

Руководство по Matplotlib

Выполнили студентки 5 курса гр.5181: Аникина Анастасия Игоревна и Кисеева Виктория Ильинична

Matplotlib - это основная библиотека для построения научных графиков в Python. [1] Включает функции для создания высококачественных визуализаций: линейных диаграмм, гистограмм и т.д. Визуализация данных и результатов - цель использования библиотеки matplotlib. При работе в среде можно вывести рисунок на экран с помощью встроенных команд:

- `%matplotlib notebook` для визуализации графика в интерактивном режиме;
- `%matplotlib inline` для получения статичного изображения.

[1] Андреас Мюллер, Сара Гвидо. Введение в машинное обучение с помощью Python.

Создание рисунка в matplotlib схоже с рисованием в реальной жизни. Так художнику нужно взять основу (холст или бумагу), инструменты (кисти или карандаши), иметь представление о будущем рисунке (что именно он будет рисовать) и, наконец, выполнить всё это и нарисовать рисунок деталь за деталью. Создание основы и процесс отображения рисунка - работа для matplotlib.

Matplotlib организована иерархически. Наиболее простыми для понимания являются высокоуровневые функции. Поэтому знакомство с matplotlib обычно начинают с самого высокоуровневого интерфейса `matplotlib.pyplot`.

Например, чтобы нарисовать гистограмму, нужно вызвать всего одну команду: `matplotlib.pyplot.hist(arr)`. Гистограмма состоит из повторяющихся по форме фигур - прямоугольников. Чтобы нарисовать прямоугольник, нужно знать координату одного угла и ширину/длину. Прямоугольник мы бы рисовали линиями, соединяя угловые точки. Этот пример отображает иерархичность рисунков, когда итоговая диаграмма (высокий уровень) состоит из простых геометрических фигур (более низкий, средний уровень), созданных несколькими универсальными методами рисования (низкий уровень).

Изначально matplotlib планировался как свободная альтернатива MATLAB, где в одной среде имелись бы средства как для рисования, так и для численного анализа.

Иерархическая структура рисунка

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

1. Рисунок (Figure)

Любой рисунок в `matplotlib` имеет вложенную структуру. Рисунок - это объект самого верхнего уровня, на котором располагаются:

- области рисования (Axes);
- элементы рисунка Artists (заголовки, легенда и т.д.);
- основа-холст (Canvas).

На рисунке может быть несколько областей рисования Axes, но данная область рисования Axes может принадлежать только одному рисунку Figure.

2. Область рисования (Axes)

Объект среднего уровня. 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.

3. Координатная ось (Axis)

Координатная ось является объектом среднего уровня, которая определяет область изменения данных. На них наносятся:

- деления ticks;
- 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.

4. Элементы рисунка (Artists)

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.

Когда происходит отображение рисунка (figure rendering), все элементы рисунка Artists наносятся на основу-холст (Canvas). Большая часть из них связывается с областью рисования Axes. Также элемент рисунка не может совместно использоваться несколькими областями Axes или быть перемещён с одной на другую.

Pyplot

Pyplot - интерфейс для построения графиков простых функций. Позволяет пользователю сосредоточиться на выборе готовых решений и настройке базовых параметров рисунка. Это его главное достоинство, поэтому изучение matplotlib лучше всего начинать именно с интерфейса pyplot.

Существует стандарт вызова pyplot в python:

```
In [ ]: ▶ import matplotlib.pyplot as plt
```

Рисунки в matplotlib создаются путём последовательного вызова команд. Графические элементы (точки, линии, фигуры и т.д.) наслаиваются одна на другую последовательно. При этом последующие перекрывают предыдущие, если они занимают общие участки на рисунке (регулируется параметром `zorder`).

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. Так как главным объектом в matplotlib является рисунок `Figure`, создание научной графики нужно начинать именно с создания рисунка. Создать рисунок `figure` позволяет метод `plt.figure()`.

После вызова любой графической команды (функции), которая создаёт какой-либо графический объект, например, `plt.scatter()` или `plt.plot()`, всегда существует хотя бы одна область для рисования (по умолчанию прямоугольной формы). Чтобы текущее состояние рисунка отразилось на экране, можно воспользоваться командой `plt.show()`. Будут показаны все рисунки (`figures`), которые были созданы.

