **более понятными, убедительными и удобными для восприятия**.

Если перед вами не стоит глобальная задача подготовить графический материал для годового отчёта большого предприятия, а нужно просто сделать более понятными результаты выполненного анализа данных, то нет необходимости в установке специальных профессиональных библиотек.

Есть несколько пакетов для построения графиков. Один из наиболее популярных — `matplotlib`. Изначально `matplotlib` планировался как свободная альтернатива MATLAB, где в одной среде имелись бы средства как для рисования, так и для численного анализа.

Если в `jupyter notebook` выполнить специальную магическую команду `%matplotlib inline`, то графики будут строиться в том же окне браузера. Есть другие варианты, в которых графики показываются в отдельных окнах. Это удобно для трёхмерных графиков — тогда их можно вращать мышкой (в случае `inline` графиков это невозможно).

Графики можно также сохранять в файлы, как в векторных форматах (`eps`, `pdf`, `svg`), так и в растровых (`png`, `jpg`). `matplotlib` позволяет строить двумерные графики практически всех нужных типов, с достаточно гибкой регулировкой их параметров; он также поддерживает основные типы трёхмерных графиков, но для серьёзной трёхмерной визуализации данных лучше пользоваться более мощными специализированными системами.

## § 2 Импорт библиотеки Matplotlib

[Matplotlib](https://matplotlib.org/stable/index.html) (<https://matplotlib.org/stable/index.html>) – библиотека для визуализации данных. Предоставляет средства для вывода двумерной (2D) и трехмерной (3D) графики, а также анимированных рисунков и диаграмм. Отличается простотой использования – для построения весьма сложных и красочно оформленных диаграмм достаточно нескольких строк кода. При этом качество получаемых изображений более чем достаточно для их публикования. В ней поддерживается множество видов визуализации: линейные, столбчатые, точечные и круговые диаграммы, гистограммы, тепловые карты и другие.

Как и любой пакет `matplotlib` можно установить, используя утилиту `pip`:

```
pip install matplotlib
pip install matplotlib --upgrade
```

Matplotlib состоит из множества модулей. Основной из них – высокоразвитый модуль `matplotlib.pyplot`. Pyplot – интерфейс для построения графиков простых функций, он позволяет пользователю сосредоточиться на выборе готовых решений и настройке базовых параметров рисунка. Наиболее распространенный метод его вызова состоит в выполнении следующей инструкции: `import matplotlib.pyplot as plt`.

При работе с графиками дополнительно используются библиотеки NumPy и Pandas (пока не будем использовать).

Т.о. если вам предстоит визуализировать какие-то данные, то следующие строки кода смело вставляйте в свой ноутбук:

```
# импорт библиотек
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

Модуль `matplotlib.pyplot` обычно переименовывается как `plt`, `numpy` как `np` (мы используем его для генерации данных). *Magic*-команда `%matplotlib inline` необходима для вывода графика на экран *Jupyter Notebook*.

Ввод [1]:

```
# импортируем библиотеки numpy и matplotlib
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

Иногда потребуется определить версию библиотеки, что можно сделать командами:

```
import matplotlib

matplotlib.__version__
```

Ввод [2]:

```
import matplotlib as mpl

# Вывод на экран текущей версии библиотеки matplotlib
print('Current version on matplotlib library is', mpl.__version__)
```

Current version on matplotlib library is 3.3.4

## § 3 Варианты отображения графиков: `%matplotlib notebook` и `%matplotlib inline`

Магическая команда `%matplotlib` определяет, каким образом будут отображаться графики в *Jupyter Notebook*. Скорее всего, эту команду вам придётся писать в начале каждого ноутбука, где будут строиться графики. Есть два основных варианта отображения графиков.

- `notebook` – для визуализации графика в интерактивном режиме. В этом режиме в ноутбуке открывается интерактивное окошко, в котором можно масштабировать участки графика по своему усмотрению. Это удобно, когда хочется рассмотреть детали, но требовательно к ресурсам при сложных визуализациях. А ещё работает не всегда.
- `inline` – для получения статичного изображения. В этом режиме в ноутбук просто вставляется статическая картинка, это указанием среде *Jupyter* встраивать в его документы графику, генерируемую библиотекой `matplotlib`. В этом режиме все создающие графики блоки в блокноте будут включать PNG-изображения итогового графика. Именно этим режимом вы будете пользоваться чаще всего.

Разберёмся, в чём между ними разница, запустив ячейки

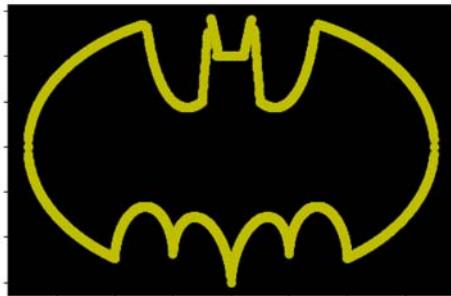
Ввод [3]:

```
import numpy as np

def plot_batman():
```

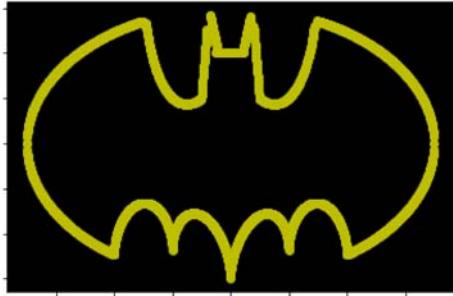
Ввод [4]:

```
%matplotlib notebook
plot_batman()
<IPython.core.display.Javascript object>
```



Ввод [5]:

```
%matplotlib inline
plot_batman()
```



При отображении изображения в отдельном окне кроме непосредственно изображения, графическое окно содержит специальные функциональные кнопки (интерактивную навигацию, пронумерована на рисунке), их назначение и возможности приведены в таблице.

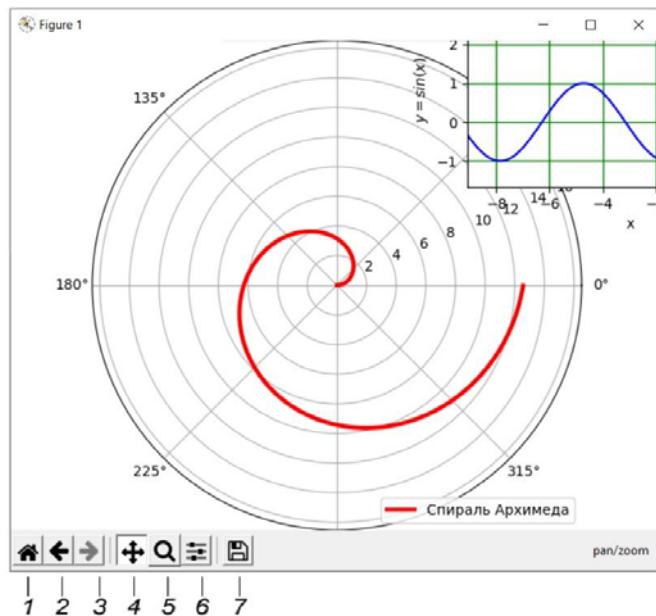


Таблица. Кнопки интерактивной навигации графического окна

Номер	Описание
1	Возвращает график к первоначальному состоянию (вид при первом запуске)
2, 3	Перемещает назад / вперед по истории изменения графика. Например, после увеличения определенной области можно вернуться к предыдущему, не увеличенному виду, используя кнопку 2
4	Перетаскивает (левая кнопка мыши) или масштабирует (правая кнопка мыши) график
5	Масштабирует заданную прямоугольную область графика
6	Открывает дополнительные настройки окна (например, отступы от краев)

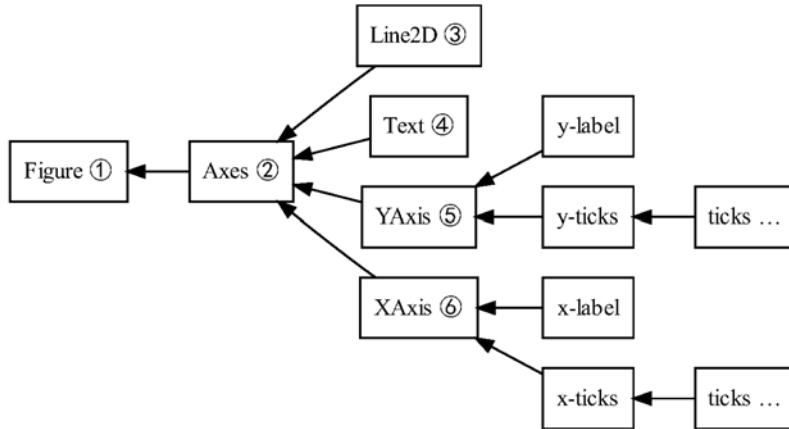
Номер	Описание
7	Вызывает диалог сохранения графика в файл

## § 4 Иерархия объектов в Matplotlib

Обычно рисунок в `matplotlib` представляет собой прямоугольную область, заданную в относительных координатах: от 0 до 1 включительно по обеим осям.

Изображение в `matplotlib` создается путем последовательного вызова команд (функций) этого модуля. Создание рисунка в `matplotlib` схоже с рисованием в реальной жизни. Подобно тому, как рисует художник: нужно взять основу (холст или бумагу), инструменты (кисти или карандаши), иметь представление о будущем рисунке (что именно он будет рисовать) и, наконец, выполнить все это и нарисовать рисунок деталь за деталью.

Библиотека `matplotlib` организована иерархически.



Главной единицей (объектом самого высокого уровня) при работе с `matplotlib` является рисунок ( `Figure` ). Любой рисунок в `matplotlib` имеет вложенную структуру и чем-то напоминает матрёшку.

### Рисунок ( `Figure` )

Рисунок `Figure` является объектом самого верхнего уровня, на котором располагаются одна или несколько областей рисования ( `Axes` ), элементы рисунка `Artists` (заголовки, легенда и т.д.) и основа-холст ( `Canvas` ). На рисунке может быть несколько областей рисования `Axes` , но данная область рисования `Axes` может принадлежать только одному рисунку `Figure` .

### Область рисования ( `Axes` )

Область рисования `Axes` является объектом среднего уровня, который является, наверное, главным объектом работы с графикой `matplotlib` в объектно-ориентированном стиле. Это то, что ассоциируется со словом `plot` , это часть изображения с пространством данных. Каждая область рисования `Axes` содержит две (или три в случае трёхмерных данных) координатных оси (`Axis` объектов), которые упорядочивают отображение данных.

### Координатная ось ( `Axis` )

Координатная ось `Axis` являются объектом среднего уровня, которые определяют область изменения данных, на них наносятся деления `ticks` и подписи к делениям `ticklabels` . Расположение делений определяется объектом `Locator` , а подписи делений обрабатывает объект `Formatter` . Конфигурация координатных осей заключается в комбинировании различных свойств объектов `Locator` и `Formatter` .

### Элементы рисунка ( `Artists` )

Элементы рисунка `Artists` являются как бы красной линией для всех иерархических уровней. Практически всё, что отображается на рисунке является элементом рисунка ( `Artist` ), даже объекты `Figure` , `Axes` и `Axis` . Элементы рисунка `Artists` включают в себя такие простые объекты как текст ( `Text` ), плоская линия ( `Line2D` ), фигура ( `Patch` ) и другие.

**Графические объекты** (точки, линии, фигуры и т.д.) **последовательно накладываются один на другой**, если они занимают общие области на рисунке.

При создании рисунка в `matplotlib` обычно поступают так: создают экземпляр класса `Figure` (`Figure instance`), на котором выделяют одну или несколько областей `Axes` ( или экземпляров `Subplot` ), и используют вспомогательные методы экземпляра класса `Axes` (`Axes instance`) для создания графических примитивов (`primitives`). Если автоматически подобранные характеристики координатной сетки, делений и их подписей не устраивают пользователя, то они настраиваются с помощью экземпляров контейнеров `Axis` и `Tick` , которые всегда присутствуют на созданной области рисования `Axes` .

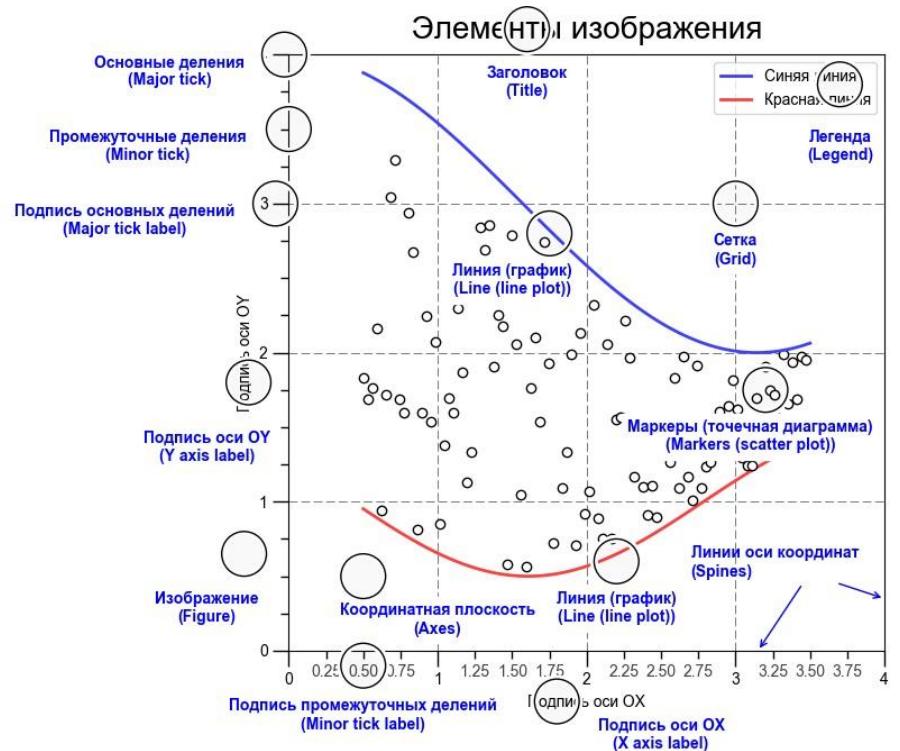
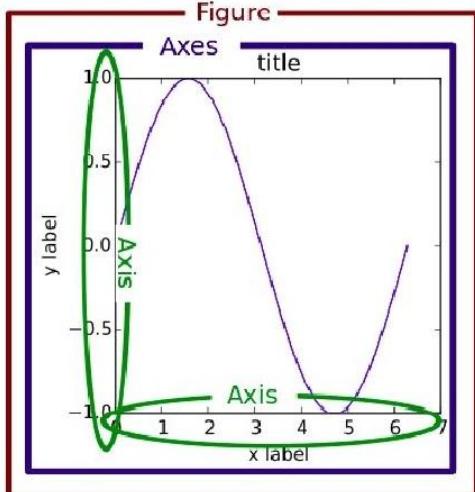
Т.о. пользовательская работа подразумевает операции с разными уровнями этой матрёшки:

`Figure` (Рисунок) -> `Axes` (Область рисования) -> `Axis` (Координатная ось)

Объект `Figure` – это **самый важный внешний контейнер** для графики `matplotlib` , который может включать в себя один или несколько объектов `Axes` (настоящих графиков), причем название `Axes` показывает **не множественное число «оси»** (как ожидается из перевода).

`Axes` – это **индивидуальный график или диаграмма** . `Axes` – это та область на которой чаще всего и отражаются графики, а так же все вспомогательные атрибуты (линии сетки, метки, указатели и т.д.). Часто, установка этой области сопровождается с вызовом `subplot` , который и помещает `Axes` на регулярную сетку. Поэтому, так же часто `Axes` и `Subplot` можно считать синонимами.

Каждая область `Axes` содержит `XAxis` и `YAxis` . Они содержат, деления, метки и прочие вспомогательные атрибуты.



В том что Figure и Axes это разные области можно легко убедиться если изменить их цвет:

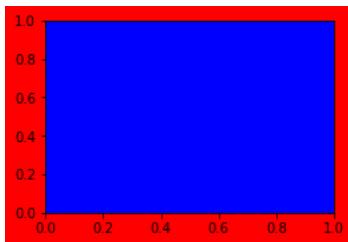
Ввод [6]:

```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(3, 2)) # создали область Figure (экземпляр класса figure)
fig.set(facecolor = 'red')

ax = fig.add_axes([0, 0, 1, 1]) # добавили к Figure область Axes
ax.set(facecolor = 'blue')

plt.show()
```



На рисунке может быть несколько областей рисования Axes , но данная область рисования Axes может принадлежать только одному рисунку Figure . Каждый объект Axes содержит две (или три в случае трёхмерных данных) координатных оси Ax's , которые упорядочивают отображение данных.

Под объектами Axes , в порядке иерархии расположены меньшие объекты, такие как индивидуальные линии, отметки, легенды, заголовки и текстовые боксы. Основное назначение Axes состоит в том, что на него наносится графика – эти самые линии, отметки, легенды, заголовки и текстовые боксы и так далее. Практически **каждый «элемент» диаграммы – это собственный манипулируемый объект Python**, вплоть до ярлыков и отметок.

**Создать рисунок figure** позволяет инструкция: `plt.figure()` . Чтобы результат рисования (т.е. текущее состояние рисунка), **отразилось на экране**, можно воспользоваться командой `plt.show()` , при этом будут показаны все рисунки ( `figures` ), которые были созданы. После создания рисунка, т.е. экземпляра класса `matplotlib.figure` , результат выполнения всех графических команд будут отображаться именно в нём. Объект `Figure` может **автоматически создаваться при первом выполнении какой-либо графической функции**.

В `matplotlib` работает правило "текущей области" (`current axes`), которое означает, что все графические элементы наносятся на текущую область рисования. Несмотря на то, что областей рисования может быть несколько, один из них всегда является текущей. При этом последующие объекты перекрывают предыдущие, если они занимают общее участки на рисунке (регулируется параметром `zorder` ).

Существует несколько подходов к созданию графиков в `Matplotlib`:

- **структурированный** – ориентированный на структуру;
- **неструктурированный (объектно-ориентированный)** – ориентированный на объект, он более гибкий и становится более удобным при усложнении отображаемых графиков.

#### 4.1 Объектно-ориентированный подход к созданию графиков

**Объектно-ориентированный подход** – наиболее прозрачный и мощный по функционалу. Процесс работы над графиком начинается с создания объекта, содержащего необходимую информацию и настройки. К этому объекту последовательно добавляются координатная плоскость и другие графические объекты.

**Объектно-ориентированный подход** к построению изображения, как правило, включает следующие шаги:

- Шаг 1. Создать объект `Figure` позволяет инструкция: `fig = plt.figure()`.
- Шаг 2. Используя объект `Figure`, добавить координатную плоскость `Axes` (одну или несколько) командой `fig.add_axes([...])`.
- Шаг 3. Используя методы `Axes`, добавить на изображение графические примитивы, указывая их параметры.
- Шаг 4. Отобразить и (или сохранить) изображение командой `plt.show()` (`plt.savefig('file_name')`).

Ввод [7]:

```
# импорт библиотек
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

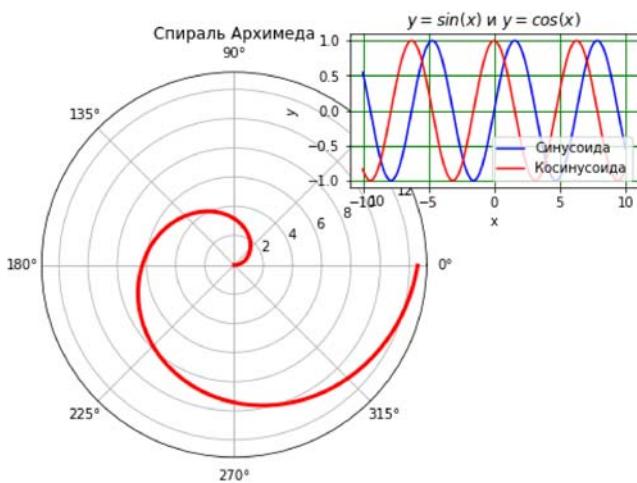
x = np.arange(-10, 10.01, 0.01)
phi = np.arange(0, 2*np.pi, 0.01)

fig = plt.figure()

# Добавление на рисунок круговой области рисования
axes_main = fig.add_axes([0, 0, 1, 1], polar=True)
axes_main.plot(phi, 2*phi, label='Сpirаль Архимеда', color='red', linewidth=3)
axes_main.set_title('Сpirаль Архимеда')

# Добавление на рисунок прямоугольной (по умолчанию) области рисования
axes_sub = fig.add_axes([0.7, 0.7, 0.5, 0.4])
axes_sub.plot(x, np.sin(x), label = 'Синусоида', color='blue')
axes_sub.plot(x, np.cos(x), label = 'Косинусоида', color='red')
axes_sub.grid(axis='both', color='g', linestyle='-', linewidth=1)
axes_sub.set_title('$y=\sin(x)$ и $y=\cos(x)$')
axes_sub.set_xlabel('x')
axes_sub.set_ylabel('y')
axes_sub.legend(loc=4)

plt.show()
```



В строке `fig = plt.figure()` создаётся объект с именем `fig`, к которому применяются все настройки.