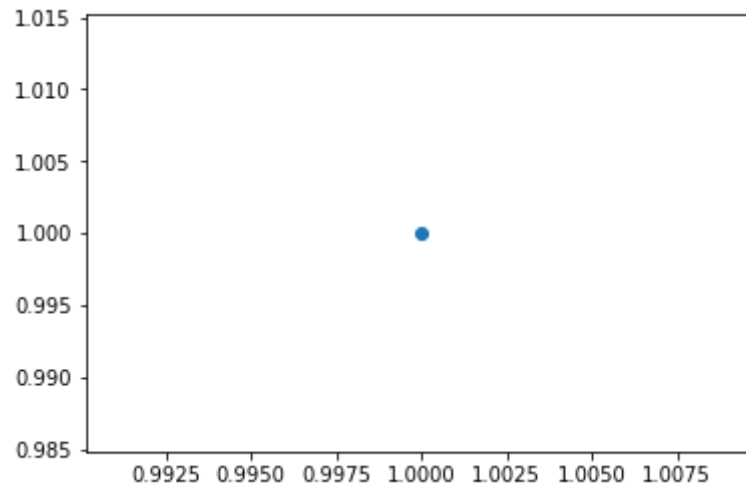
In [2]: `import matplotlib.pyplot as plt`

```
# Создание объекта Figure
fig = plt.figure()
# тип объекта Figure
print (type(fig))
# scatter - метод для нанесения маркера в точке (1.0, 1.0)
plt.scatter(1.0, 1.0)

print (fig.axes)
# После нанесения графического элемента в виде маркера, список текущих областей состоит из одной области

plt.show()
```

```
<class 'matplotlib.figure.Figure'>
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fe16021d860>]
```



Обычно рисунок в matplotlib представляет собой прямоугольную область, заданную в относительных координатах: от 0 до 1 включительно по обеим осям. Вторым распространённым вариантом типа рисунка - круглая область (polar plot).

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

```
plt.savefig('{}.{:}'.format(name, fmt), fmt='png').
```

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

Текст

Текст является одним из базовых графических элементов рисунка в matplotlib. Подписи координатных осей и их делений, заголовки, пояснительные подписи на графиках и диаграммах - это всё текст. В Matplotlib возможна поддержка кириллицы для создания научной графики с подписями на русском языке. Одним из весомых преимуществ matplotlib при работе с текстом является простая поддержка математических формул с помощью LaTeX.

Работа с текстом

Matplotlib имеет обширную текстовую поддержку, включая поддержку математических выражений, поддержку truetype для растровых и векторных выходных данных, разделенный новой строкой текст с произвольными поворотами и поддержкой юникода. Matplotlib также осуществляет поддержку FreeType, что обеспечивает создание красивых, сглаженных шрифтов. matplotlib включает в себя собственный matplotlib.font_manager, который реализует кросс-платформенный, W3C-совместимый алгоритм поиска шрифтов.

Matplotlib встраивает шрифты непосредственно в выходные документы, поэтому то, что вы видите на экране, - это то, что вы получаете в печатном виде. Matplotlib предоставляет пользователю большой контроль над свойствами текста (размер шрифта, вес шрифта, расположение и цвет текста и т. д.)

Файл настройки matplotlibrc

matplotlibrc - файл настройки в котором хранятся значения по умолчанию для разных свойств графических элементов. Он инициализируется при каждой загрузке модуля matplotlib. Изменив содержание файла matplotlibrc можно сохранить пользовательские настройки для работы при следующих загрузках модуля matplotlib.


matplotlibrc представляет собой текстовый файл, каждая строка которого описывает параметры в виде:

параметр : значение

При этом некоторые параметры имеют вложенную структуру:

параметр.подпараметр1 : значение параметр.подпараметр2 : значение

Файл matplotlibrc имеет стандартный вид, ниже представлен пример организации файла:

```
In [ ]:  #...  
##### CONFIGURATION BEGINS HERE  
  
# The default backend; one of GTK GTKAgg GTKCairo GTK3Agg GTK3Cairo  
# MacOSX Qt4Agg Qt5Agg TkAgg WX WXAgg Agg Cairo GDK PS PDF SVG  
# Template.  
# You can also deploy your own backend outside of matplotlib by  
# referring to the module name (which must be in the PYTHONPATH) as  
# 'module://my_backend'.  
#backend      : TkAgg  
  
# If you are using the Qt4Agg backend, you can choose here  
# to use the PyQt4 bindings or the newer PySide bindings to  
# the underlying Qt4 toolkit.  
#backend.qt4 : PyQt4      # PyQt4 | PySide  
  
# Note that this can be overridden by the environment variable  
# QT_API used by Enthought Tool Suite (ETS); valid values are  
# "pyqt" and "pyside". The "pyqt" setting has the side effect of  
# forcing the use of Version 2 API for QString and QVariant.  
  
# The port to use for the web server in the WebAgg backend.  
# webagg.port : 8888  
  
# If webagg.port is unavailable, a number of other random ports will  
# be tried until one that is available is found.  
# webagg.port_retries : 50  
  
# When True, open the webbrowser to the plot that is shown  
# webagg.open_in_browser : True  
  
# When True, the figures rendered in the nbagg backend are created with  
# a transparent background.  
# nbagg.transparent : False  
  
# if you are running pyplot inside a GUI and your backend choice  
# conflicts, we will automatically try to find a compatible one for  
# you if backend_fallback is True  
#backend_fallback: True  
  
#interactive : False
```

```
#toolbar      : toolbar2      # None | toolbar2 ("classic" is deprecated)
#timezone     : UTC           # a pytz timezone string, e.g., US/Central or Europe/Paris

# Where your matplotlib data lives if you installed to a non-default
# location. This is where the matplotlib fonts, bitmaps, etc reside
#datapath : /home/jdhunter/mpldata

### LINES
# See http://matplotlib.org/api/artist\_api.html#module-matplotlib.lines for more
# information on line properties.
#lines.linewidth      : 1.5      # line width in points
#lines.linestyle       : -        # solid line
#lines.color           : C0       # has no affect on plot(); see axes.prop_cycle
#lines.marker          : None     # the default marker
#lines.markeredgewidth : 1.0      # the line width around the marker symbol
#lines.markersize      : 6        # markersize, in points
#lines.dash_joinstyle  : miter     # miter/round/bevel
#lines.dash_capstyle   : butt      # butt/round/projecting
#lines.solid_joinstyle : miter     # miter/round/bevel
#lines.solid_capstyle  : projecting # butt/round/projecting
#lines.antialiased     : True      # render lines in antialiased (no jaggies)

# The three standard dash patterns. These are scaled by the linewidth.
#lines.dashed_pattern : 2.8, 1.2
#lines.dashdot_pattern : 4.8, 1.2, 0.8, 1.2
#lines.dotted_pattern : 1.1, 1.1
#lines.scale_dashes   : True

#markers.fillstyle: full # full/left/right/bottom/top/none

### PATCHES
# Patches are graphical objects that fill 2D space, like polygons or
# circles. See
# http://matplotlib.org/api/artist\_api.html#module-matplotlib.patches
# information on patch properties
#patch.linewidth      : 1        # edge width in points.
#patch.facecolor       : C0
#patch.edgewidth       : black   # if forced, or patch is not filled
#patch.force_edgewidth : False   # True to always use edgewidth
#patch.antialiased     : True     # render patches in antialiased (no jaggies)
```



```
### HATCHES
#hatch.color      : k
#hatch.linewidth  : 1.0

### Boxplot
#boxplot.notch      : False
#boxplot.vertical   : True
#boxplot.whiskers   : 1.5
#boxplot.bootstrap : None
#boxplot.patchartist : False
#boxplot.showmeans  : False
#boxplot.showcaps   : True
#boxplot.showbox    : True
#boxplot.showfliers : True
#boxplot.meanline   : False
#...
```

Чтобы отобразить откуда был загружен используемый файл matplotlibrc, используйте `matplotlib_fname()`:

```
In [3]: ► import matplotlib as plt
        plt.matplotlib_fname()
```

```
Out[3]: '/opt/anaconda3/envs/env_tf/lib/python3.6/site-packages/matplotlib/mpl-data/matplotlibrc'
```

Чтобы просмотреть текущие настройки введите следующее:

In [5]: ▶ plt.rcParams

```
/opt/anaconda3/envs/env_tf/lib/python3.6/site-packages/matplotlib/__init__.py:886: MatplotlibDeprecationWarning:
examples.directory is deprecated; in the future, examples will be found relative to the 'datapath' directory.
  "found relative to the 'datapath' directory.".format(key))
```

Шрифты. Стили и форматы

В настройках `matplotlibrc` или `rcParams` существует такой параметр как `fonts`, то есть шрифты. Всего существует 5 наборов шрифтов:

`cursive`;

`fantasy`;

`monospace`;

`sans-serif`;

`serif`.