В строке `axes_main = fig.add_axes([0,0,1, 1], polar=True)` создаётся координатная плоскость (`axes`) с полярными координатами (где координатными осями являются радиус и угол наклона), которая начинается в левом нижнем углу без отступов (координаты 0, 0) и занимает отведённое место в области (ширина и высота равны 1).

На одном объекте можно размещать несколько систем координат, что позволит отобразить вспомогательную информацию на основном графике. Для добавления второй системы координат необходимо повторно применить к объекту `fig` метод `add_axes`, указав новое имя для второй системы координат, что и было сделано в строке с кодом: `axes_sub = fig.add_axes([0.7,0.7, 0.5, 0.4])`.

Первые два числовых параметра метода `add_axes`, указанные при создании систем координат, – это отступ снизу и слева, а следующие два – ширина и высота относительно ширины и высоты всего пространства (в долях единицы) для построения графика.

Теперь можем разместить в созданных нами координатной плоскости любую графическую информацию. В примере для созданных областей рисования были добавлены заголовки, подписи осей, сетка, легенда.

**Для построения графика** используется метод `plot()`, которому в случае заданий значений по осям `x` и `y` необходимо передать два списка. **Важно:** два первых аргумента функции `matplotlib.pyplot.plot` принимают **списки одинаковой длины**.

Параметры внешнего вида можно указывать вместе с данными внутри `ax.plot()`, `ax.scatter()` и `ax.set()`. **Это делает код более читаемым и лаконичным.**

Ввод [8]:

```

import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])

x = np.arange(-5, 5, 0.25)
ax.plot(x, np.sin(x),
        color='red',
        linewidth = 3,
        marker = '^',
        markeredgewidth=2,
        markersize=10,
        markerfacecolor='yellow',
        label=r'$y=\sin(x)$')

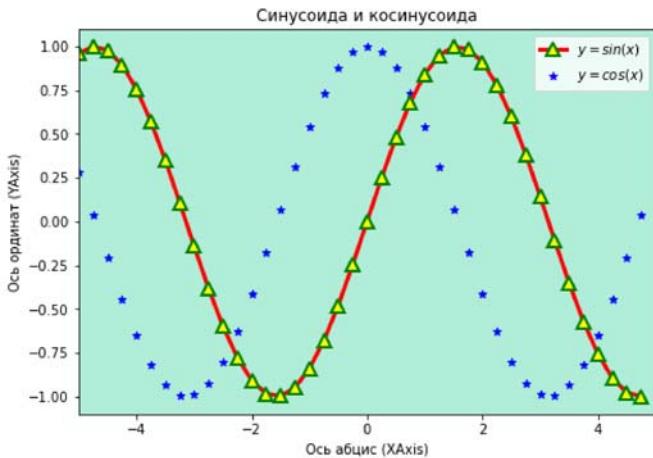
ax.scatter(x, np.cos(x),
           color = 'blue',
           marker = '*',
           label=r'$y=\cos(x)$')

ax.set(facecolor = "#B0EED9",
      xlim = [-5, 5],
      ylim = [-1.1, 1.1],
      title = 'Синусоида и косинусоида',
      xlabel = 'Ось абсцис (XAxis)',
      ylabel = 'Ось ординат (YAxis)')

ax.legend(loc='best')

plt.show()

```



## 4.2 Подход к созданию графиков, ориентированный на его структуру

Практически все функции `pyplot`, такие как `plt.plot()`, так или иначе, ссылаются на существующий объект `Figure` и объект `Axes`, или создают их, если какой-либо из них не существует. Поэтому считается, что `plt.plot()` – это интерфейс, который неявно отслеживает текущую фигуру, а поскольку существует только один объект `Figure` или `Axes`, который вы используете в данное время, то вам не нужна явная ссылка на этот объект.

И поскольку вызов `plt.plot()` – это удобный способ получения текущего объекта `Axes` текущего объекта `Figure`, то утверждается, что структурированный интерфейс всегда «неявно отслеживает» график, на который он хочет ссылаться.

Помните о том, что структурированный подход (`plt.plot()`) неявно имеет представление о текущем объекте `Figure` и `Axes`.

Для построения графика (аналогично объектно-ориентированному подходу) используется метод `plot()`. В самом минимальном варианте его можно использовать без параметров, что приведет к отображению пустого поля. Если же в качестве параметра функции `plot()` передать список, то значения из этого списка будут отложены по оси ординат (ось `y`), а по оси абсцисс (ось `x`) будут отложены индексы элементов массива.

При использовании структурированного подхода, **начинать и заканчивать построения можно командами:**

```

plt.figure(figsize=(n, m))
. . .
plt.show()

```

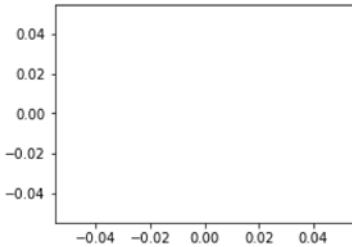
Между этими командами необходимо располагать методы, добавляющие на изображение графические примитивы, с указанием их параметров. Аргумент `figsize` функции `figure()` позволяет **задать размер подложки** – кортеж из двух `float` элементов, определяющих **высоту и ширину подложки в дюймах** (`(n, m)`). Если этого не сделать, то ее значения ее размеры будут заданы по умолчанию.

Ввод [9]:

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

%matplotlib inline

plt.figure(figsize=(4, 3))
plt.plot()
plt.show()
```



Передав функции `plot()` два списка, можно отобразить соответствие значений по осям `x` и `y`.

Ввод [10]:

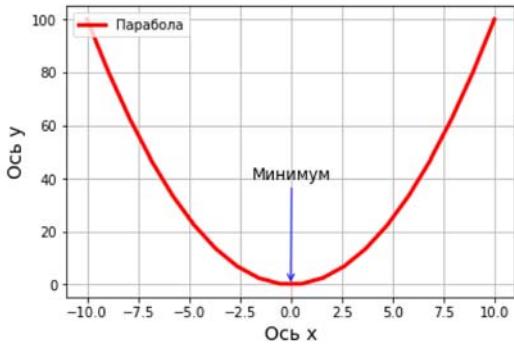
```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

x = np.linspace(-10, 10, 20)
y = x**2

plt.plot(x, y, color='red', linewidth=3, label='Парабола');
plt.grid() # сетка
plt.xlabel('Ось x', fontsize=14) # добавляем подпись к оси абсцисс
plt.ylabel('Ось y', fontsize=14) # добавляем подпись к оси ординат
plt.title('График функций $y = x^2$', fontsize=16,
          y=1.05, color="#CC00CC"); # добавляем заголовок
plt.annotate("Минимум", xy=(0.,0.), xytext=(-1.9, 40),
            arrowprops=dict(arrowstyle = '->',color = 'blue'),
            fontsize=12, ) # добавляем аннотации
plt.legend(loc=2)
plt.show()
```

График функций  $y = x^2$



## § 5 Настройка внешнего вида линейного графика: линии и маркеры

### 5.1 Построение линейного графика, функция `plot()`

Линейный график – это один из наиболее часто используемых видов графиков для визуализации данных.

Линии, посредством которых происходит отрисовка графиков, характеризуются следующими параметрами: *тип линии*, её *толщина*, *цвет*, *форма*, наличие *маркера* на линии, его *форма*, *цвет маркера* и т.д.

Параметры, которые отвечают за отображение графика можно задать непосредственно в самой функции `plot()`:

```
plt.plot(x, y, linestyle='--', color='green', marker='o',
         markersize=10, markerfacecolor='red')
```

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

```
plot([x1], y1, [fmt], *, data=None, **kwargs)

plot([x1], y1, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

Рассмотрим аргументы функции `plot()`:

- `x1, x2, ...` – набор данных ( `array` ) для оси абсцисс первого, второго и т.д. графика;
- `y1, y2, ...` – набор данных ( `array` ) 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, полный список можно посмотреть по ссылке: [https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.lines.Line2D.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.lines.Line2D.html) ([https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.lines.Line2D.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.lines.Line2D.html)).

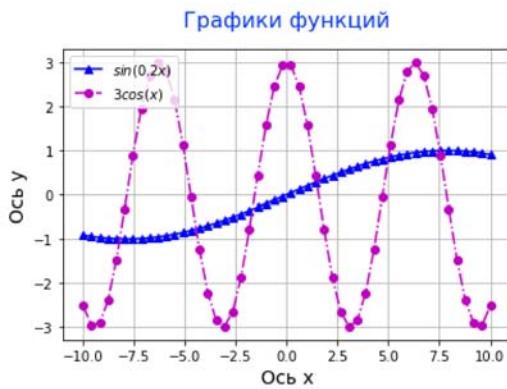
Ввод [11]:

```
# На одном рисунке можно построить несколько графиков функций
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

x = np.linspace(-10, 10, 50)

plt.plot(x, np.sin(0.2*x), '^--b', x, 3*np.cos(x), 'o-.m')
plt.grid() # сетка
plt.xlabel('Ось x', fontsize=14) # добавляем подпись к оси абсцисс
plt.ylabel('Ось y', fontsize=14) # добавляем подпись к оси ординат
plt.title('Графики функций', fontsize=16, y=1.05,
          color='#0036FF'); # добавляем заголовок
plt.legend(['$sin(0.2x)$', '$3cos(x)$'], loc=2)
plt.show()
```



## 5.2 Стиль линии графика linestyle

Стиль линии графика задается через параметр `linestyle` или `ls` , который может принимать значения из приведенной ниже таблицы.

Таблица. Стили линии линейного графика

Значение параметра <code>linestyle</code>	Описание
- или solid	Непрерывная линия
-- или dashed	Штриховая линия
-. или dashdot	Штрихпунктирная линия
:	Пунктирная линия
None или `` или ``	Не отображать линию

Стиль линии можно передать сразу после списков с координатами без указания, что это параметр `linestyle` .

```
plt.plot(x, y1, ls='--')
plt.plot(x, y2, linestyle='-.')
```

## 5.3 Цвет линии color

Задание цвета линии графика производится через параметр `color` (или `c` , если использовать сокращенный вариант). Значение может быть представлено в одном из следующих форматов:

- RGB (0, 1, 0) или RGBA (0, 1, 0, 1) кортеж значений с плавающей точкой в диапазоне [0, 1] (пример: (0.1, 0.2, 0.3)), четвертое значение указывает  $\alpha$ -канал, т.е. степень прозрачности;
- RGB или RGBA значение в в шестнадцатеричном формате ( hex ) (пример: #00FF00 );
- оттенком серого – строковым представлением числа с плавающей точкой в диапазоне [0, 1] (пример: 0.7 );
- символ из набора { `b` , `g` , `r` , `c` , `m` , `y` , `k` , `w` }, соответствующий конкретному цвету графика: синий ( `b` ), зелёный ( `g` ), красный ( `r` ), бирюзовый ( `c` ), фиолетовый/ пурпурный ( `m` ), жёлтый ( `y` ), чёрный ( `k` ), белый ( `w` );
- имя цвета из палитры X11/CSS4;
- цвет из палитры xkcd (<https://xkcd.com/color/rgb/> (<https://xkcd.com/color/rgb/>)), должен начинаться с префикса `xkcd:` ;
- цвет из набора Tableau Color (палитра T10), должен начинаться с префикса `tab:` .

Если не указывать цвет, `matplotlib` выберет его самостоятельно.

Если цвет задается с помощью символа из набора { `b` , `g` , `r` , `c` , `m` , `y` , `k` , `w` }, то он может быть совмещен со стилем линии в рамках параметра `fmt` функции `plot()` , который задается в виде строки: `[marker][line][color]` .

Например, штриховая красная линия с круглым маркером будет задаваться так: `o--r`, а штрих-пунктирная зеленая с треугольным маркером так `^-.g`:

```
plt.plot(x, y1, 'o--r')
plt.plot(x, y2, '^-.g')
plt.plot(x, y3, ls=':', color="#00FF00")
plt.plot(x, y4, ls='.-', color=(0.3, 1, 0.7, 0.8))
plt.plot(x, y5, color='indigo')
```

В Matplotlib есть возможность задания цвета с помощью строки, описывающей цвет полным английским словом. Таких именованных цветов намного больше, чем однобуквенных именованных цветов. На рисунке представлена таблица со всеми существующими именованными цветами.

black	k	dimgray	dimgrey
gray	grey	darkgray	darkgrey
silver	lightgray	lightgrey	gainsboro
whitesmoke	w	white	snow
rosybrown	lightcoral	indianred	brown
firebrick	maroon	darkred	r
red	mistyrose	salmon	tomato
darksalmon	coral	orangered	lightsalmon
sienna	seashell	chocolate	saddlebrown
sandybrown	peachpuff	peru	linen
bisque	darkorange	burlywood	antiquewhite
tan	navajowhite	blanchedalmond	papayawhip
moccasin	orange	wheat	oldlace
floralwhite	darkgoldenrod	goldenrod	cornsilk
gold	lemonchiffon	khaki	palegoldenrod
darkkhaki	ivory	beige	lightyellow
lightgoldenrodyellow	olive	darkolivedgreen	yellow
olivedrab	yellowgreen	honeydew	greenyellow
chartreuse	lawngreen	forestgreen	darksseagreen
palegreen	lightgreen	green	limegreen
darkgreen	g	springgreen	mintcream
seagreen	mediumseagreen	aquaamarine	turquoise
mediumspringgreen	mediumaquamarine	azure	lightcyan
lightseagreen	mediumturquoise	darkslategray	teal
paleteurquoise	darkslategray	cyan	cyan
darkcyan	cadetblue	powderblue	lightblue
darkturquoise	skyblue	lightskyblue	steelblue
deepskyblue	dodgerblue	lightslategray	lightslategray
aliceblue	slategray	lightsteelblue	cornflowerblue
slategray	ghostwhite	lavender	midnightblue
royalblue	darkblue	mediumblue	b
navy	slateblue	darkslateblue	mediumslateblue
blue	rebeccapurple	blueviolet	indigo
mediumpurple	darkviolet	mediumorchid	thistle
darkorchid	violet	purple	darkmagenta
plum	fuchsia	hotpink	orchid
m	deeppink	pink	lavenderblush
mediumvioletred	crimson		lightpink

[Здесь](https://www.webucator.com/article/python-color-constants-module/) (<https://www.webucator.com/article/python-color-constants-module/>) можно посмотреть соответствие между различными способами задания цвета.

## 5.4 Толщина линии linewidth

Толщина линии `linewidth` или `lw` задается вещественным числом.

```
plt.plot(x, y1, 'o--r', lw=2.0)
plt.plot(x, y2, '^-.g', lw=2.5)
```

## 5.4 Тип маркера marker, его размер markersize и цвет

Тип маркера `marker` для точки графика (по умолчанию отсутствуют).

Таблица. Некоторые (чаще используемые) типы маркеров

Значение параметра <code>marker</code>	Описание
.	Точка (point marker)
,	Пиксель (pixel marker)
o	Окружность (circle marker)
v	Треугольник, направленный вниз (triangle_down marker)
^	Треугольник, направленный вверх (triangle_up marker)
<	Треугольник, направленный влево (triangle_left marker)
>	Треугольник, направленный вправо (triangle_right marker)
s	Квадрат (square marker)
*	Звезда (star marker)
+	Плюс (plus marker)
x	X-образный маркер (x marker)
D	Ромб (diamond marker)
d	Ромб (thin_diamond marker)

Полный список маркеров можно посмотреть в документации.

Цвета маркеров определяются атрибутами `markeredgecolor`, `markeredgewith`, `markerfacecolor`, `markerfacecoloralt`:

- `markeredgecolor` или `mec` – цвет границы маркера (`color`);
- `markeredgewith` или `mew` – толщина границы маркера (`float`);
- `markerfacecolor` или `mfc` – цвет заливки маркера (`color`);
- `markerfacecoloralt` или `mfcalt` – альтернативный цвет заливки маркера (`color`).

**Размер маркера** определяются атрибутом `markersize` или `ms` действительное число ( `float` ).

```
plt.plot(x, x*x, '--r', marker='v', ms=10, mfc='#0000FF')
```

В случае, когда нужно задать интервал отображения маркеров, используется параметр `markevery` , который может принимать одно из следующих значений:

- `None` – отображаться будет каждая точка;
- `N` – отображаться будет каждая `N`-я точка;
- `(start, N)` – отображается каждая `N`-я точка начиная с точки `start`;
- `slice(start, end, N)` – отображается каждая `N`-я точка в интервале от `start` до `end` ;
- `[i, j, m, n]` – будут отображены только точки `i, j, m, n` .

```
plt.plot(x, x*x, '--r', marker='v', ms=10, mfc='#0000FF', markevery=(1, 3))
```

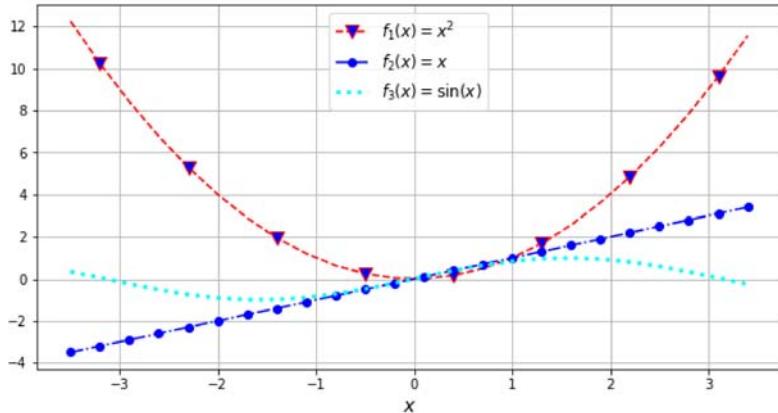
Ввод [12]:

```
# Настройка внешнего вида линейного графика
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-3.5, 3.5, 0.3)
plt.figure(figsize=(10, 5))

plt.plot(x, x*x, '--r', marker='v', ms=10, mfc='#0000FF',
         markevery=(1, 3), label=r'$f_1(x)=x^2$')
plt.plot(x, x, 'o-b', label=r'$f_2(x)=x$')
plt.plot(x, np.sin(x), ls=':', lw=3, c='#00FFFF', label=r'$f_3(x)=\sin(x)$')
plt.xlabel(r'$x$', fontsize=14)
plt.grid()

plt.legend(loc='best', fontsize=12)
plt.show()
```



Если при построении параметр `linestyle=''` , то точки кривой не соединяются линиями.

При построении нескольких кривых на одном чертеже можно использовать маркеры разных типов. Тип определяется строкой из одного символа, который чем-то похож на нужный маркер. В добавок к стандартным маркерам, можно определить самодельные.