Один из этих пяти наборов является текущим. За это отвечает параметр `font.family`. Каждый набор может состоять из одного или более шрифтов. Данная настройка определяет шрифт для всех подписей и текста на рисунке. Если конкретную подпись необходимо сделать другим шрифтом, можно указать шрифт из текущего стиля прямо в команде, передав в качестве соответствующего параметра словарь:

```
{'fontname':'название_шрифта'}.
```

Помимо семейств, текст также может иметь стиль. Атрибут стиля `style` может быть либо `'italic'`, либо `'oblique'`, либо `'normal'` (по умолчанию). Толщина или "жирность" шрифта, может быть задана через атрибут `fontweight`, который принимает значения `'bold'`, `'light'` или `'normal'` (по умолчанию). Стили и форматы можно комбинировать.

```

In [7]: ► import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

mpl.rcParams['font.fantasy'] = 'Arial', 'Times New Roman', 'Tahoma', 'Comic Sans MS', 'Courier'
mpl.rcParams['font.family'] = 'fantasy'

# Текущий стиль-семейство шрифтов
cfam = mpl.rcParams.get('font.family')[0]
print('cfam %s' % cfam)
cfont = mpl.rcParams.get('font.fantasy')[0]

# Первый шрифт в текущем семействе
print(mpl.rcParams.get('font.%s' % cfam))

N = 100
x = np.arange(N)
# Задаём выборку из Гамма-распределения с параметрами формы=1. и масштаба=3.0
y = np.random.gamma(1.0, 3.0, N)

fig = plt.figure()
cc = plt.hist(y)

text_style = ['italic', 'oblique', 'normal']
font_weights = ['bold', 'light', 'normal']
for i, ts in enumerate(text_style):
    plt.text(6, 20-5*i, '%s text style' % ts, {'fontname':'Courier'}, style=ts, fontsize=14)
    plt.text(6, 35-4*i, '%s & %s text style' % (ts, font_weights[i]), {'fontname':'Courier'},
             style=ts, fontweight=font_weights[i], fontsize=12)

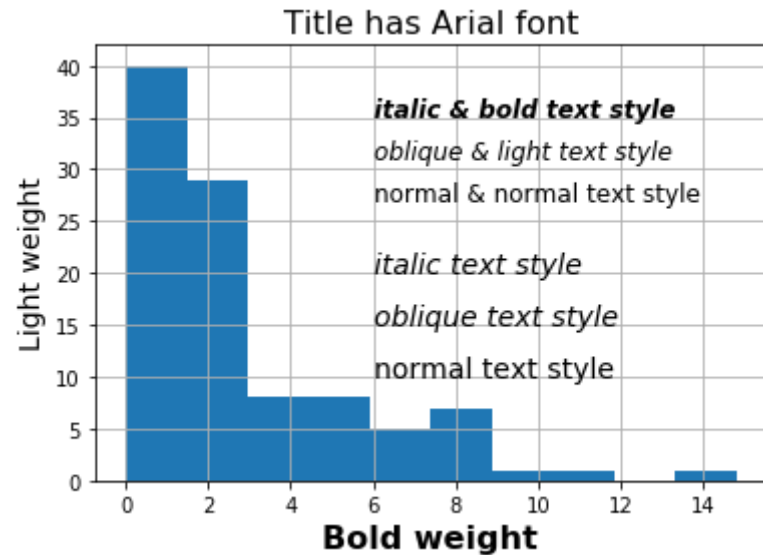
plt.title('Title has %s font' % cfont, fontweight='normal', color='k', fontsize=16)
plt.xlabel('Bold weight', {'fontname':'Times New Roman'}, fontweight='bold', fontsize=16)
plt.ylabel('Light weight', {'fontname':'Times New Roman'}, fontweight='light', fontsize=14)
plt.grid(True)

plt.show()

```

cfam fantasy

```
['Arial', 'Times New Roman', 'Tahoma', 'Comic Sans MS', 'Courier']
```



Поддержка LaTeX

Для работы с LaTeX необходимо установить его дистрибутив

В matplotlib осуществлена безбарьерная работа с LaTeX. Нужно, чтобы в системе была установлена библиотека LaTeX. Код LaTeX оформляется в виде "raw" строки (перед строкой ставится символ `r` -> `r'строка'`). Также в настройках pyplot нужно указать, чтобы текст отрисовывался с учётом синтаксиса LaTeX. Это можно сделать через `plt.rc('text', usetex=True)`

Текст на рисунке

Одними из самых базовых графических команд являются команды, отображающие текст. Такой командой, не привязанной к какому-либо объекту вроде координатной оси или делений координатной оси, является команда `plt.text()`.

В качестве входных данных она принимает координаты положения будущей строки и сам текст в виде строки. По умолчанию координаты положения строки будут приурочены к области изменения данных. Можно задать положение текста в относительных координатах, когда вся область рисования изменяется по обеим координатным осям от 0 до 1 включительно.

Таким образом, координата (0.5, 0.5) в относительных координатах означает центр области рисования. Текст можно выровнять с помощью параметров **horizontalalignment** и **verticalalignment**, а также заключать его в рамку с цветным фоном, передав параметр **bbox**. Bbox - это словарь, работающий со свойствами прямоугольника (объектом Rectangle).

Помимо метода `text()` существуют и другие методы отображения текста в `ruplot`. Ниже представлен список текстовых команд в `ruplot`.

- `plt.xlabel()` - добавляет подпись оси абсцисс ОХ;
- `plt.ylabel()` - добавляет подпись оси ординат ОУ;
- `plt.title()` - добавляет заголовок для области рисования Axes;
- `plt.figtext()` - добавляет текст на рисунок Figure;
- `plt.suptitle()` - добавляет заголовок для рисунка Figure;
- `plt.annotate()` - добавляет примечание, которое состоит из текста и необязательной стрелки в указанную область на рисунке.

Каждый текст на рисунке - это экземпляр либо исходного класса, либо дочернего класса для `matplotlib.text.Text`.

Рассмотрим представленные команды на простейшем графике $y=x^2$:

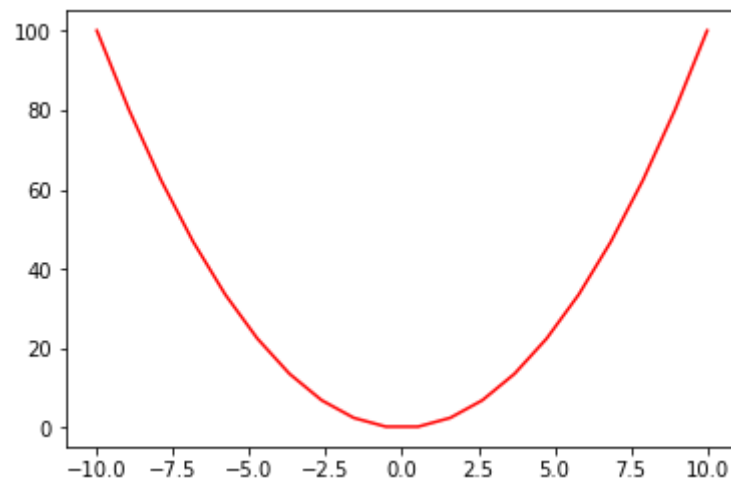
Зададим точки. Абсциссы определяются в диапазоне (-10.,10.), таким образом, чтобы всего получилось 20 точек. Ординаты рассчитываются через уравнение $y=x^2$

```
In [8]:  ▶ import matplotlib.pyplot as plt
import numpy as np

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

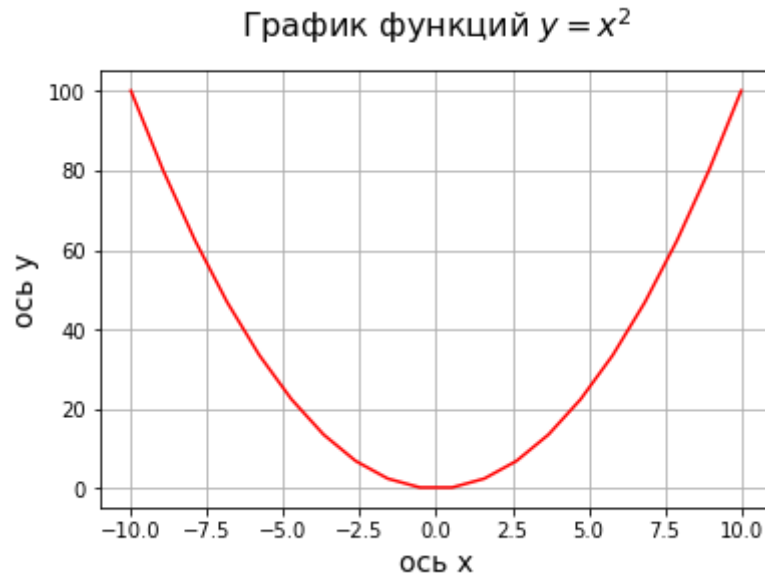
Изобразим на графике.Ниже представлен простейший график, без редактирования и работы с текстом.