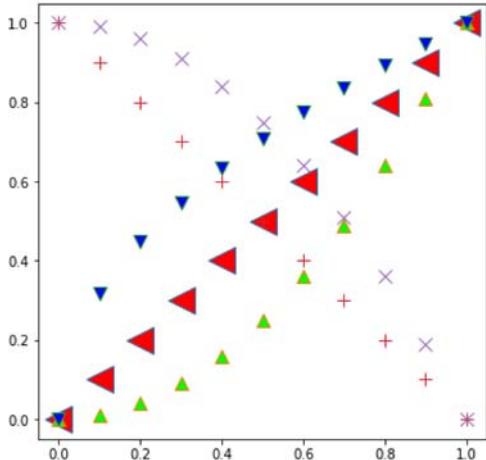
Ввод [13]:

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

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

plt.figure(figsize=(6, 6))
plt.plot(x, x, ls='', marker='<', ms=20, mfc='#FF0000')
plt.plot(x, x ** 2, ls='', marker='^', ms=10, mfc='#00FF00')
plt.plot(x, x ** (1/2), ls='', marker='v', ms=10, mfc='#0000FF')
plt.plot(x, 1 - x, ls='', marker='+', ms=10, mfc='#0F0F00')
plt.plot(x, 1 - x ** 2, ls='', marker='x', ms=10, mfc='#0F000F')
plt.axis([-0.05, 1.05, -0.05, 1.05])

plt.show()
```



## § 6 Настройка элементов графика

### 6.1 Настройка основных и вспомогательных делений

В `matplotlib` существуют **главные деления** (`major ticks`) и **вспомогательные** (`minor ticks`) для каждой координатной оси. По умолчанию рисуются только главные деления и связанные с ними линии сетки `grid`. Вспомогательная сетка связана с делениями координатных осей (`ticks`), которые определяются автоматически исходя из значений выборки.

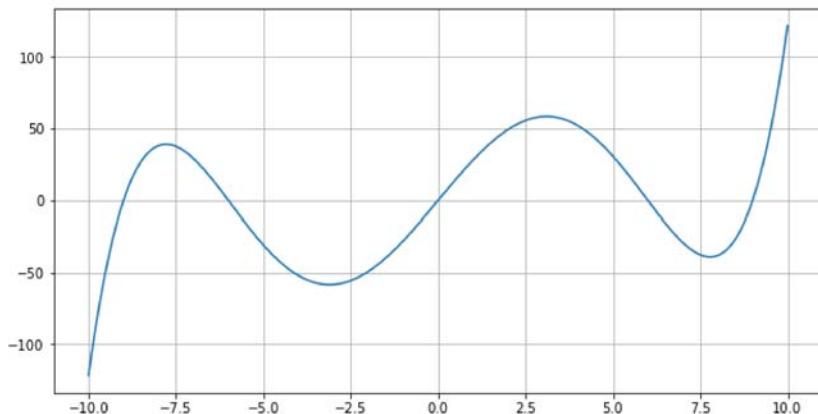
Ввод [54]:

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

x = np.linspace(-10, 10, 200)
y = 0.01*(x + 9)*(x + 6)*(x - 6)*(x - 9) * x

fig = plt.figure(figsize=(8, 4))

# Добавим на рисунок область рисования
ax = fig.add_axes([0, 0, 1, 1])
ax.plot(x, y)
ax.grid()
plt.show()
```



**Деления** – это экземпляры класса `matplotlib.axis.Tick`, которые визуализируют деления (размер, цвет, толщину и т.д.) и подписи к ним. Деления создаются динамически исходя из области изменения переданных данных. В результате на координатной оси появляются и хранятся экземпляры классов `matplotlib.axis.XTick` и `matplotlib.axis.YTick`. Они родственны классу `matplotlib.axis.Tick`.

Ввод [68]:

```

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 200)
y = 0.01*(x + 9)*(x + 6)*(x - 6)*(x - 9) * x

fig = plt.figure(figsize=(8, 4))

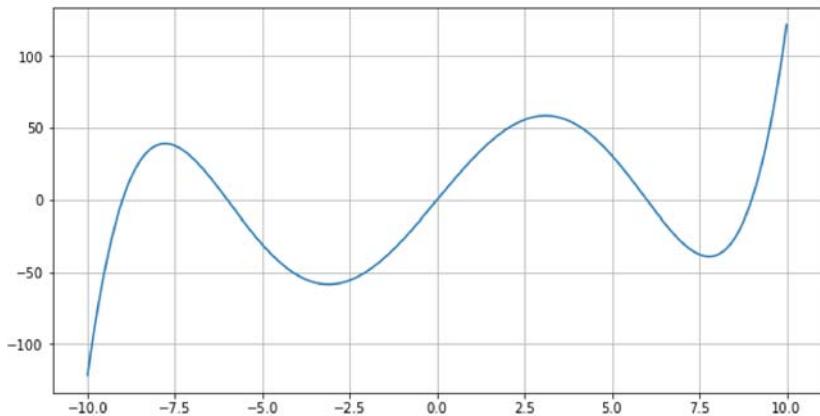
# Добавим на рисунок область рисования
ax = fig.add_axes([0, 0, 1, 1])
ax.plot(x, y)
ax.grid()
plt.show()

# экземпляр xaxis
xax = ax.xaxis

print ('-----')
print ('Тип изменения оси:', xax.get_scale())
print ('Область изменения отображения данных:', xax.get_view_interval())
print ('Область изменения отображения данных:', xax.get_data_interval())
print ('Линии сетки:', xax.get_gridlines())
print ('Подпись оси:', xax.get_label())
print ('Подписи делений:', xax.get_ticklabels())
print ('Линии делений:', xax.get_ticklines())
print ('Расположение делений:', xax.get_ticklocs())
print ('locator главных делений:', xax.get_major_locator())
print ('formatter главных делений:', xax.get_major_formatter())
print ('Список главных делений оси:', xax.get_major_ticks())
print ('locator вспомогательных делений:', xax.get_minor_locator())
print ('formatter вспомогательных делений:', xax.get_minor_formatter())
print ('Список вспомогательных делений оси:', xax.get_minor_ticks())
print ('-----')

# экземпляр yaxis
yax = ax.yaxis
print ('Область изменения отображения данных:', yax.get_view_interval())
print ('Область изменения отображения данных:', yax.get_data_interval())
print ('Расположение делений:', yax.get_ticklocs())
print ('Список вспомогательных делений оси:', xax.get_minor_ticks())
print ('-----')

```



```

-----  

Тип изменения оси: linear  

Вход [41]:  

Область изменения отображения данных: [-11. 11.]  

Область изменения отображения данных: [-10. 10.]  

Линии сетки: <a list of 11 Line2D gridline objects>  

Подпись оси: Text(0.5, 3.2000000000000003, '')  

Подпись, делений: [Text(-12.5, 0, '-12.5'), Text(-10.0, 0, '-10.0'), Text(-7.5, 0, '-7.5'), Text(-5.0, 0, '-5.0'), Text(-2.5, 0, '-2.5'), Text(0.0, 0, '0'), Text(0.6, 0, '0.6'), Text(2.5, 0, '2.5'), Text(5.0, 0, '5.0'), Text(7.5, 0, '7.5'), Text(10.0, 0, '10.0'), Text(12.5, 0, '12.5')]  

Линии делений: <a list of 22 Line2D ticklines objects>  

Расположение делений: [-12.5 -10. -7.5 -5. 0. 2.5 5. 7.5 10. 12.5]  

locator главных делений: <matplotlib.ticker.AutoLocator object at 0x000001F38DF7C400>  

formatter главных делений: <matplotlib.ticker.ScalarFormatter object at 0x000001F38DF7C910>  

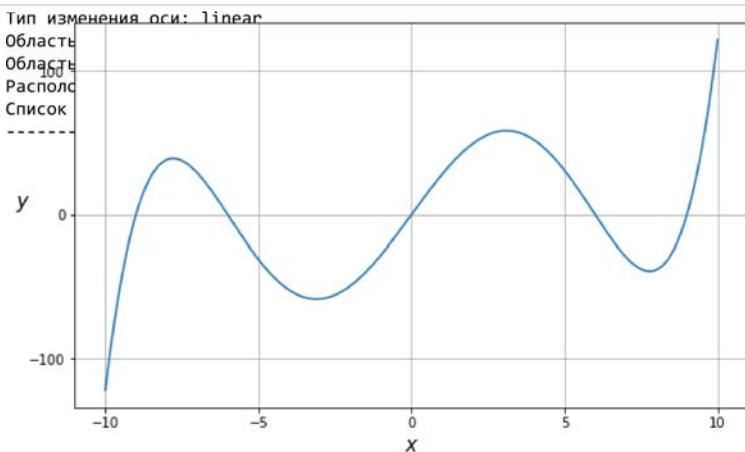
Список главных делений: [<matplotlib.axis.XTick object at 0x000001F38DF7CD90>, <matplotlib.axis.XTick object at 0x000001F38DF7CD60>, <matplotlib.axis.XTick object at 0x000001F38DFA0F40>, <matplotlib.axis.XTick object at 0x000001F38DFAF490>, <matplotlib.axis.XTick object at 0x000001F38DFA9A0>, <matplotlib.axis.XTick object at 0x000001F38DFAEB0>, <matplotlib.axis.XTick object at 0x000001F38DFB590>, <matplotlib.axis.XTick object at 0x000001F38DFB5400>, <matplotlib.axis.XTick object at 0x000001F38DFB5E20>, <matplotlib.axis.XTick object at 0x000001F38DFC370>]  

ax.set_xlabel("$x$", family="monospace", style="italic", weight="heavy", size=15)  

locator вспомогательных делений: <matplotlib.ticker.NullLocator object at 0x000001F38DF665B0>  

formatter вспомогательных делений: <matplotlib.ticker.NullFormatter object at 0x000001F38DF66B50>  

Список вспомогательных делений оси: []
-----
```



Деления ticks неотделимы от координатной оси на которой они находятся. Однако свойства самих делений (цвет, длина, толщина и др.), и их подписей (кегль, поворот, цвет шрифта, шрифт и др.), а также связанные с ними линии вспомогательной сетки grid , удобно хранить в отдельном хранилище-контейнере Ticks , а не в контейнере Axis .

Контейнер matplotlib.axis.Tick - это последний и самый низкоуровневый из Artists – контейнеров, самая маленькая "матрёшка". Он содержит экземпляры делений ticks , линий вспомогательной сетки grid lines и подписей labels для верхних (*upper ticks*) и нижних делений (*lower ticks*). К каждому из них есть прямой доступ как к одному из атрибуту экземпляра Tick . Также контейнер существуют логические переменные с помощью которых можно определить с какой стороны будут нанесены подписи и деления: для оси ординат справа/слева, а для оси абсцисс – сверху/снизу в прямоугольной системе координат.

Для работы непосредственно с экземпляром Tick , необходимо "спуститься" к нему с более высоких контейнеров-уровней.

Axes - Axis(YAxis) - методы "set-get" -> ax.yaxis.get\_major\_ticks()

Атрибут Tick	Описание
tickline	экземпляр Line2D
tick2line	экземпляр Line2D
gridline	экземпляр Line2D
label1	экземпляр Text
label2	экземпляр Text
gridOn	логическая переменная, разрешающая рисовать tickline
tick1On	логическая переменная, разрешающая рисовать первую tickline
tick2On	логическая переменная, разрешающая рисовать вторую tickline
label1On	логическая переменная, разрешающая рисовать tick label
label2On	логическая переменная, разрешающая рисовать tick label

Атрибуты с номером 1 – это стандартно отображаемые деления, которые располагаются слева и/или снизу. Атрибуты с номером 2 – соответственно справа и/или сверху.

Ввод [88]:

```

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 200)
y = 0.01*(x + 9)*(x + 6)*(x - 6)*(x - 9) * x

fig = plt.figure(figsize=(7, 4))

# Добавим на рисунок область рисования
ax = fig.add_axes([0, 0, 1, 1])
ax.plot(x, y, color='r', linewidth=2.5)
ax.patch.set_facecolor('pink')

ax.grid()
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')

# Ось абсцисс
for tick in ax.xaxis.get_major_ticks():
    tick.label1.set_color('red')
    tick.label1.set_fontsize(14)
    tick.tick1line.set_markeredgewidth(2)
    tick.tick1line.set_markersize(5)
    tick.gridline.set_color('green')
    tick.gridline.set_linewidth(0.5)

# Ось ординат
for tick in ax.yaxis.get_major_ticks()[::2]:
    tick.label1.set_color('green')
    tick.label1.set_fontsize(14)
    # деления на оси OY слева
    tick.tick1line.set_markeredgecolor('green')
    tick.tick1line.set_markeredgewidth(2)
    tick.tick1line.set_markersize(8)
    # линии сетки для оси OX
    tick.gridOn = True
    tick.gridline.set_color('yellow')
    tick.gridline.set_linewidth(1)

plt.show()

```



#### Методы Locator и Formatter

Если главные деления будут автоматически созданы при вызове графической команды (например `ax.plot()`), и с ними, соответственно, можно работать с помощью `ax.xaxis.get_major_ticks()` или `ax.yaxis.get_major_ticks()`, то вспомогательные деления необходимо задавать вручную. Удобнее всего это делать с помощью методов `Formatter` и `Locator`.

Методы `Locator` и `Formatter` относятся к модулю `matplotlib.ticker`. Этот модуль содержит классы, позволяющие наиболее полно определять форматирование и местоположение делений. Это самый низкоуровневый способ форматирования делений на координатных осях.

Контейнеры координатных осей `XAxis` и `YAxis` имеют специальные методы для работы с объектами типа `Formatter`: `.set_major/minor_locator()` и `.set_major/minor_formatter()`. Например:

- `xax.set_major_locator();`
- `xax.set_major_formatter();`
- `yax.set_minor_locator();`
- `yax.set_minor_formatter();`

В качестве входящих данных методам нужно передать какой-либо экземпляр класса из модуля `matplotlib.ticker`, например, `matplotlib.ticker.MultipleLocator`.

**Класс Locator** является базовым классом для всех производных классов-локаторов, отвечающих за расположение делений. Локаторы работают с автоматомасштабированием в пределах области изменения данных, и исходя из этого определяют положение делений на оси. Одним из наиболее удобных является полуавтоматический локатор `MultipleLocator`. В качестве входящих данных он принимает целое число, например 10, и самостоятельно подбирает пределы изменений на оси и располагает деления в местах, кратных заданному числу.

Другой пример - **субкласс AutoMinorLocator()**. Он позволяет автоматически определить положение вспомогательных делений. В качестве входящего параметра даётся целое число  $n$  - число промежутков, разделённых вспомогательными делениями, между двумя главными делениями. Это один из самых простых способов задать положение и значения вспомогательных делений. Форматирование подписей к таким делениям (их строковое представление), легче всего осуществить с помощью методов-форматеров.

**Класс Formatter** является базовым для всех производных классов-форматеров, отвечающих за форматирование делений. Форматеры в качестве входящих данных принимают строку формата, например `%d`, которая применяется к подписи каждого деления.

Т.о. для управления делениями осей – тиками необходим модуль `matplotlib.ticker` и методы для установки интервалов основных и вспомогательных делений.

```
# Подключаем модуль управления тиками:
import matplotlib.ticker as ticker

fig = plt.figure(figsize=(8, 4))

# Добавим на рисунок область рисования
ax = fig.add_axes([0, 0, 1, 1])

# Включаем видимость вспомогательных делений:
ax.minorticks_on()

# Устанавливаем интервал основных и
# вспомогательных делений:
ax.xaxis.set_major_locator(ticker.MultipleLocator(10))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(100))
ax.yaxis.set_minor_locator(ticker.MultipleLocator(10))
```

Рассмотрим на примере, как управлять параметрами основных и вспомогательных делений.

Ввод [24]:

```

import numpy as np
import matplotlib.pyplot as plt
# Подключаем модуль управления тиками:
import matplotlib.ticker as ticker

x = np.linspace(-10, 10, 200)
y = 0.01*(x + 9)*(x + 6)*(x - 6)*(x - 9) * x

fig, ax = plt.subplots(figsize=(12, 8))

ax.plot(x, y, color = 'r', linewidth = 3)

# Устанавливаем интервал основных и
# Вспомогательных делений:
ax.xaxis.set_major_locator(ticker.MultipleLocator(5))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(100))
ax.yaxis.set_minor_locator(ticker.MultipleLocator(20))

# Устанавливаем формат основных и
# Вспомогательных делений:
ax.xaxis.set_major_formatter(ticker.FormatStrFormatter('%.1f'))
ax.xaxis.set_minor_formatter(ticker.FormatStrFormatter('%d'))
ax.yaxis.set_major_formatter(ticker.FormatStrFormatter('%.1f'))
ax.yaxis.set_minor_formatter(ticker.FormatStrFormatter('%d'))

# Включаем видимость вспомогательных делений:
# ax.minorticks_on()

# Настраиваем вид основных тиков:
ax.tick_params(axis = 'both',
               which = 'major',
               direction = 'inout',
               length = 20,
               width = 4,
               color = 'm',
               bottom = True,
               top = True,
               left = True,
               right = True,
               pad = 10,
               labelsize = 15,
               labelcolor = 'r',
               labelbottom = True,
               labeltop = True,
               labelleft = True,
               labelright = True,
               labelrotation = 45) # Параметры к обеим осям
# Параметры к основным делениям
# Рисуем деления внутри и снаружи графика
# Длина делений
# Толщина делений
# Цвет делений
# Рисуем метки снизу
# сверху
# слева
# справа
# Рассстояние между чертой и ее подписью
# Размер подписи
# Цвет подписи
# Рисуем подписи снизу
# сверху
# слева
# справа
# Поворот подписей

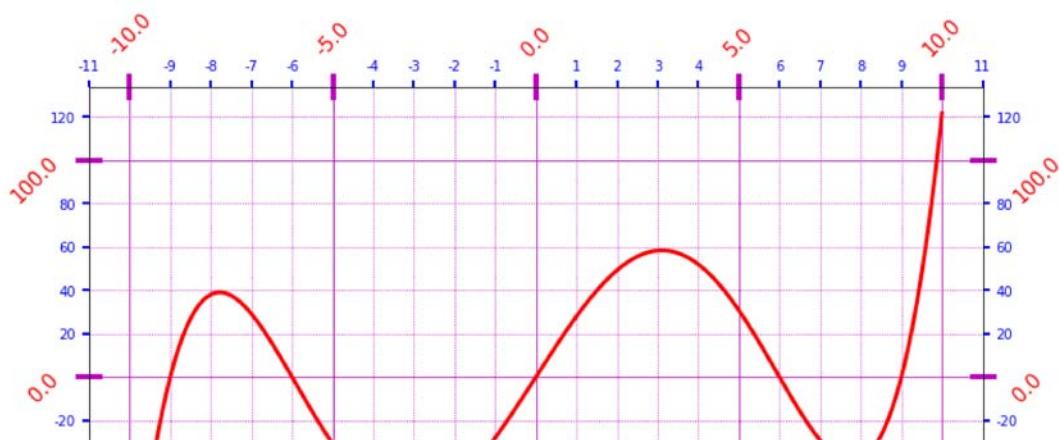
# Настраиваем вид вспомогательных тиков:
ax.tick_params(axis = 'both',
               which = 'minor',
               direction = 'out',
               length = 5,
               width = 2,
               color = 'b',
               bottom = True,
               top = True,
               left = True,
               right = True,
               pad = 5,
               labelsize = 10,
               labelcolor = 'b',
               labelbottom = True,
               labeltop = True,
               labelleft = True,
               labelright = True) # Параметры к обеим осям
# Параметры к вспомогательным делениям
# Рисуем деления снаружи графика
# Длина делений
# Толщина делений
# Цвет делений
# Рисуем метки снизу
# сверху
# слева
# справа
# Рассстояние между чертой и ее подписью
# Размер подписи
# Цвет подписи
# Рисуем подписи снизу
# сверху
# слева
# справа

# Добавляем линии основной сетки:
ax.grid(which='major',
        color = 'm')

# Теперь можем отдельно задавать внешний вид
# Вспомогательной сетки:
ax.grid(which='minor',
        color = 'm',
        linestyle = ':')

plt.show()

```





Ввод [27]:

```

import numpy as np
import matplotlib.pyplot as plt
# Подключаем модуль управления тиками:
import matplotlib.ticker as ticker

x = np.linspace(-10, 10, 200)
y = 0.01*(x + 9)*(x + 6)*(x - 6)*(x - 9) * x

fig, ax = plt.subplots(figsize=(12, 8))

ax.plot(x, y, color = 'r', linewidth = 3)

# Переместим оси в начало координат
ax = plt.gca() # захватывает текущую ось и возвращает её
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data', 0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data', 0))

# Устанавливаем интервал основных и
# Вспомогательных делений:
ax.xaxis.set_major_locator(ticker.MultipleLocator(5))
ax.xaxis.set_minor_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(100))
ax.yaxis.set_minor_locator(ticker.MultipleLocator(20))

# Устанавливаем формат основных и
# Вспомогательных делений:
ax.xaxis.set_major_formatter(ticker.FormatStrFormatter('.1f'))
ax.xaxis.set_minor_formatter(ticker.FormatStrFormatter('%d'))
ax.yaxis.set_major_formatter(ticker.FormatStrFormatter('.1f'))
ax.yaxis.set_minor_formatter(ticker.FormatStrFormatter('%d'))

# Включаем видимость вспомогательных делений:
# ax.minorticks_on()

# Настраиваем вид основных тиков:
ax.tick_params(axis = 'both',
               which = 'major',
               direction = 'inout',
               length = 20,
               width = 4,
               color = 'm',
               bottom = True, # Рисуем метки снизу
               top = False, # сверху
               left = True, # слева
               right = False, # справа
               pad = 10, # Расстояние между чертой и ее подписью
               labelsize = 15, # Размер подписи
               labelcolor = 'r', # Цвет подписи
               labelbottom = True, # Рисуем подписи снизу
               labeltop = False, # сверху
               labelleft = True, # слева
               labelright = False) # справа

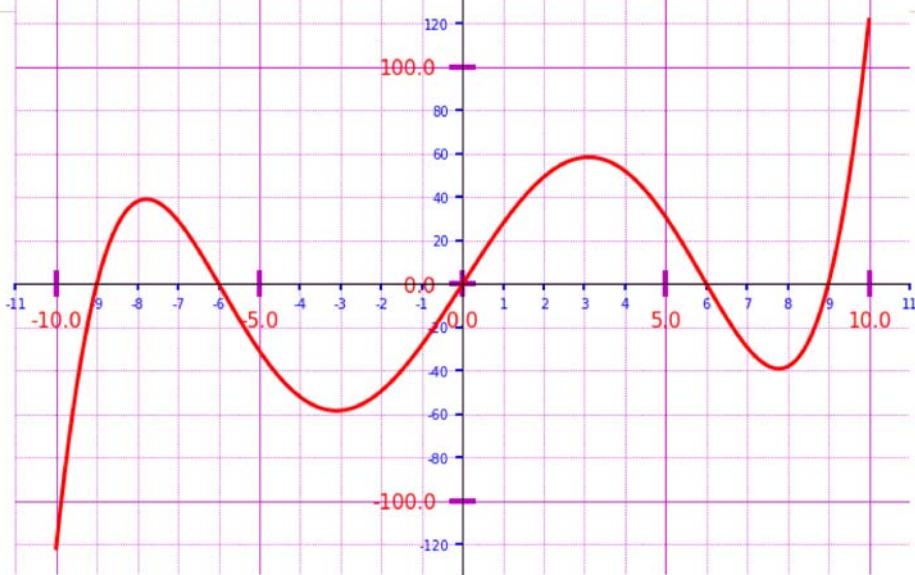
# Настраиваем вид вспомогательных тиков:
ax.tick_params(axis = 'both',
               which = 'minor',
               direction = 'out',
               length = 5,
               width = 2,
               color = 'b',
               bottom = True, # Рисуем метки снизу
               top = False, # сверху
               left = True, # слева
               right = False, # справа
               pad = 5, # Расстояние между чертой и ее подписью
               labelsize = 10, # Размер подписи
               labelcolor = 'b', # Цвет подписи
               labelbottom = True, # Рисуем подписи снизу
               labeltop = False, # сверху
               labelleft = True, # слева
               labelright = False) # справа

# Добавляем линии основной сетки:
ax.grid(which='major',
        color = 'm')

# Теперь можем отдельно задавать внешний вид
# Вспомогательной сетки:
ax.grid(which='minor',
        color = 'm',
        linestyle = ':')

```

```
plt.show()
```



## 6.2 Координатные оси Axis

Данные, нанесённые тем или иным графическим способом на рисунок `Figure` и область рисования `Axes`, не представляют особого интереса для научного анализа, пока они не привязаны к какой-либо системе координат.