```
In [9]: ▶ plt.plot(x, y, color='red', label='Линия 1');
```



Нанесем на график сетку, добавим подписи к осям и заголовок.

```
In [10]: ▶ plt.plot(x, y, color='red', label='Линия 1');  
plt.grid() # сетка  
plt.xlabel('ось x', fontsize=14) # добавляем подпись к оси абцисс "ось x"  
plt.ylabel('ось y', fontsize=14) # добавляем подпись к оси ординат "ось y"  
plt.title(r'График функций  $y = x^2$ ', fontsize=16, y=1.05); # добавляем заголовок к графику "График функции  $y=x^2$ "
```

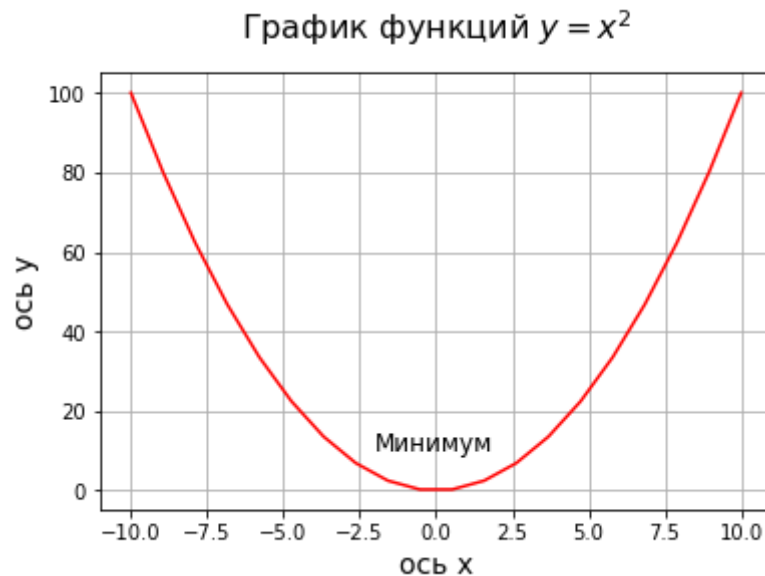


Также matplotlib позволяет нанести на график подписи с помощью функций `text()` и `annotate()`.

Функция `text()` позволяет нанести подпись в любом месте изображения, для этого указываются координаты, где должен располагаться текст.

Добавим подпись к точке экстремума с помощью функции `text()`:

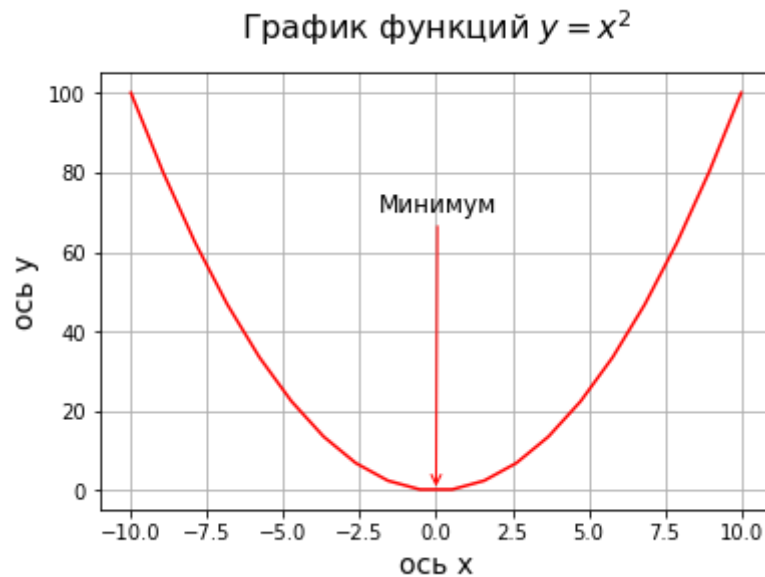