При создании экземпляра `axes` на вновь созданной области рисования автоматически задаётся прямоугольная система координат, если не указаны атрибуты `polar` или `projection`. Система координат определяет вид координатных осей. По умолчанию задаётся прямоугольная система координат: ось абсцисс (ось "OХ") и ось ординат (ось "OY"). В полярной системе координат координатными осями являются радиус и угол наклона, что выражается в виде своеобразного тригонометрического круга.

Для каждой оси можно задать:

- метку (подпись);
- основные (*major*) и дополнительные (*minor*) тики, их подписи, размер, толщину;
- диапазоны значений.

Для хранения и форматирования каждой координатной оси существует контейнер `Axis`.

**Axis** (не путать с областями рисования `Axes`) – контейнер в `matplotlib`, который привязан к области рисования `Axes` и на котором располагаются деления осей (*ticks*), подписи делений (*tick labels*) и подписи осей (*axis labels*). Это третья "матрёшка" после `Figure` и `Axes`. Координатные оси являются экземплярами класса `matplotlib.axis.Axis`.

Оси задают границы области графика. Для объекта `Axes` можно настроить расположенные (сверху, снизу, слева и справа), каждую ось можно отформатировать, указав цвет и ширину, а можно сделать невидимым (если цвет оси не задан).

Для изменения позиции конкретной линии оси координат используется метод `spines`, который перемещает линию оси координат в указанную позицию.

Ввод [19]:

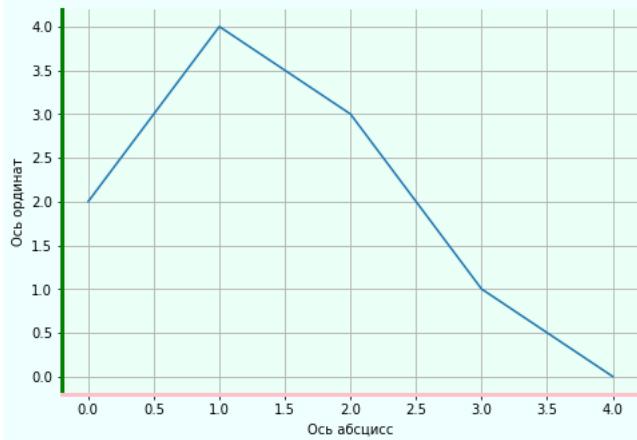
```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])

ax.spines['bottom'].set_color('pink')
ax.spines['bottom'].set_linewidth(3)
ax.spines['left'].set_color('green')
ax.spines['left'].set_linewidth(3)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.set_xlabel("Ось абсцисс")
ax.set_ylabel("Ось ординат")
ax.set_facecolor('#eaffff')
ax.grid()
fig.set_facecolor('#ffffff')

ax.plot([2, 4, 3, 1, 0])

plt.show()
```



Можно отцентрировать расположение координатных осей.

Ввод [34]:

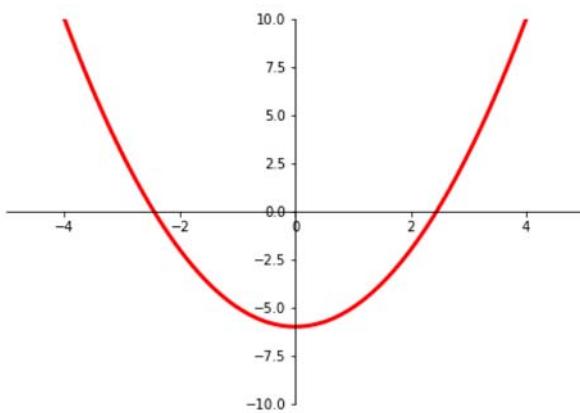
```
import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 100)
y = x**2 - 6

fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])

ax.set_xlim(-5, 5)
ax.set_ylim(-10, 10)
ax.spines["left"].set_position ("center")
ax.spines["bottom"].set_position("center")
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.plot(x, y, color='red', linewidth=3, label='Парабола')

plt.show()
```



Любая область рисования Axes содержит два особых Artist -контейнера: XAxis и YAxis . Они отвечают за отрисовку делений ( ticks ) и подписей ( labels ) координатных осей, которые хранятся как экземпляры в переменных xaxis и yaxis . Чтобы обратиться к экземпляру axis , отвечающему за ось ординат, нужно обратиться к контейнеру yaxis соответствующей области рисования ax . Причём запись уах=ax.get\_yaxis() идентична записи уах=ax.yaxis .

Чтобы переместить оси в начало координат, можно воспользоваться командами:

```
ax = plt.gca() # захватываем текущую ось и возвращаем её

ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data', 0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data', 0))
```

Ввод [6]:

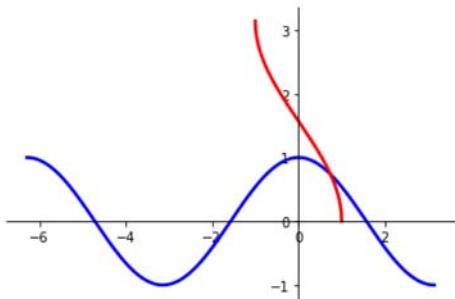
```
import matplotlib.pyplot as plt
import numpy as np

x1 = np.linspace(-2*np.pi, np.pi, 200, endpoint=True)
x2 = np.linspace(-1, 1, 200, endpoint=True)
cos, arccos = np.cos(x1), np.arccos(x2)

ax = plt.gca() # захватываем текущую ось и возвращаем её
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data', 0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data', 0))

ax.plot(x1, cos, 'b-', lw = 2.5, label = "Косинус")
ax.plot(x2, arccos, 'r-', lw = 2.5, label = "Арккосинус")

plt.show()
```



**Пояснение.** Для задания интервала по оси  $X$  мы воспользовались функцией `numpy.linspace()`, которая создает одномерный массив со значениями из заданного интервала с указанным количеством отсчетов в нет (в нашем случае 200). По умолчанию правый конец интервала также включается в результат (если правый конец не нужно включать в результат, то можно передать параметр `endpoint`, равный `False`).

Ввод [48]:

```

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 200)
y = 0.01*(x + 9)*(x + 6)*(x - 6)*(x - 9) * x

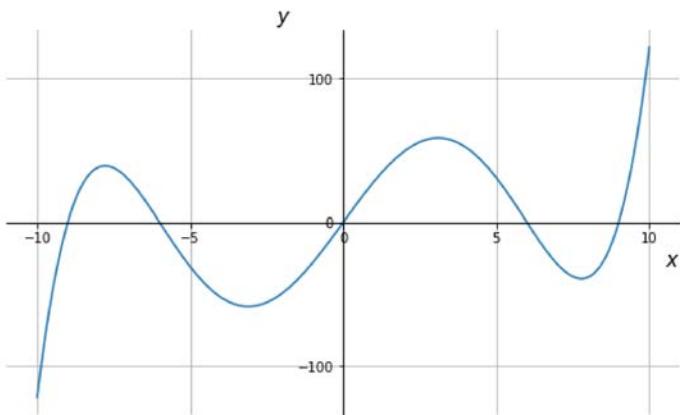
fig = plt.figure(figsize=(7, 4))

# Добавим на рисунок область рисования
ax = fig.add_axes([0, 0, 1, 1], xticks=range(-10, 11, 5), yticks=[-100, 0, 100])

ax = plt.gca() # захватывает текущую ось и возвращает её
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data', 0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data', 0))
ax.plot(x, y)

ax.set_xlabel('$x$', loc='right', family="monospace",
             style="italic", size=15)
ax.set_ylabel('$y$', loc='top', family="monospace",
             style="italic", size=15, rotation='horizontal')
ax.grid()
plt.show()

```



### 6.3 Работа с сеткой

Функция `grid()` устанавливает или отключает видимость сетки внутри фигуры. Также можно отобразить основные / второстепенные (или оба) галочки сетки. Так как сетка состоит из линий, то метод `grid()` поддерживает все параметры, используемые для настройки отображения линий: `color`, `linestyle` и `linewidth`. Метод позволяет нанести на рисунок линии основной и вспомогательной сетки. Чтобы работать с дополнительной сеткой необходимо сначала включить вспомогательные отметки на осях (которые по умолчанию отключены и к которым привязаны линии дополнительной сетки) с помощью функции `pyplot.minorticks_on`.

Параметры метода `grid()`:

```
matplotlib.pyplot.grid(b, which, axis, color, linestyle, linewidth, **kwargs)
```

где:

- `b` – логическое значение (`True/False`), определяющее отображать линии сетки или нет;
- `which` – параметр, определяющий параметры какой линии (основной, вспомогательной или всех) настраиваются;
- `axis` – параметр, определяющий какие именно линии сетки необходимо отображать: горизонтальные (`axis = 'y'`), вертикальные (`axis = 'x'`) или все вместе;
- `color` – определение цвета линий сетки;
- `linestyle` – определение стиля отображения линии (сплошная, пунктирная, штриховая и т.п.)
- `linewidth` – определение ширины линий сетки.

Варианты отображения сетки:

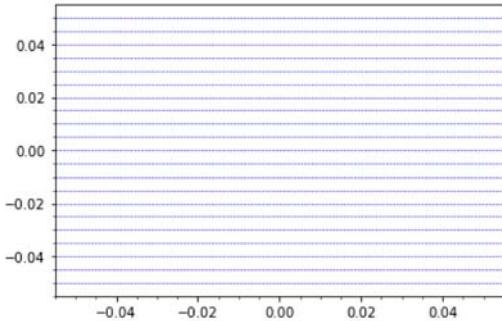
- `plt.grid(True)` – отобразить сетку с параметрами, заданными по умолчанию;
- `plt.grid(True, axis='y', color='b', ls = '-.', linewidth = 2)` – отобразить горизонтальную сетку синего цвета, с пунктирной линией и толщиной линий, равной 2 пт.

Ввод [32]:

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

%matplotlib inline

plt.figure(figsize=(6, 4))
# Включаем дополнительные отметки на осях
plt.minorticks_on()
plt.plot()
plt.grid(True, which='both', axis='y', color='b', ls = '--', linewidth = 0.5)
plt.show()
```



Линии сетки (подобно меткам делений) бывают как **основные** так и **вспомогательные**. Контролировать то как они отображаются отдельно позволяет параметр `which`, который может принимать одно из трех значений:

- `major` – применение параметров внешнего вида к основным линиям (установлен по умолчанию);
- `minor` – к вспомогательным линиям;
- `both` – к обеим линиям.

Ввод [35]:

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

fig = plt.figure(figsize=(8, 4))
ax = fig.add_axes([0,0,1,1])

x = np.linspace(-np.pi, np.pi, 100)
y = np.sinc(x)

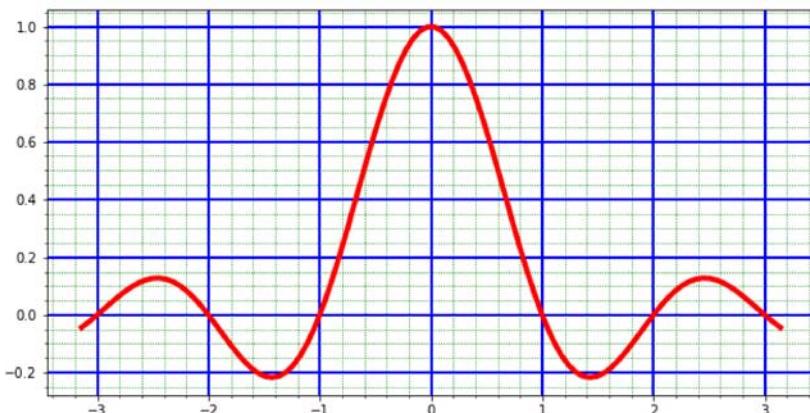
ax.plot(x, y, color = 'r', linewidth = 4)

# Прежде чем рисовать вспомогательные линии
# необходимо включить второстепенные деления осей
ax.minorticks_on()

# Определяем внешний вид линий основной сетки
ax.grid(which='major', color = 'b', linewidth = 2)

# Определяем внешний вид линий вспомогательной сетки
ax.grid(which='minor', color = 'g', linestyle = ':')

plt.show()
```



## 6.4 Установка пределов отображения осей

Matplotlib автоматически достигает минимального и максимального значений переменных, которые будут отображаться вдоль осей `x`, `y` (и оси `z` в случае трехмерного графика). Однако можно установить ограничения явно с помощью функций `set_xlim()` и `set_ylim()`.

В зависимости от подхода к построению графика, для установки пределов отображения осей можно использовать команды:

```
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
...
ax.set_xlim(-3, 3)
ax.set_ylim(-1, 1)
```

или

```
plt.figure()
...
plt.xlim(-3, 3)
plt.ylim(-1, 1)
```

На следующем графике показаны автомасштабированные пределы осей x и y :

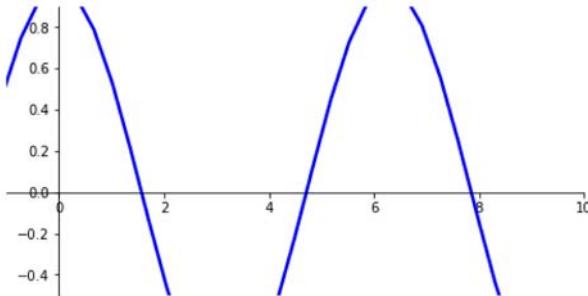
Ввод [9]:

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

x = np.linspace(-2*np.pi, 20*np.pi, 200, endpoint=True)
y = np.cos(x)

fig = plt.figure(figsize=(6, 3))
ax = fig.add_axes([0, 0, 1, 1])

ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data', 0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data', 0))
ax.set_xlim(-1, 10)
ax.set_ylim(-0.5, 0.9)
ax.plot (x, y, 'b-', lw = 2.5, label = "Косинус")
plt.show()
```



Настройку параметров графика можно выполнить более коротким способом:

Ввод [10]:

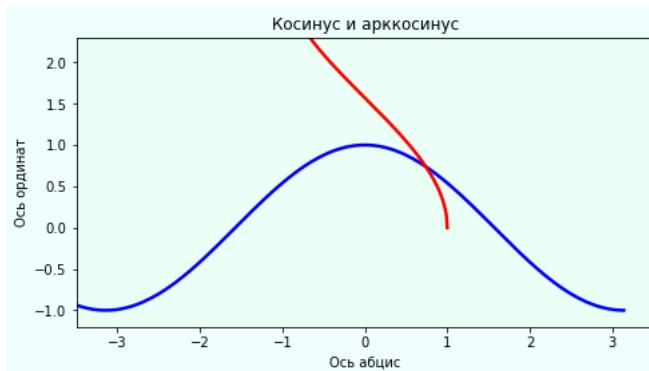
```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(6, 3))
ax = fig.add_axes([0, 0, 1, 1])

fig.set(facecolor="#eaffff")
ax.set(facecolor = '#eafff5',
      xlim = [-3.5, 3.5],
      ylim = [-1.2, 2.3],
      title = 'Косинус и арккосинус',
      xlabel = 'Ось абсис',
      ylabel = 'Ось ординат')

x1 = np.linspace(-2*np.pi, np.pi, 200, endpoint=True)
x2 = np.linspace(-1, 1, 200, endpoint=True)
cos, arccos = np.cos(x1), np.arccos(x2)
ax.plot(x1, cos, 'b-', lw = 2.5, label = "Косинус")
ax.plot(x2, arccos, 'r-', lw = 2.5, label = "Арккосинус")

plt.show()
```



## § 7 Текстовые надписи на графике

Matplotlib позволяет нанести на график подсказки. В части **текстового наполнения** при построении графика выделяют следующие составляющие:

- заголовок поля (`title`);
- заголовок фигуры (`suptitle`);
- подписи осей (`xlabel`, `ylabel`);
- текстовый блок на поле графика (`text`), либо на фигуре (`figtext`);
- аннотация (`annotate`) – текст и указатель.

### 7.1 Параметры текста

Большинство методов, позволяющие дополнить элемент текстом, дополнительно принимают в качестве аргументов параметры конструктора класса `matplotlib.text.Text`. В таблице вот некоторые из них:

Таблица. Параметры текста

Параметр	Назначение параметра	Возможные значения
<code>alpha</code>	Уровень прозрачности надписи	Параметр задается числом <code>float</code> в диапазоне от 0 до 1. 0 – полная прозрачность, 1 – полная непрозрачность
<code>color</code>	Цвет текста подсказки	Один из доступных способов задания цвета (аналогично цвету линии)
<code>fontfamily</code> или <code>family</code>	Шрифт текста	Задается в виде строки <code>str</code> из следующего набора: <code>serif</code> , <code>sans-serif</code> , <code>cursive</code> , <code>fantasy</code> , <code>monospace</code> . Можно использовать свой шрифт
<code>fontsize</code> или <code>size</code>	Размер шрифта	Число <code>int</code> либо значение из списка: <code>xx-small</code> , <code>x-small</code> , <code>small</code> , <code>medium</code> , <code>large</code> , <code>x-large</code> , <code>xxlarge</code>
<code>fontstyle</code> или <code>style</code>	Стиль шрифта	Значение из списка: <code>normal</code> , <code>italic</code> , <code>oblique</code>
<code>fontvariant</code> или <code>variant</code>	Начертание шрифта	Задается из следующего набора: <code>normal</code> , <code>small-caps</code>
<code>fontweight</code> или <code>weight</code>	Насыщенность шрифта	Число в диапазоне от 0 до 1000 либо значение из списка: <code>ultralight</code> , <code>light</code> , <code>normal</code> , <code>regular</code> , <code>book</code> , <code>medium</code> , <code>roman</code> , <code>semibold</code> , <code>dembold</code> и пр.

Для группового задания свойств можно использовать параметр `fontproperties` или `font_properties`, которому в качестве значения можно передать объект класса `font_manager.FontProperties`. Данный объект создается следующим образом (перед использованием необходимо предварительно импортировать `FontProperties`):

```
from matplotlib.font_manager import FontProperties

plt.title("Title", fontproperties=FontProperties(family="monospace", style="italic", weight="heavy", size=15))
```

Параметры, отвечающих за отображение шрифта:

- `family` – имя шрифта;
- `style` – стиль шрифта;
- `variant` – начертание;

- stretch – ширина шрифта;
- weight – насыщенность шрифта;
- size – размер шрифта.

## 7.2 Ориентация, вращение и позиция надписи

Для надписи можно задать выравнивание, позицию, вращение и z -порядок:

- horizontalalignment или ha – горизонтальное выравнивание, задается из следующего набора { center , right , left }, строковая величина str ;
- verticalalignment или va – вертикальное выравнивание, задается из следующего набора { center , top , bottom , baseline , center\_baseline }, строковая величина str ;
- position – позиция надписи, определяется двумя координатами x и y , которые передаются в параметр position в виде кортежа из двух вещественных элементов ( float , float );
- rotation – вращение. Ориентацию надписи можно задать в виде текста { vertical , horizontal } ( str ), либо численно – значение в градусах ( float );
- rotation\_mode – режим вращения. Данный параметр (типа str ) определяет очередность вращения и выравнивания. Если он равен default , то вначале производится вращение, а потом выравнивание. Если равен anchor , то наоборот.
- zorder – порядок расположения. Значение параметра ( float ) определяет очередьность вывода элементов. Элемент с минимальным значением zorder выводится первым.

## 7.3 Заголовок графика

Для задания заголовка графика используется функция title() .

Параметры функции:

- label – строка ( str ), содержащая текст заголовка;
- loc – выравнивание заголовка: значение из набора: { center , left , right };
- y – расположение вертикальных осей для заголовка (1.0 – вверху), значение типа float . Если None (по умолчанию), y определяется автоматически;
- pad – смещение заголовка от верха осей в пунктах float .

```
plt.title('Тригонометрические функции', alpha=0.75, color="r", loc='left', pad=20)
```

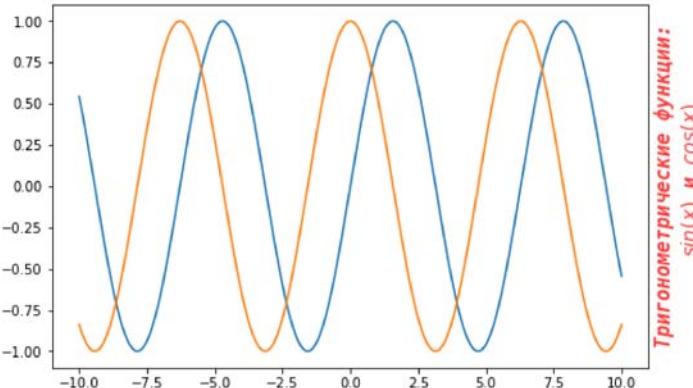
Ввод [11]:

```
import numpy as np
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt

x = np.arange(-10, 10.01, 0.01)

plt.figure(figsize=(8, 5))
plt.title('Тригонометрические функции: \n$\sin(x)$ и $\cos(x)$',
          alpha=0.75, color="r",
          fontproperties=FontProperties(family="monospace", style="italic",
                                         weight="heavy", size=15),
          rotation='vertical', y=0.05, position=(1.05, 0)
         )
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))

plt.show()
```



## 7.4 Подписи осей

Для задания подписи оси x используется функция xlabel() , оси y – ylabel() .

Основными параметрами данных функций xlabel() и ylabel() являются:

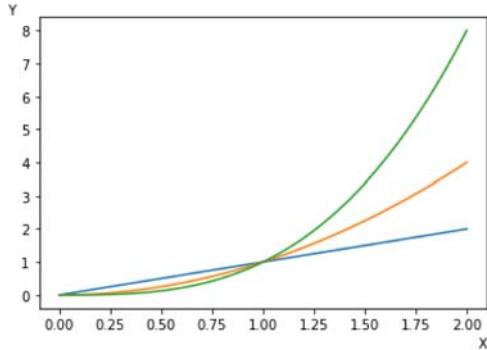
- xlabel (или ylabel) – текст подписи (тип str );
- labelpad – расстояние между областью графика, включающим оси, и текстом подписи. Численное значение либо None ; значение по умолчанию: None ;
- loc – выравнивание подписи: для xlabel – значение из набора: { center , left , right }, для ylabel – значение из набора { bottom , center , top }.

Ввод [7]:

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

x = np.linspace(0, 2, 100)

plt.figure()
plt.plot(x, x, x, x**2, x, x**3)
plt.xlabel('X', loc='right')
plt.ylabel('Y', loc='top', rotation='horizontal')
plt.show()
```



## 7.5 Текстовый блок

Функция `text()` позволяет нанести подпись в любом месте изображения. Для этого указываются сам текст надписи и координаты (по умолчанию координаты задаются в координатах активных осей), где должен располагаться текст, дополнительно можно указать параметры для настройки шрифта.

Основные параметры функции `text()`:

- `x` – значение float координаты x надписи;
- `y` – значение float координаты y надписи;
- `s` – строка str с текстом надписи.

В простейшем варианте использование `text()` будет выглядеть так:

```
plt.text(0, 7, 'HELLO!', fontsize=15)
```

Текстовые поля в `matplotlib` могут содержать разметку LaTeX, заключенную в знаки \$. Буква `r` перед кавычками говорит Python, что символ "" следует оставить как есть и не интерпретировать как начало спецсимвола (например, перевода строки - \n ).

Ввод [13]:

```

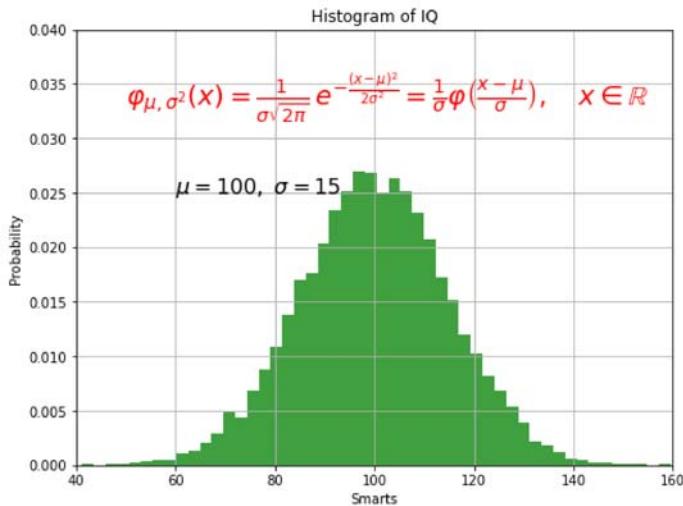
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

plt.figure(figsize=(8, 6))
n, bins, patches = plt.hist(x, 50, density=True, facecolor='g', alpha=0.75)

plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100, \sigma=15$')
plt.text(50, .033, r'$\varphi_{\mu, \sigma^2}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} = \frac{1}{\sigma} \varphi(\frac{x-\mu}{\sigma})$, $x \in \mathbb{R}$')
plt.axis([40, 160, 0, 0.04])
plt.grid(True)
plt.show()

```



## 7.6 Отображение легенды

Для отображения легенды на графике используется функция `legend()`. Возможны следующие варианты её вызова:

- `legend();`
- `legend(labels);`
- `legend(handles, labels);`

**В первом варианте** в качестве меток для легенды будут использоваться метки, указанные в функциях построения графиков (параметр `label`). Это наиболее удобный вариант для использования.

**Второй вариант** позволяет самостоятельно указать текстовую метку для отображаемых данных.

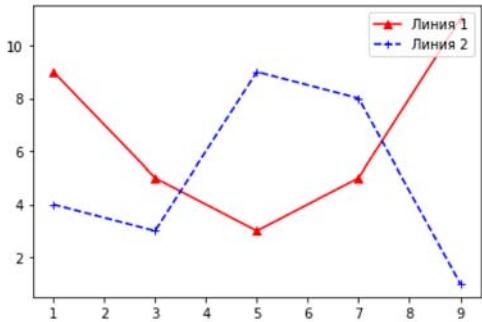
**В третьем варианте** можно вручную указать соответствие линий и текстовых меток

Ввод [14]:

```
# Первый вариант вызова функции Legend()
import matplotlib.pyplot as plt
%matplotlib inline

x = [1, 3, 5, 7, 9]
y1 = [9, 5, 3, 5, 11]
y2 = [4, 3, 9, 8, 1]

plt.plot(x, y1, '^-r', label='Линия 1')
plt.plot(x, y2, '+--b', label='Линия 2')
plt.legend(loc=1)
plt.show()
```

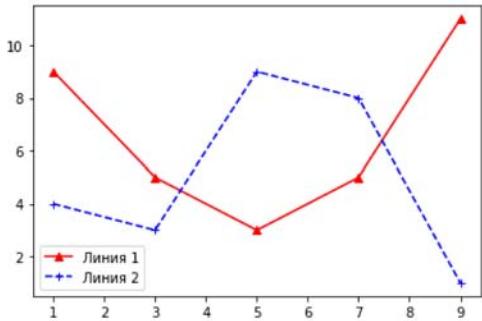


Ввод [15]:

```
# Второй вариант вызова функции Legend()
import matplotlib.pyplot as plt
%matplotlib inline

x = [1, 3, 5, 7, 9]
y1 = [9, 5, 3, 5, 11]
y2 = [4, 3, 9, 8, 1]

plt.plot(x, y1, '^-r')
plt.plot(x, y2, '+--b')
plt.legend(['Линия 1', 'Линия 2'], loc=3)
plt.show()
```

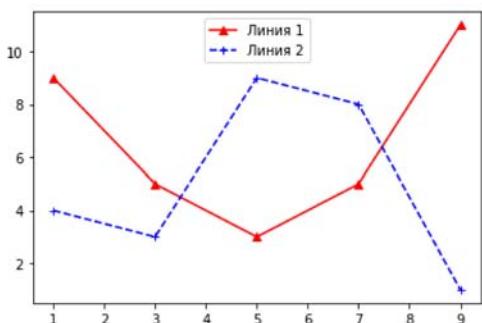


Ввод [16]:

```
# Третий вариант вызова функции Legend()
import matplotlib.pyplot as plt
%matplotlib inline

x = [1, 3, 5, 7, 9]
y1 = [9, 5, 3, 5, 11]
y2 = [4, 3, 9, 8, 1]

line_1, = plt.plot(x, y1, '^-r')
line_2, = plt.plot(x, y2, '+--b')
plt.legend((line_1, line_2), ['Линия 1', 'Линия 2'], loc=9)
plt.show()
```



**Место расположения легенды** определяется параметром `loc`, который может принимать одно из значений, указанных в таблице.

Таблица. Параметры расположения легенды на графике

Строковое описание	Код	Описание
best	0	Лучшее положение
upper right	1	Вверху справа
upper left	2	Вверху слева
lower left	3	Внизу слева
lower right	4	Внизу справа
right	5	Справа
center left	6	В центре слева
center right	7	В центре справа
lower center	8	Снизу по центру
upper center	9	Сверху по центру
center	10	По центру

Больше возможностей о настройке расположения легенды предоставляет параметр `bbox_to_anchor` функции `legend()`. С его помощью можно отобразить легенду за пределами графика, что может быть необходимо в случае, когда график перегружен, легенда необходима, но ее отображение закроет необходимый фрагмент изображения.

Параметру `bbox_to_anchor` передается кортеж, состоящий из четырёх или двух элементов:

```
bbox_to_anchor = (x, y, width, height)
bbox_to_anchor = (x, y)
```

- `x`, `y` – это координаты расположения легенды;
- `width` – ширина;
- `height` – высота.

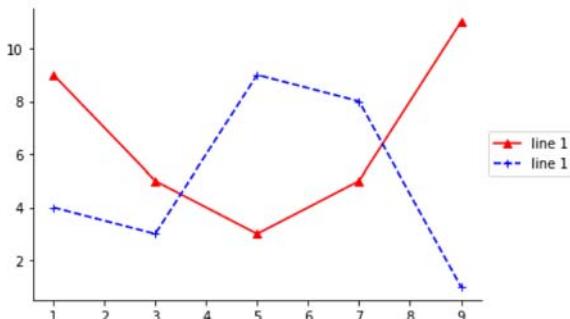
Ввод [17]:

```
# Пример использования параметра bbox_to_anchor:
import matplotlib.pyplot as plt
%matplotlib inline

x = [1, 3, 5, 7, 9]
y1 = [9, 5, 3, 5, 11]
y2 = [4, 3, 9, 8, 1]

plt.plot(x, y1, '^--r', label='line 1')
plt.plot(x, y2, '+--b', label='line 1')
plt.legend(bbox_to_anchor=(1, 0.6))

ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.show()
```



## 7.7 Аннотации

Функция `annotate()` позволяет нанести подпись с необязательной стрелкой, указывающей на конкретное место на графике.

Основными аргументами функции `annotate()` являются:

- `text` – текст аннотации (строка `str`);
- `xy` – координаты места, на которое будет указывать стрелка - итерируемый объект из 2-х элементов (`float`, `float`);
- `xytext` – координаты расположения текстовой надписи - итерируемый объект из 2-х элементов (`float`, `float`);
- `xycoords` – система координат, в которой определяется расположение указателя `str`;
- `textcoords` – система координат, в которой определяется расположение текстового блока (`str`);
- `arrowprops` – параметры конфигурирования стрелки-указателя на аннотацию. Параметр принимает в качестве значения словарь, ключами которого являются параметры конструктора класса `FancyArrowPatch`, такие, как например, `arrowstyle` (отвечает за стиль стрелки) и `connectionstyle` (отвечает за стиль соединительной линии).

Ввод [36]:

```
# Пример кода, демонстрирующий работу с аннотациями
import matplotlib.pyplot as plt
import numpy as np

%matplotlib notebook

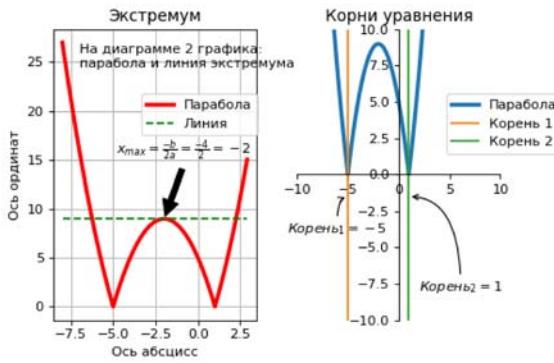
fig, (ax1, ax2) = plt.subplots(ncols=2) # 2 диаграммы по горизонтали
fig.canvas.set_window_title("Графики функций")

# 1-й график
x = np.arange(-8, 3, 0.1)
y1 = abs((x + 5) * (x - 1))
y2 = [9] * len(x)
ax1.plot(x, y1, 'r', linewidth=3, label="Парабола")
ax1.plot(x, y2, '--g', label="Линия")
ax1.set_title("Экстремум")
ax1.set_xlabel("Ось абсцисс")
ax1.set_ylabel("Ось ординат")
ax1.grid(True)
ax1.annotate(r"$x_{\max} = \frac{-b}{2a} = \frac{-4}{2} = -2$",
            xy=(-2, 9), xytext=(-4.8, 15.5),
            arrowprops=dict(facecolor="black", shrink=0.05))
ax1.text(-7, 24.5, "На диаграмме 2 графика: парабола и линия экстремума")
ax1.legend(bbox_to_anchor=(0.4, 0.8))

# 2-й график
y2 = list(range(-10, 11))
x2 = [-5] * len(y2)
x3 = [1] * len(y2)
ax2.set_title("Корни уравнения")
ax2.plot(x, y1, linewidth=3, label="Парабола")
ax2.plot(x2, y2, label="Корень 1")
ax2.plot(x3, y2, label="Корень 2")
ax2.annotate("$Корень_1 = -5$", xy=(-5, -1.5), xytext=(-11, -4),
            arrowprops=dict(facecolor="black", connectionstyle="angle3",
                            arrowstyle="->"))
ax2.annotate("$Корень_2 = 1$", xy=(1, -1.5), xytext=(2, -8),
            arrowprops=dict(facecolor="black", connectionstyle="angle3",
                            arrowstyle="->"))
)
ax2.set_xlim(-10, 10)
ax2.set_ylim(-10, 10)
ax2.spines["left"].set_position("center")
ax2.spines["bottom"].set_position("center")
ax2.spines["top"].set_visible(False)
ax2.spines["right"].set_visible(False)
ax2.legend(bbox_to_anchor=(0.7, 0.8))

plt.show();
```

&lt;IPython.core.display.Javascript object&gt;



## § 8 Размещение графиков отдельно друг от друга

Существуют три основных подхода к размещению графиков на разных областях:

- использование функции `subplot()` для указания места размещения области с графиком;
- использование функции `subplots()` для предварительного задания сетки, в которую будут укладываться области;
- использование `GridSpec`, для более гибкого задания геометрии размещения полей с графиками на сетке.

Работа с `matplotlib` основана на использовании графических окон и осей (оси позволяют задать некоторую графическую область). Все построения применяются к текущим осям. Это позволяет изображать несколько графиков в одном графическом окне. **По умолчанию создаётся одно графическое окно `figure(1)` и одна графическая область `subplot(111)` в этом окне.**

### 8.1 Работа с функцией `subplot()`

Команда `subplot` позволяет разбить графическое окно на несколько областей. Она имеет три параметра: `nr`, `nc`, `np`:

- параметры `nr` и `nc` определяют количество строк и столбцов на которые разбивается графическая область;
- параметр `pr` определяет номер текущей области ( `pr` принимает значения от 1 до `nr*nc`).

Если `nr*nc < 10`, то передавать параметры `nr`, `nc`, `pr` можно без использования запятой.

**Например**, допустимы формы `subplot(2, 1, 2)` и `subplot(212)`. В обоих командах цифры сообщают о том, что надо подготовить разметку с двумя строками и одним столбцом, элемент вывести в первую позицию второй строки.

Ввод [42]:

```
# Выход трёх графиков с помощью subplot()
# Используем подход к созданию графиков, ориентированный на его структуру
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

x = np.arange(-5, 5, 0.5)

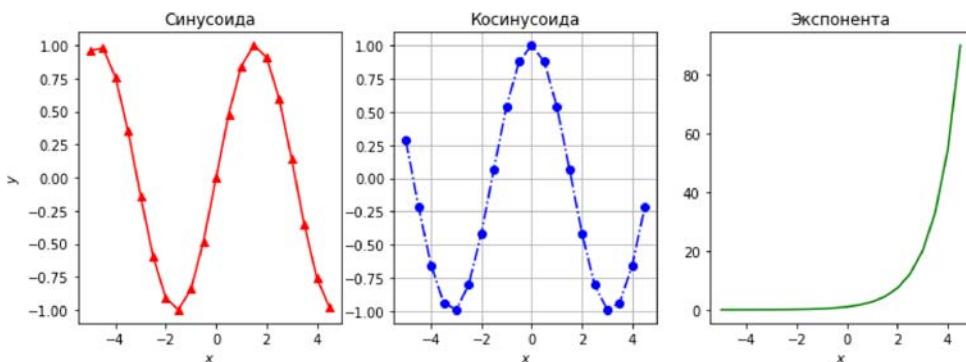
plt.subplots(1, 3, figsize=(12, 4))

plt.subplot(131)
plt.plot(x, np.sin(x), '^-r')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.title('Синусоида')

plt.subplot(132)
plt.plot(x, np.cos(x), 'o-.b')
plt.title('Косинусоида')
plt.xlabel('$x$')
plt.grid()

plt.subplot(133)
plt.plot(x, np.exp(x), '-g')
plt.xlabel('$x$')
plt.title('Экспонента')

plt.show()
```



При построении здесь использован подход к созданию графиков, ориентированный на его структуру. Рассмотрим, как построить аналогичные графики с использованием объектно-ориентированного подхода.

В объектно-ориентированном подходе используется аналогичный `subplot`-у метод `add_subplot()`. Он разбивает `Figure` на указанное количество строк и столбцов. После такого разбиения `Figure` можно представить как таблицу (или координатную сетку). Затем область `Axes` помещается в указанную ячейку. Каждая область `Axes` является независимой от других, то есть на них могут быть нарисованы самые разные графики и установлены самые разные параметры внешнего вида.

Для всего этого `add_subplot()` необходимо всего три числа, которые мы и передаем ему в качестве параметров:

- первое – количество строк;
- второе – количество столбцов
- третье – индекс ячейки.

Индексирование полученных ячеек начинается с левого верхнего угла, выполняется построчно слева-направо и заканчивается в правом нижнем углу.

Ввод [56]:

```

import matplotlib.pyplot as plt

x = np.arange(-5, 5, 0.25)

fig = plt.figure(figsize=(10, 5))

ax_1 = fig.add_subplot(2, 1, 1)
ax_2 = fig.add_subplot(2, 3, 4)
ax_3 = fig.add_subplot(2, 3, 5)
ax_4 = fig.add_subplot(2, 3, 6)

ax_1.set(title = 'ax_1', xticks=[], yticks[])
ax_1.plot(x, np.sin(x))

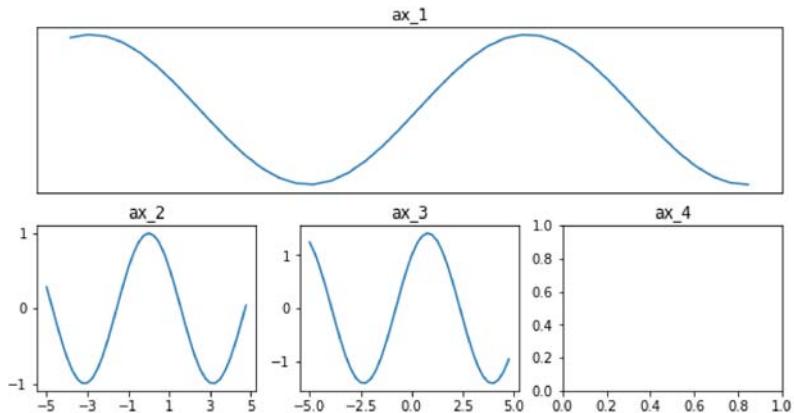
ax_2.set(title = 'ax_2', xticks=list(range(-5, 6, 2)), yticks=[-1, 0, 1])
ax_2.plot(x, np.cos(x))

ax_3.set(title = 'ax_3')
ax_3.plot(x, np.sin(x)+np.cos(x))

ax_4.set(title = 'ax_4')

plt.show()

```



Ввод [45]:

```

# Вывод трёх графиков с помощью subplot()
# Используем объектно-ориентированный подход к созданию графиков
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-5, 5, 0.5)

fig = plt.figure(figsize=(12, 4))

ax1 = fig.add_subplot(131)
ax2 = fig.add_subplot(132)
ax3 = fig.add_subplot(133)

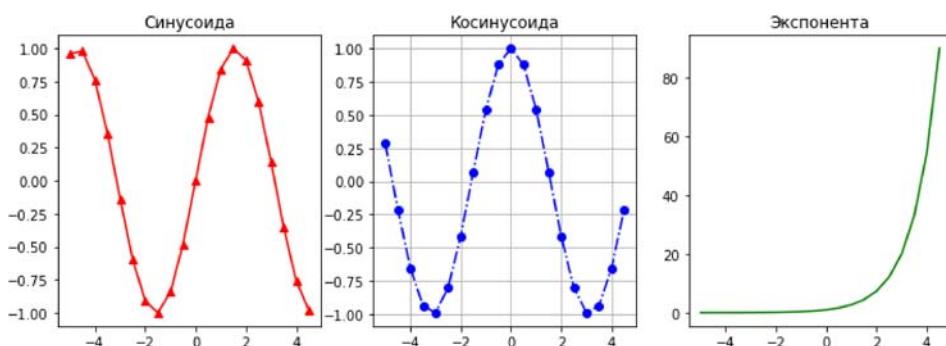
ax1.plot(x, np.sin(x), '^-r')
ax1.set_title('Синусоида')

ax2.plot(x, np.cos(x), 'o-b')
ax2.set_title('Косинусоида')
ax2.grid()

ax3.plot(x, np.exp(x), '-g')
ax3.set_title('Экспонента')

plt.show()

```



Почти все методы `axes` присутствуют в модуле `pyplot`. Например, при вызове `plt.title('Заголовок')` модуль `pyplot` вызывает `ax.set_title('Заголовок')`. Можно сказать, что модуль `pyplot` создает `Figure` и `Axes` автоматически (хотя это не совсем так). Но, поскольку (из дзена Python) "Явное лучше чем неявное", явное определение `Figure` и `Axes` делает код более очевидным и понятным, пускай даже за счет увеличения его объема.

## 8.2 Работа с функцией `subplots()`

Неудобство использования последовательного вызова функций `subplot()` заключается в том, что каждый раз приходится указывать количество строк и столбцов сетки. Для того, чтобы этого избежать, можно воспользоваться функцией `subplots()`.

Функция `subplots()` возвращает два объекта, первый – это `Figure`, подложка, на которой будут размещены поля с графиками, второй – объект (или массив объектов) `Axes`, через который можно получить полных доступ к настройке внешнего вида отображаемых элементов. В параметрах фигуры необходимо указать количество строк и столбцов, в которых будут размещаться графики. При желании можно задать размер фигуры в дюймах (ширина X высота):

```
fig, axs = plt.subplots(n, m, figsize(a, b))
```

После выполнения кода переменная `axs` содержит список, состоящий из  $n \times m$  объектов. Обращаться к элементам списка можно по индексам (от 0 до  $n \times m - 1$ ). Индексация начинается с нуля и для нумерации строк и для нумерации столбцов.

По умолчанию количество строк и столбцов в методе `subplots` равно 1, что удобно для быстрого создания `Figure` с одной областью `Axes`:

```
import matplotlib.pyplot as plt

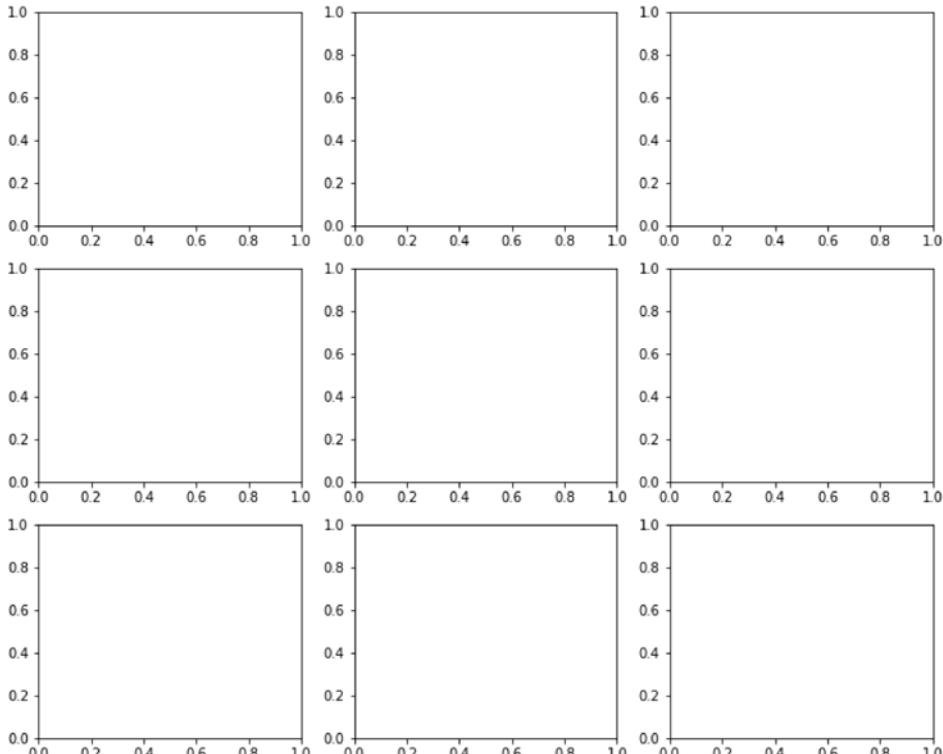
fig, axs = plt.subplots()      # одна строка вместо двух:
# fig = plt.figure()
# axs = fig.add_subplot(111)

axs.set(title='Axes')
plt.show()
```

Ввод [54]:

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(figsize=(12, 10), nrows = 3, ncols = 3)
plt.show()
```



Ввод [49]:

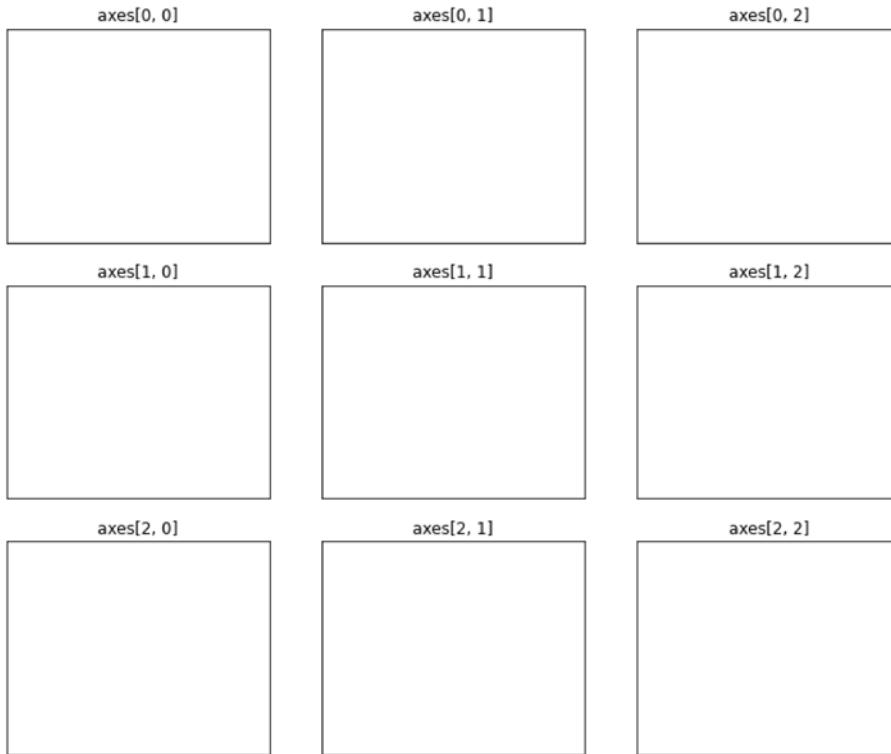
```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(figsize=(12, 10), nrows = 3, ncols =3)

for i in range(0, 3):
    for j in range(0, 3):
        axes[i,j].set(title='axes[' + str(i) + ', ' + str(j) + ']')

for ax in axes.flat:
    ax.set(xticks=[], yticks=[])

plt.show()
```



Ввод [50]:

```
# Вывод четырёх графиков с помощью subplots()
import numpy as np
import matplotlib.pyplot as plt

x1 = np.arange(-10, 10, 0.25)
x2 = np.arange(-4, 4, 0.5)

fig, axs = plt.subplots(2, 2, figsize=(12, 10))

axs[0, 0].plot(x2, np.sinh(x2), '-r', label=r'$y=\sinh(x)$')
axs[0, 0].set(title='Гиперболический синус')
axs[0, 0].grid(axis='both', color='g', linestyle='-', linewidth=0.25)

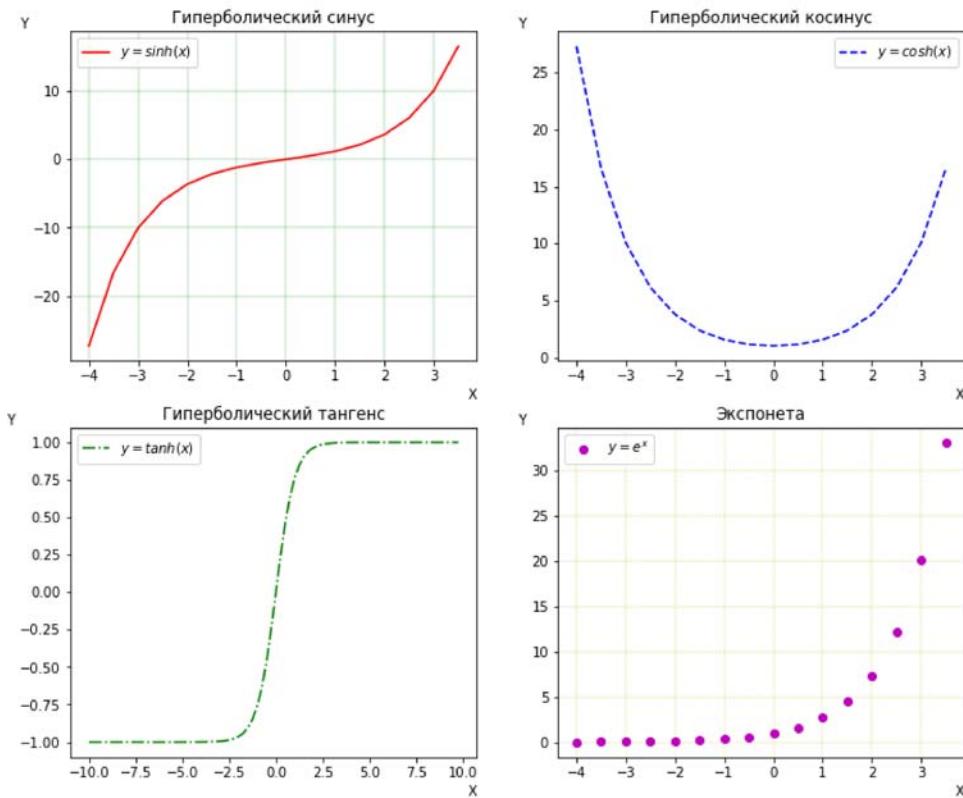
axs[0, 1].plot(x2, np.cosh(x2), '--b', label=r'$y=\cosh(x)$')
axs[0, 1].set(title='Гиперболический косинус')

axs[1, 0].plot(x1, np.tanh(x1), '-.g', label=r'$y=\tanh(x)$')
axs[1, 0].set(title='Гиперболический тангенс')

axs[1, 1].plot(x2, np.exp(x2), 'om', label=r'$y=e^x$')
axs[1, 1].set(title='Экспонента')
axs[1, 1].grid(axis='both', color='y', linestyle='-.', linewidth=0.3)

for ax in axs.flat:
    ax.set_xlabel('X', loc='right')
    ax.set_ylabel('Y', loc='top', rotation='horizontal')
    ax.legend(loc='best')

plt.show()
```



Здесь `fig` - это `Figure`, а `axs` - это массив `NumPy`, элементами которого являются объекты `Axes`. Далее для каждой области `Axes` определили свою кривую (метод `plot`) и свой заголовок (метод `set`). Поскольку каждый заголовок уникальный, приходится вручную вызвать каждую область `Axes` и устанавливать для нее параметр `title`.

Параметры, одинаковые для каждой области, можно установить в цикле, например как в примере:

```
for ax in axs.flat:
    ax.set_xlabel('X', loc='right')
    ax.set_ylabel('Y', loc='top', rotation='horizontal')
    ax.legend(loc='best')
```

### 8.3 Работа с функцией `GridSpec()`

Рассмотренные ранее подходы к компоновке графиков хороши при размещении графиков в ячейках регулярной сетки. Для более сложных элементов компоновки хорошо подходит класс `GridSpec` из ветки:

```
matplotlib.gridspec
```

То есть, вначале его нужно импортировать, например, так:

```
from matplotlib.gridspec import GridSpec
```

Далее, с его помощью мы также разбиваем все пространство фигуры на строки и столбцы:

```
fig = plt.figure(figsize=(7, 4))
gs = GridSpec(ncols=3, nrows=2, figure=fig)
```

В конструктор класса `GridSpec` передаётся количество столбцов, строк и фигура, на которой все будет отображено.

Альтернативный вариант создания объекта `GridSpec` выглядит так:

```
fig = plt.figure(figsize=(7, 4))
gs = fig.add_gridspec(2, 3)
```

Здесь `fig` - это объект `Figure`, у которого есть метод `add_gridspec()`, позволяющий добавить на него сетку с заданными параметрами (в нашем случае две строки и три столбца).

Затем, используя синтаксис срезов пакета `numpy`, можно формировать координатные оси, охватывающие произвольное число ячеек.

Эта функция позволяет разбить область на сетку подграфиков, каждого из которых можно присвоить свой график, так что результатом будет сетка с подграфиками разных размеров с разными направлениями.

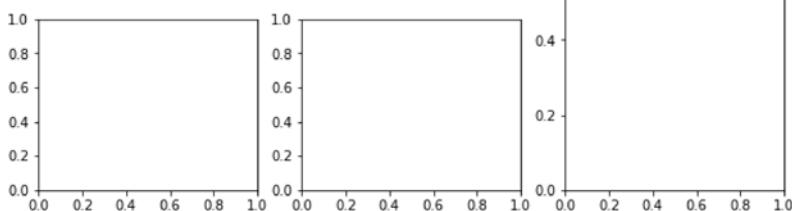
Ввод [58]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

fig = plt.figure(figsize=(10, 8))
gs = GridSpec(ncols=3, nrows=3, figure=fig) # gs = fig.add_gridspec(3, 3)

ax1 = plt.subplot(gs[0, 0:3], xticks=[], yticks[])
ax2 = fig.add_subplot(gs[1, 0:2], xticks=[], yticks[])
ax3 = plt.subplot(gs[1:3, 2])
ax4 = fig.add_subplot(gs[2, 0])
ax5 = fig.add_subplot(gs[2, 1])

plt.show()
```



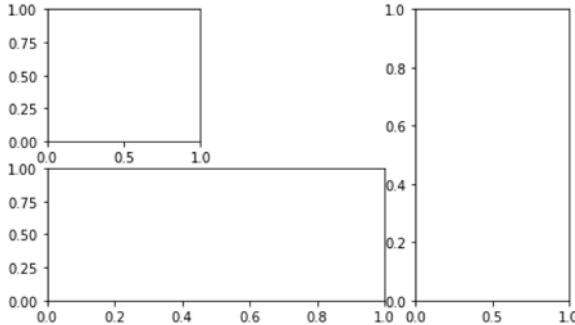
Причем, обратите внимание, что можно формировать оси как с помощью функции `subplot()` (в примере `ax1`, `ax3`), так и с помощью метода `add_subplot()` (в примере `ax2`, `ax4`, `ax5`). То есть, можно использовать любой способ формирования полей для вывода графиков.

Ввод [58]:

```
fig = plt.figure(figsize=(7, 4))
gs = GridSpec(ncols=3, nrows=2, figure=fig)

ax1 = plt.subplot(gs[0, 0])
ax2 = plt.subplot(gs[1, 0:2])
ax3 = fig.add_subplot(gs[:, 2])

plt.show()
```



При использовании `GridSpec` можно задавать размеры выводимых графиков, передав списки с размерами по ширине и высоте в параметры `width_ratios` и `height_ratios` в конструктор `GridSpec()`.

Ввод [60]:

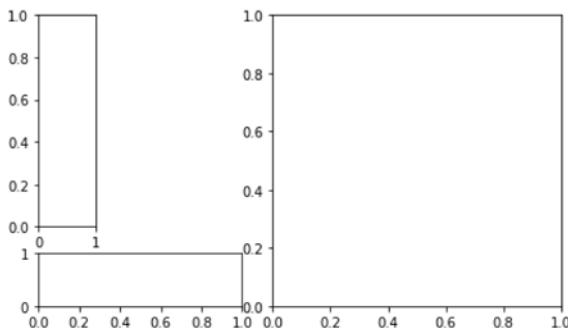
```
fig = plt.figure(figsize=(7, 4))

ws = [1, 2, 5] # какую долю по ширине будет занимать каждый столбец: 1/8, 2/8, 5/8
hs = [2, 0.5] # какую высоту по высоте будет занимать каждая строка

gs = GridSpec(ncols=3, nrows=2, figure=fig, width_ratios=ws, height_ratios=hs)

ax1 = plt.subplot(gs[0, 0])
ax2 = plt.subplot(gs[1, 0:2])
ax3 = fig.add_subplot(gs[:, 2])

plt.show()
```



Осталось разобраться, как заполнить сетку данными. Для этого нужно использовать объект `Axes`, который возвращает каждая функция `add_subplot()`. В конце остается вызвать `plot()` для вывода конкретного графика.

Ввод [81]:

```
gs = plt.GridSpec(3,3)

fig = plt.figure(figsize=(12, 10))

x = np.linspace(-10, 10, 50)
y = np.sin(x)

x1 = np.array([1, 3, 2, 5])
y1 = np.array([4, 3, 7, 2])

x2 = np.arange(5)
y2 = np.array([3, 2, 4, 6, 4])

s1 = fig.add_subplot(gs[1,:2])
s1.plot(x, y, 'r')

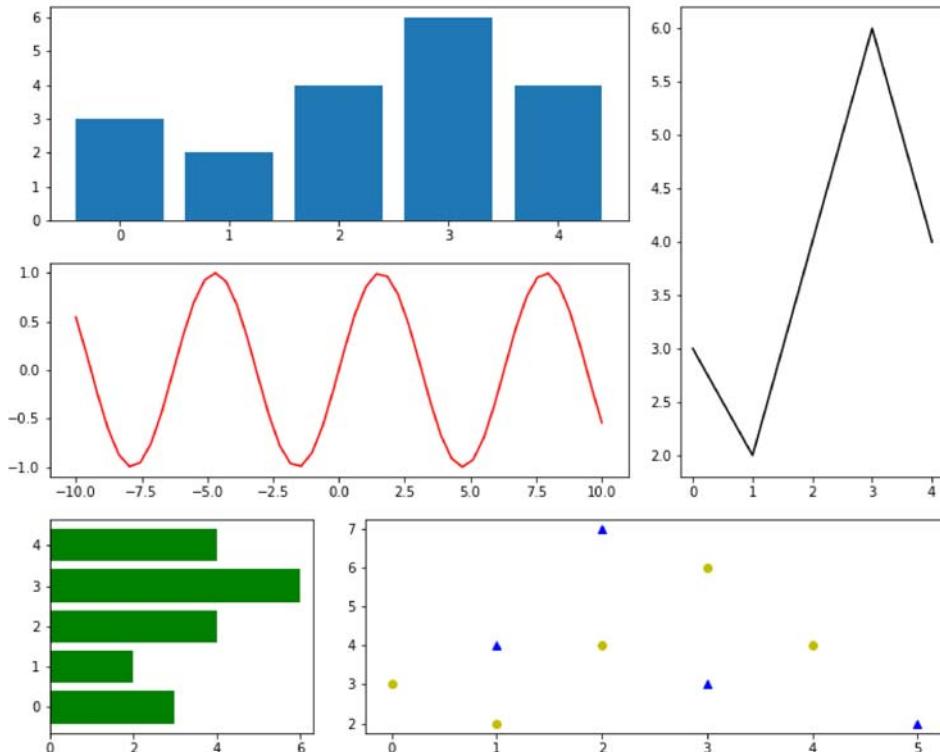
s2 = fig.add_subplot(gs[0,:2])
s2.bar(x2, y2)

s3 = fig.add_subplot(gs[2,0])
s3.barh(x2, y2, color='g')

s4 = fig.add_subplot(gs[:,2])
s4.plot(x2, y2, 'k')

s5 = fig.add_subplot(gs[2, 1:])
s5.plot(x1, y1, 'b^', x2, y2, 'yo')

plt.show()
```



Ввод [160]:

```

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
import random

fig = plt.figure(figsize=(10, 8))
gs = GridSpec(ncols=3, nrows=3, figure=fig) # gs = fig.add_gridspec(3, 3)

# ax1
ax1 = fig.add_subplot(gs[:, 2])
x = range(10)
y = [random.randint(5, 45) for _ in range(10)]
ax1.barh(x, y, color='pink')

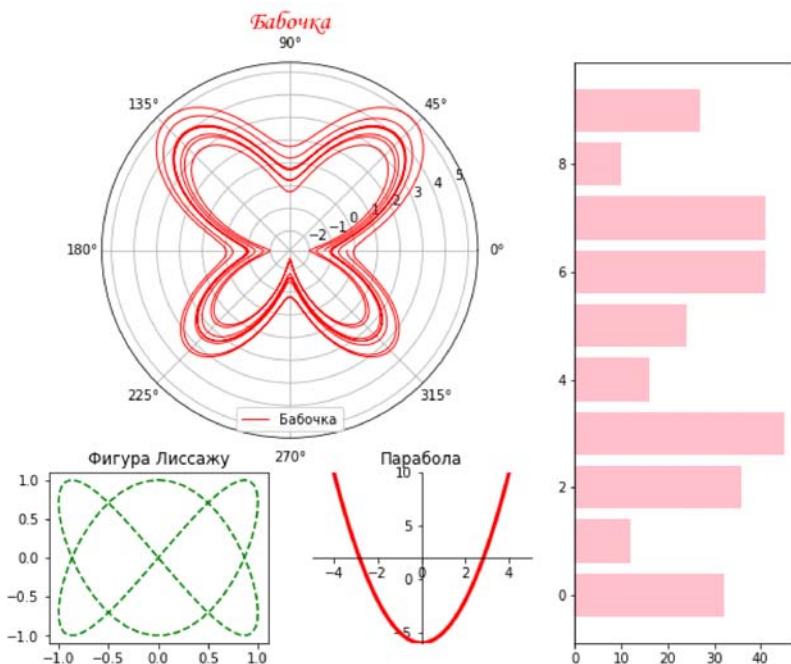
# ax2
ax2 = fig.add_subplot(gs[0:2, 0:2], polar=True)
phi = np.arange(-8 * np.pi, 8 * np.pi, 0.01)
r = np.exp(1) ** np.sin(phi) - 2 * np.cos(4 * phi) + (np.sin((2 * phi - np.pi) / 24)) ** 5
ax2.plot(phi, r,
          color='red',
          linewidth = 1,
          label='Бабочка')
ax2.set_title('Бабочка',
              family="Monotype Corsiva",
              weight="heavy",
              size=20,
              color='red')
ax2.legend(loc=8)

# ax3
ax3 = fig.add_subplot(gs[2, 0])
t = np.linspace(0, 2 * np.pi, 100)
x = np.sin(2 * t)
y = np.cos(3 * t)
ax3.plot(x, y, '--g')
ax3.set_title('Фигура Лиссажу')

# ax4
ax4 = fig.add_subplot(gs[2, 1])
x = np.linspace(-5, 5, 100)
y = x**2 - 6
ax4.set_xlim(-5, 5)
ax4.set_ylim(-6, 10)
ax4.spines["left"].set_position ("center")
ax4.spines["bottom"].set_position("center")
ax4.spines['top'].set_visible(False)
ax4.spines['right'].set_visible(False)
ax4.plot(x, y, color='red', linewidth=3)
ax4.set_title('Парабола')

plt.show()

```



## § 9 Примеры построения 2D графиков

Теперь, когда вы знаете, как настраивать параметры графиков, рассмотрим построение различных диаграмм, которые могут пригодиться для визуализации информации.

Все графики строятся по точкам, координаты которых хранятся в массивах *NumPy*. Для формирования значений *x* можно воспользоваться одним из вариантов:

- процедурой `np.linspace(start, stop, count_point)`, которая возвращает указанное количество чисел на интервале, определенным пользователем;
- процедурой `np.arange(beg, end, step)`, которая получает массив чисел от начального значения до конечного с определенным шагом.

Вспомним (из всего выше сказанного) основные **функции**, которые пригодятся при **настройке внешнего вида** графиков:

- `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.

Кроме этого перечислим некоторые **функции отрисовки**, которые далее будут применяться при построении графиков:

- `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`.

Перед построением графиков импортируем необходимые библиотеки и настроим вариант отображения графиков:

Ввод [44]:

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

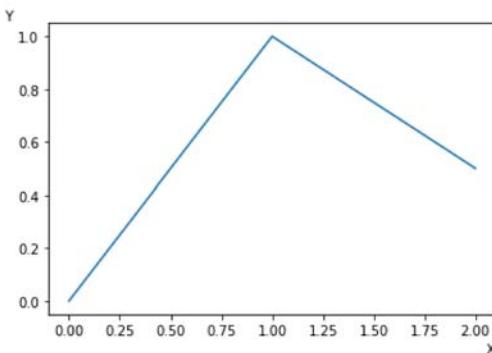
%matplotlib inline
```

## 9.1 Построение ломаной по точкам

**Пример.** Рисуем график с помощью списка у-координат; *x*-координаты образуют последовательность 0, 1, 2, ... . Точки соединяются прямыми, т.е. строится ломаная линия

Ввод [2]:

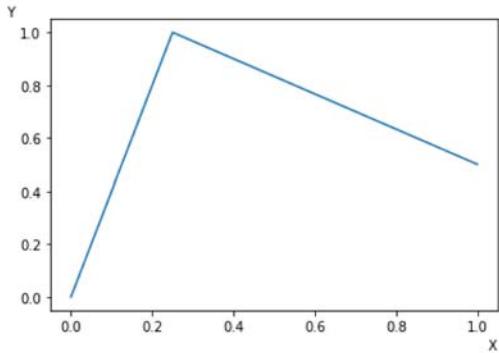
```
plt.figure()
plt.plot([0, 1, 0.5])
plt.xlabel('X', loc='right')
plt.ylabel('Y', loc='top', rotation='horizontal')
plt.show()
```



**Пример.** Пусть задан списки координат точек *x* и *y*. Точки соединяются прямыми, т.е. строится ломаная линия.

Ввод [3]:

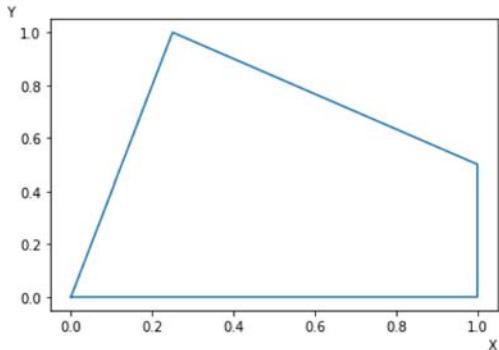
```
plt.figure()
plt.plot([0, 0.25, 1], [0, 1, 0.5])
plt.xlabel('X', loc='right')
plt.ylabel('Y', loc='top', rotation='horizontal')
plt.show()
```



$x$ -координаты не обязаны монотонно возрастать. Тут, например, мы строим замкнутый многоугольник.

Ввод [4]:

```
plt.figure()
plt.plot([0, 0.25, 1, 1, 0], [0, 1, 0.5, 0, 0])
plt.xlabel('X', loc='right')
plt.ylabel('Y', loc='top', rotation='horizontal')
plt.show()
```



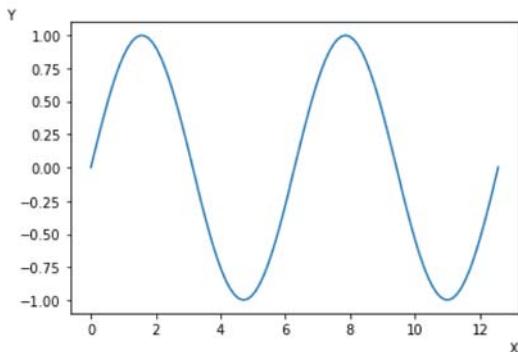
## 9.2 Построение графиков функций, заданных аналитически

Построение графиков функций производится на основе того факта, что когда точек много, ломаная неотличима от гладкой кривой.

Ввод [5]:

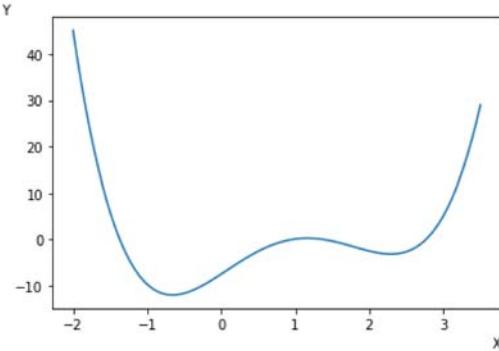
```
x = np.linspace(0, 4 * np.pi, 100)
y = np.sin(x)

plt.figure()
plt.plot(x, y)
plt.xlabel('X', loc='right')
plt.ylabel('Y', loc='top', rotation='horizontal')
plt.show()
```



Ввод [6]:

```
x = np.linspace(-2, 3.5, 100)
y = (1.5*x ** 4 - 5.6 * x ** 3 + 1.2 * x ** 2 + 10.5 * x - 7.5)
plt.figure()
plt.plot(x, y)
plt.xlabel('X', loc='right')
plt.ylabel('Y', loc='top', rotation='horizontal')
plt.show()
```



### 9.3 Диаграмма рассеивания или точечный график

**Диаграмма рассеивания** – это полезный способ исследования соотношений между двумя одномерными рядами данных (демонстрации наличия или отсутствия корреляции между двумя переменными), отображающий значения двух переменных в виде точек на декартовой плоскости.

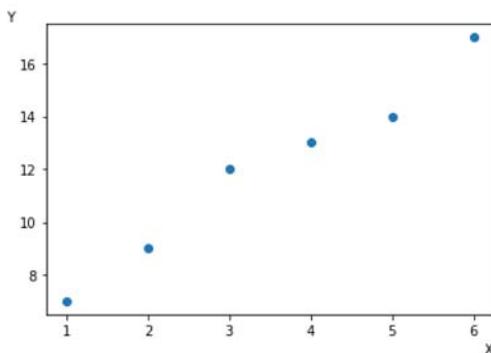
На диаграмме рассеяния каждому наблюдению (или элементарной единице набора данных) соответствует точка, координаты которой (в декартовой системе координат) равны значениям двух каких-то параметров этого наблюдения. Если предполагается, что один из параметров зависит от другого, то обычно значения независимого параметра откладывается по горизонтальной оси, а значения зависимого – по вертикальной.

Для отображения точечного графика предназначена функция `scatter()`. Функция `scatter` просто рисует точки, не соединяя их линиями.

В простейшем виде точечный график можно получить, передав функции `scatter()` наборы точек для `x`, `y` координат:

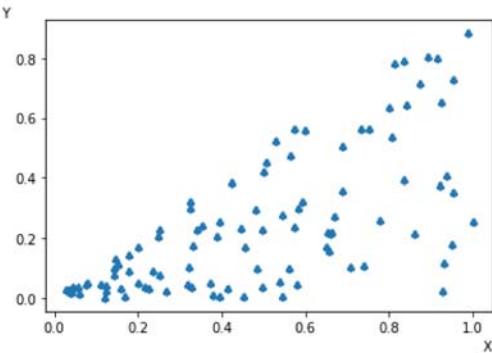
Ввод [161]:

```
x = range(1, 7)
y = [7, 9, 12, 13, 14, 17]
plt.figure()
plt.scatter(x, y)
plt.xlabel('X', loc='right')
plt.ylabel('Y', loc='top', rotation='horizontal')
plt.show()
```



Ввод [169]:

```
x = np.random.rand(100)
y = np.random.rand(100)
plt.scatter(x, y*x, marker=r'$\clubsuit$')
plt.xlabel('X', loc='right')
plt.ylabel('Y', loc='top', rotation='horizontal')
plt.show()
```



Для более детальной настройки необходимо воспользоваться дополнительными параметрами функции `scatter()`.

**Синтаксис функции:**

```
scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None,
edgecolors=None, *, plotnonfinite=False, data=None, **kwargs)
```

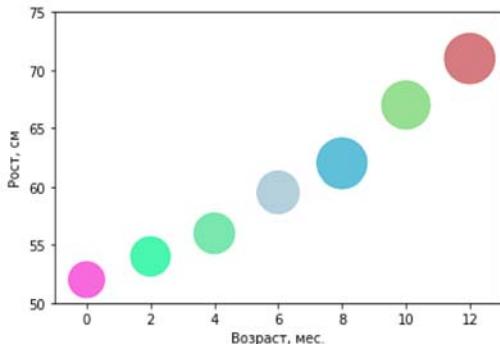
Рассмотрим некоторые из её параметров:

- `x` - набор данных для оси абсцисс (ось `x`) (массив, `shape(n, )`);
- `y` - набор данных для оси ординат (ось `y`) (массив, `shape(n, )`);
- `s` - масштаб точек (скалярная величина или массив, `shape(n, ), optional`);
- `c` - цвет или набор цветовых элементов;
- `marker` - стиль точки (`Markerstyle`);
- `cmap` - цветовая карта (`str, Colormap, optional`, значение по умолчанию: `'None'`);
- `norm` - нормализация данных (`Normalize, optional`, значение по умолчанию: `None`);
- `alpha` - прозрачность скалярная величина (`optional`, значение по умолчанию: `None`);
- `linewidths` - ширина границы маркера (скалярная величина или массив, `optional`, значение по умолчанию: `None` )
- `edgecolors` - цвет границы { `face` , `none` , `None` } (цвет или набор цветовых элементов, `optional`).

Ввод [37]:

```
age = np.array([0, 2, 4, 6, 8, 10, 12])
w = np.array([3.6, 4.32, 4.6, 4.93, 7, 6.5, 7.1])
h = np.array([52, 54, 56, 59.5, 62, 67, 71])
colors = ['#f543d4', '#1af59a', '#56e29c', '#a2c5d4', '#38b1d0', '#7dd777', '#cc5a5f']
sizes = 200 * w

plt.scatter(age, h, s=sizes, alpha=0.8, c=colors)
plt.xlabel('Возраст, мес.')
plt.xlim(-1, 13)
plt.ylabel('Рост, см')
plt.ylim(50, 75)
plt.show()
```



Ввод [2]:

```

import numpy as np
import matplotlib.pyplot as plt

x = np.random.rand(50)
y1 = np.random.rand(50)
y2 = np.random.rand(50)

fig, axes = plt.subplots(2, 2)

axes[0][0].set_facecolor('black')
axes[0][0].scatter(x, y1,
                   marker = 's',
                   c = 'fuchsia')
axes[0][0].set_title('marker, c')

colors_1 = np.random.rand(50)
axes[0][1].scatter(x, y1,
                   marker = '*',
                   c = colors_1,
                   s = 700)
axes[0][1].set_title('marker, c, s')

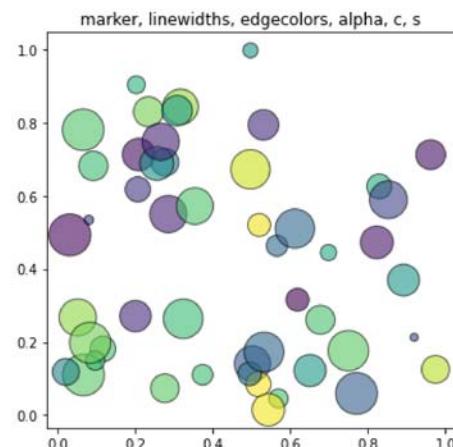
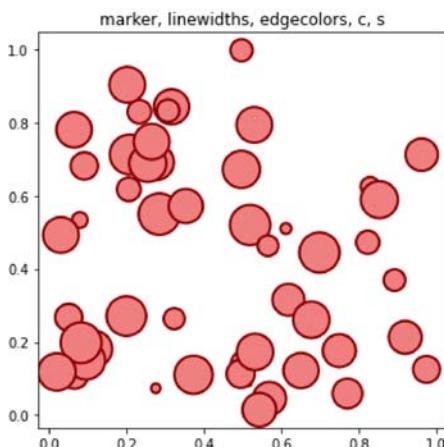
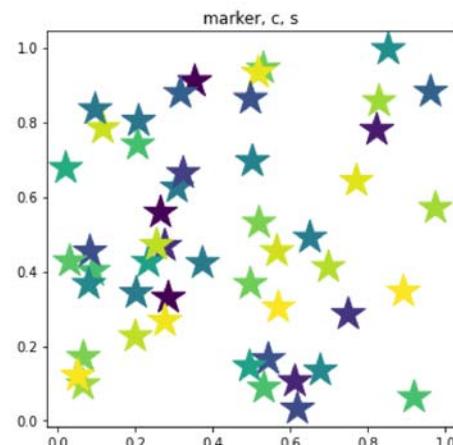
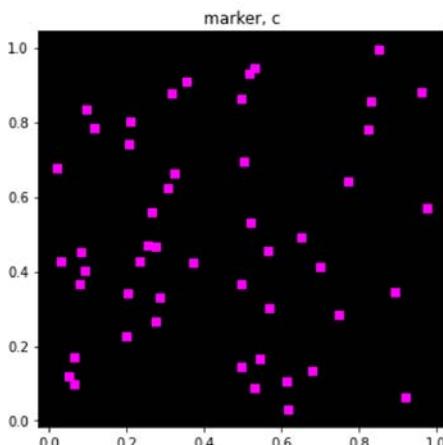
size = 1000*np.random.rand(50)
axes[1][0].scatter(x, y2,
                   marker = 'o',
                   c = 'lightcoral',
                   s = size,
                   linewidths = 2,
                   edgecolors = 'darkred')
axes[1][0].set_title('marker, linewidths, edgecolors, c, s')

size = 1000*np.random.rand(50)
colors_2 = np.random.rand(50)
axes[1][1].scatter(x, y2,
                   marker = 'o',
                   c = colors_2,
                   s = size,
                   edgecolors = 'black',
                   alpha = 0.6)
axes[1][1].set_title('marker, linewidths, edgecolors, alpha, c, s')

fig.set_figwidth(12)      # ширина и
fig.set_figheight(12)     # высота "Figure"

plt.show()

```



## 9.4 Stem-график

Визуально этот график выглядит как набор линий от точки с координатами  $(x, y)$  до базовой линии, в верхней точке которой ставится маркер.

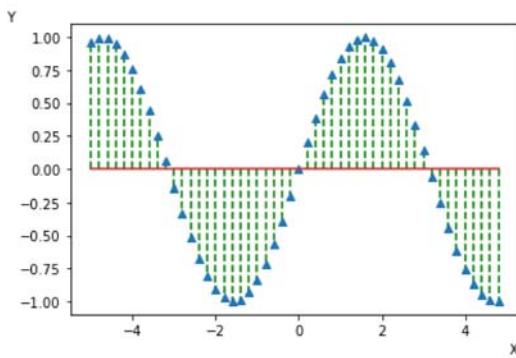
Параметры функции `stem()` :

- `linefmt` – стиль вертикальной линии ( `-`, `--`, `-.`, `:` );
- `markerfmt` – формат маркера ( `o`, `+`, `*`, `^`, `<`, `h` и др.);
- `basefmt` – формат базовой линии;
- `bottom` –  $y$ -координата базовой линии (по умолчанию 0).

Стили вертикальной линии и формат маркеров, который можно использовать такие же, как рассмотренные выше.

Ввод [172]:

```
x = np.arange(-5, 5, 0.2)
y = np.array([np.sin(i) for i in x])
plt.xlabel('X', loc='right')
plt.ylabel('Y', loc='top', rotation='horizontal')
plt.stem(x, y, linefmt="g--", markerfmt="^");
```



## 9.5 График функции, заданной параметрически

Matplotlib позволяет строить параметрическую линию  $x = x(t)$ ,  $y = y(t)$ .

Ввод [173]:

```
t = np.linspace(0, 2 * np.pi, 100)
x = (1 + np.cos(t)) * np.cos(t)
y = (1 + np.cos(t)) * np.sin(t)

fig, axs = plt.subplots(1, 3, figsize=(12, 4))

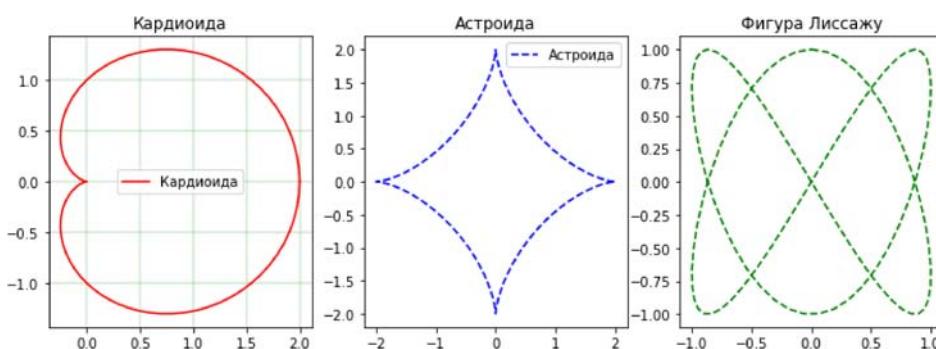
axs[0].plot(x, y, '-r', label='Кардиоида')
axs[0].set_title('Кардиоида')
axs[0].grid(axis='both', color='g', linestyle='-', linewidth=0.25)
axs[0].legend(loc='best')

x = 2 * np.sin(t) ** 3
y = 2 * np.cos(t) ** 3

axs[1].plot(x, y, '--b', label='Астроида')
axs[1].legend(loc='best')
axs[1].set_title('Астроида')

x = np.sin(2 * t)
y = np.cos(3 * t)

axs[2].plot(x, y, '--g')
axs[2].set_title('Фигура Лиссажу');
```



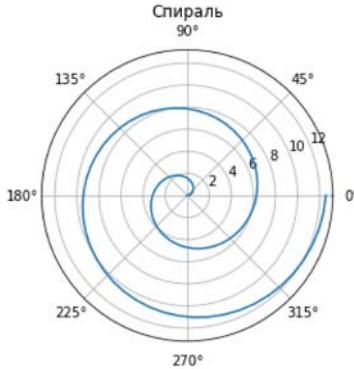
## 9.6 График функции, заданной в полярных координатах

Первый массив – угол  $\varphi$ , второй – расстояние  $r$ .

Ввод [174]:

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

plt.figure()
plt.polar(phi, phi)
plt.title('Сpirаль')
plt.show()
```



Ввод [175]:

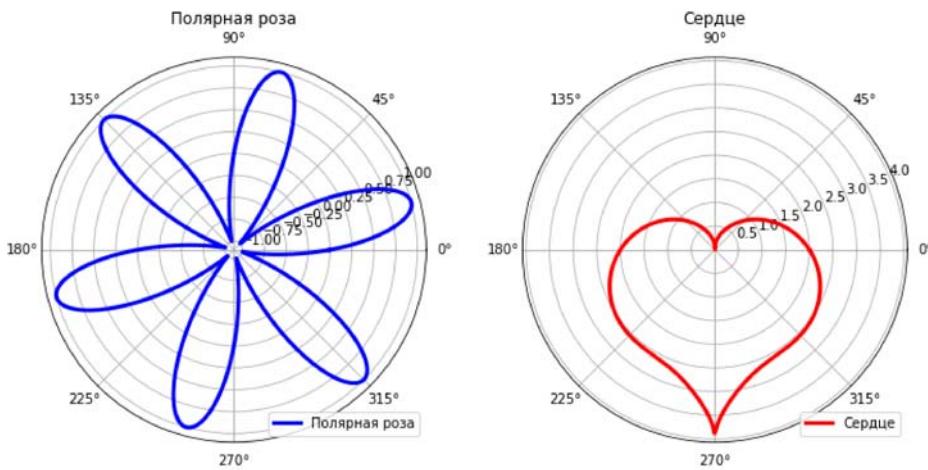
```
fig = plt.figure(figsize=(10, 4))

phi = np.arange(0, 2*np.pi, 0.01)
r1 = np.sin(6*phi)
r2 = 2 - 2*np.sin(phi) + np.sin(phi)*np.sqrt(abs(np.cos(phi)))/(np.sin(phi) + 1.4)

# Добавление на рисунок круговой области рисования
ax1 = fig.add_axes([0, 0, 1, 1], polar=True)
ax1.plot(phi, r1, label='Полярная роза', color='blue', linewidth=3)
ax1.set_title('Полярная роза')
ax1.legend(loc=4)

# Добавление на рисунок круговой области рисования
ax2 = fig.add_axes([0.5, 0, 1, 1], polar=True)
ax2.plot(phi, r2, label='Сердце', color='red', linewidth=3)
ax2.set_title('Сердце')
ax2.legend(loc=4)

plt.show()
```



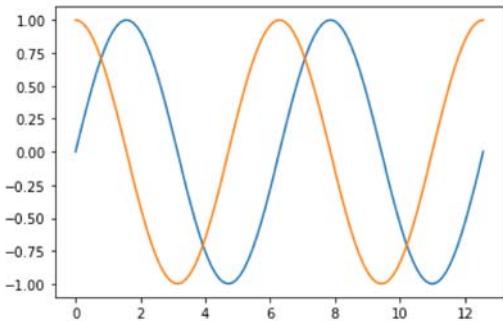
## 9.7 Несколько кривых на одном графике

При построении нескольких кривых на одном графике, каждая кривая задаётся парой массивов –  $x$  и  $y$  координаты. По умолчанию, им присваиваются цвета из некоторой последовательности цветов, которые можно изменить.

Ввод [176]:

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

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



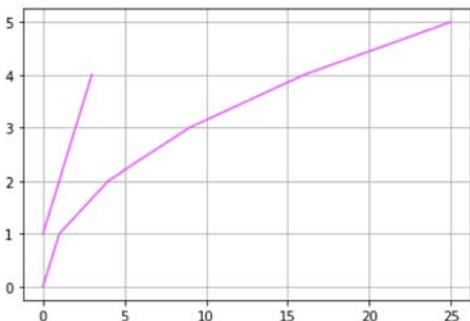
Ввод [34]:

```
y1 = np.array(range(6))
x1 = np.array([n*n for n in y1])

x2 = [0, 1, 2, 3]
y2 = [n + 1 for n in x2]

plt.plot(x1, y1, x2, y2, color=(0.9, 0.3, 1, 0.9))

plt.grid()
plt.show()
```

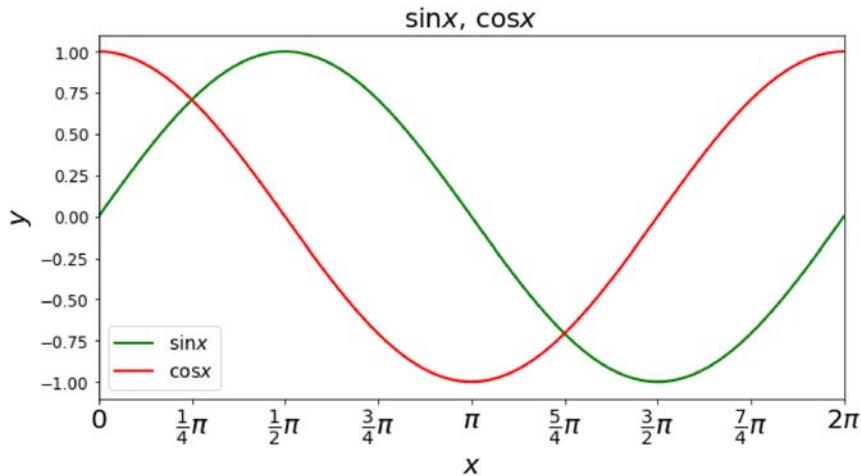


Вот пример настройки почти всего, что можно настроить. Можно задать последовательность засечек на оси  $x$  (и  $y$ ) и подписи к ним, в которых, как и в других текстах, можно использовать *LTEX*-овские обозначения. Задать подписи осей  $x$  и  $y$  и заголовок графика. Во всех текстовых элементах можно задать размер шрифта. Можно задать толщину линий и штрихи. Можно задать подписи к кривым ( legend ); где разместить эти подписи тоже можно регулировать.

Ввод [51]:

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

plt.figure(figsize=(10, 5))
plt.plot(x, np.sin(x), linewidth=2, color='g', label=r'$\sin x$')
plt.plot(x, np.cos(x), linewidth=2, color='r', label=r'$\cos x$')
plt.axis([0, 2 * np.pi, -1.1, 1.1])
plt.xticks(np.linspace(0, 2 * np.pi, 9), # Где сделать отмечки
          ('0', r'$\frac{1}{4}\pi$', r'$\frac{1}{2}\pi$', r'$\frac{3}{4}\pi$', r'$\pi$', r'$\frac{5}{4}\pi$', r'$\frac{3}{2}\pi$', r'$\frac{7}{4}\pi$', '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=14, loc=0)
plt.show()
```



## 9.8 Заливка области между кривыми

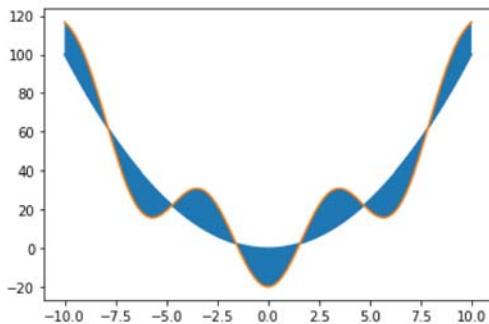
В matplotlib для построения графиков с заполненными областями между кривыми необходимо воспользоваться методом `fill_between`.

Ввод [177]:

```
x = np.linspace(-10, 10, 100)
y1 = x**2
y2 = x**2 - 20*np.cos(x)

plt.plot(x, y1)
plt.plot(x, y2)
plt.fill_between(x, y1, y2)

plt.show()
```



Ввод [52]:

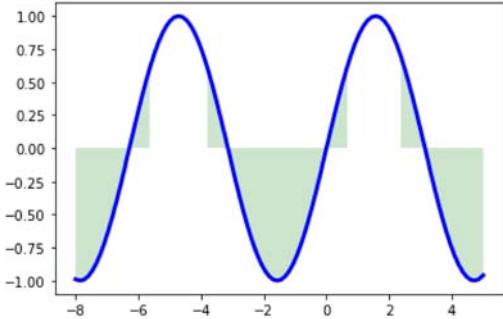
```
x = np.linspace(-8, 5, 100)
y = np.sin(x)

fig, ax = plt.subplots()

ax.plot(x, y, color = 'b', linewidth = 3)

ax.fill_between(x, y,
                 where= (y < 0.7),
                 facecolor='green',
                 alpha = 0.2)

plt.show()
```



Ввод [43]:

```
x = np.linspace(-12, 12, 200)
y1 = x**2
y2 = x**2 - 40*np.cos(x)

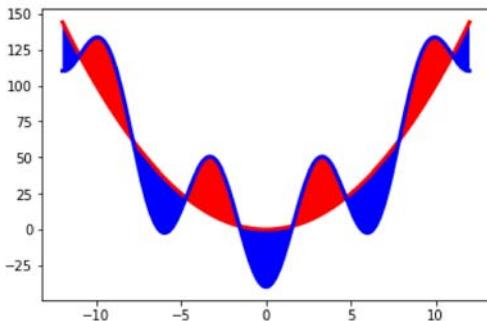
fig, ax = plt.subplots()

ax.plot(x, y1, color = 'r', linewidth = 3)
ax.plot(x, y2, color = 'b', linewidth = 3)

ax.fill_between(x, y1, y2,
                 where= (y2 > y1),
                 facecolor='red')
ax.fill_between(x, y1, y2,
                 where= (y2 < y1),
                 facecolor='blue')
```

Out[43]:

&lt;matplotlib.collections.PolyCollection at 0x1a7fc5b7af0&gt;



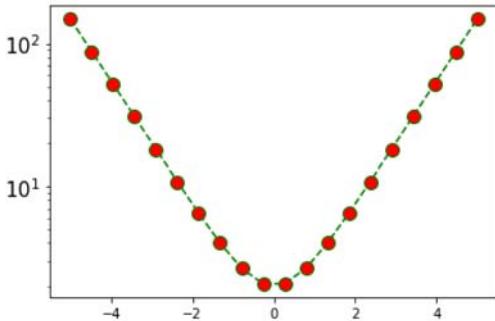
## 9.9 Использование логарифмического масштаба

Если  $y$  меняется на много порядков, то удобно использовать логарифмический масштаб по  $y$ .

Ввод [179]:

```
x = np.linspace(-5, 5, 20)

plt.figure()
plt.plot(x, np.exp(x) + np.exp(-x), linestyle='--', color='green', marker='o', markersize=10, markerfacecolor='red')
plt.yscale('log')
plt.yticks(fontsize=15)
plt.show()
```

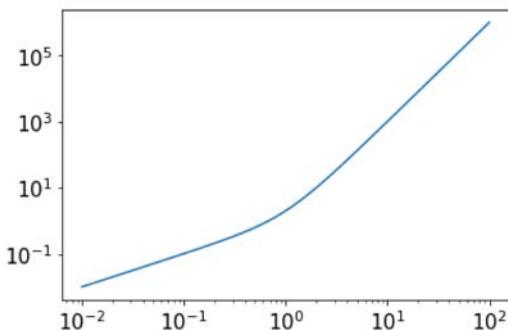


Можно задать логарифмический масштаб по обоим осям.

Ввод [198]:

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

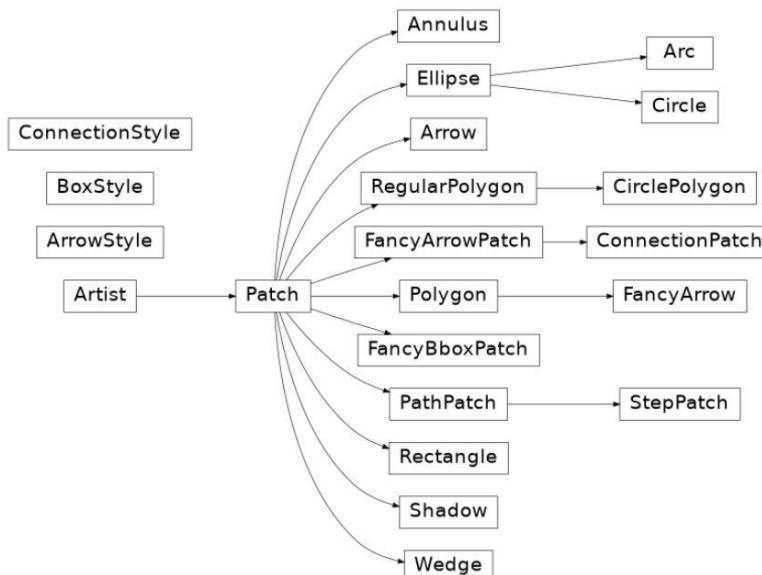
plt.figure()
plt.plot(x, x + x ** 3)
plt.xscale('log'), plt.xticks(fontsize=15)
plt.yscale('log'), plt.yticks(fontsize=15)
plt.show()
```



## 9.10 Создание плоских фигур

Рисование плоских фигур осуществляется с помощью модуля `matplotlib.pyplot`. Плоские геометрические фигуры создаются с помощью библиотеки `matplotlib.patches` ([https://matplotlib.org/stable/api/patches\\_api.html](https://matplotlib.org/stable/api/patches_api.html)), из которой можно импортировать шаблоны для различных плоских геометрических фигур:

```
from matplotlib.patches import Circle, Wedge, Polygon, Ellipse, Arc, Path, PathPatch
```



**Алгоритм рисования**

Все геометрические объекты рисуются в несколько шагов.

**Шаг 1.** Импортируются все необходимые модули, функции, шаблоны:

```
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
```

**Шаг 2.** Определяются координаты окна, в котором будет создаваться изображение, например:

```
plt.xlim(0, 12)
plt.ylim(0, 12)
```

**Шаг 3.** Создается область для рисования, связанная с осями координат с помощью метода `gca()`:

```
ax = plt.gca()
```

**Шаг 4.** Создается геометрическая фигура на основе описания из модуля `matplotlib.patches`. Например, круг с центром в точке (6, 7) и радиусом 5 и заносим результат в переменную `'circle'`:

```
circle = Circle((6, 7), 5)
```

**Шаг 5.** Созданная фигура добавляется в область `ax` с помощью метода `add_patch()`:

```
ax.add_patch(circle)
```

**Шаг 6.** Рисунок отображается в графическом окне:

```
plt.show()
```

Таблица. Функции создания примитивов

Функция и ее параметры	Описание функции
<code>Rectangle((x, y), width, height)</code>	Создает прямоугольник, левый верхний угол которого располагается в точке с координатами <code>(x, y)</code> , высота которого равна <code>width</code> , а длина - <code>height</code>
<code>Circle((x, y), radius)</code>	Создает круг с центром в точке <code>(x, y)</code> , радиуса <code>radius</code>
<code>Ellipse((x, y), width, height)</code>	Создает эллипс с центром в точке <code>(x, y)</code> , диаметр которого по горизонтальной оси равен <code>width</code> , а диаметр по вертикальной - <code>height</code>
<code>Wedge((x, y), radius, t1, t2)</code>	Создает сектор с центром в точке <code>(x, y)</code> , радиуса <code>radius</code> , ограниченный линиями углов <code>t1</code> и <code>t2</code>
<code>Arc((x, y), width, height, angle, t1, t2)</code>	Создает дугу с центром в точке <code>(x, y)</code> , диаметр которой по горизонтальной оси равен <code>width</code> , а диаметр по вертикальной - <code>height</code> , угол поворота дуги относительно центра <code>angle</code> , дуга ограничена линиями углов <code>t1</code> и <code>t2</code>
<code>Polygon([(x0, y0), (x1, y1), (x2, y2), ...], closed)</code>	Создает ломаную линию по точкам <code>(x0, y0), (x1, y1), (x2, y2), ...</code> , если параметр <code>closed</code> равен <code>True</code> (или он отсутствует), первая точка соединяется с последней, в противном случае ( <code>closed= False</code> ), ломаная линия остается незамкнутой

Основные свойства, описывающие стиль вывода каждой фигуры, представлены в таблице, большинство из них могут быть описаны в полной или сокращенной форме:

Таблица. Свойства, описывающие стиль вывода каждой фигуры, общие для всех фигур

Характеристика	Описание
<code>facecolor="цвет"</code> или <code>fc="цвет"</code>	Цвет заливки фигуры
<code>linewidth=значение</code> или <code>lw=значение</code>	Толщина границы фигуры
<code>edgecolor="цвет"</code> или <code>ec="цвет"</code>	Цвет границы фигуры
<code>fill=значение</code>	Фигура закрашенная ( <code>True</code> , по умолчанию) или незакрашенная( <code>False</code> )

Ввод [7]:

```
import matplotlib.pyplot as plt
from matplotlib.patches import Circle, Rectangle

%matplotlib inline

fig = plt.figure()
plt.xlim(0, 8)
plt.ylim(0, 8)

ax = plt.gca()

rect=Rectangle((1, 2), 4, 5, facecolor='cyan', fill=True)
ax.add_patch(rect) # добавление прямоугольника на графическую область
circle = Circle((6, 5), 2, facecolor='r', linestyle='--', edgecolor='green', linewidth=2)
ax.add_patch(circle) # добавление круга на графическую область

plt.show()
```