```
In [11]: ▶ plt.plot(x, y, color='red', label='Линия 1');  
plt.grid() # сетка  
plt.xlabel('ось x', fontsize=14) # добавляем подпись к оси абцисс "ось x"  
plt.ylabel('ось y', fontsize=14) # добавляем подпись к оси ординат "ось y"  
plt.title(r'График функций  $y = x^2$ ', fontsize=16, y=1.05); # добавляем заголовок к графику "График функции  $y=x^2$ "  
  
plt.text(-2., 10., 'Минимум', fontsize=12); # добавляем подпись к графику в точке (-2.5, 10)
```



Функция `annotate()` позволяет нанести подпись с необязательной стрелкой. Параметр `xy` определяет координаты точки, которую необходимо подписать (конечная точка стрелки), параметр `xytext` определяет координаты точки, в которой будет располагаться текст.

```
In [12]: ▶ plt.plot(x, y, color='red', label='Линия 1');
plt.grid() # сетка
plt.xlabel('ось x', fontsize=14) # добавляем подпись к оси абцисс "ось x"
plt.ylabel('ось y', fontsize=14) # добавляем подпись к оси ординат "ось y"
plt.title(r'График функций $y = x^2$, fontsize=16, y=1.05); # добавляем заголовок к графику "График функции y=x^2"

plt.annotate("Минимум", xy=(0.,0.), xytext=(-1.9,70.),fontsize=12, arrowprops = dict(arrowstyle = '->',color = 'red')
```



Расположение текста на графике

```
In [13]: ▶ import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

a = 1.
x = np.arange(0., 2*np.pi, 0.1)
```

Текст в координатах данных

```
In [14]:  xz = a*(np.sin(x) - np.cos(2*x))

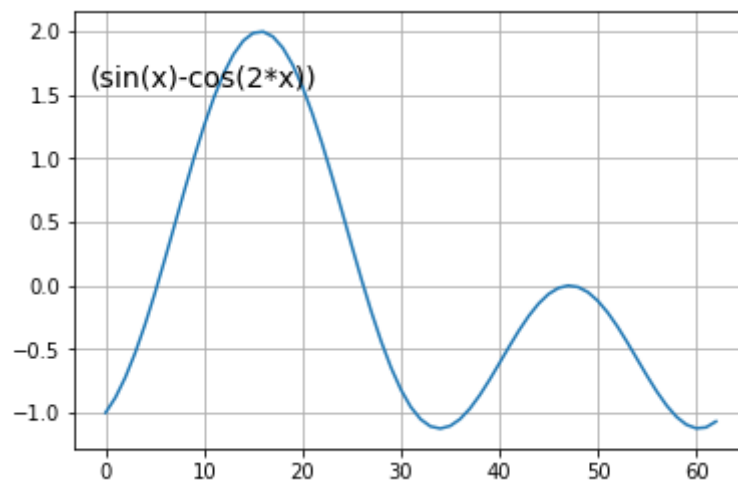
fig = plt.figure()
plt.plot(xz)

# выравнивание текста по левому краю
str1 = plt.text(-np.pi/2., np.pi/2., '(sin(x)-cos(2*x))', fontsize=14)

print('Text class: %s' % str1.__class__)

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

Text class: <class 'matplotlib.text.Text'>

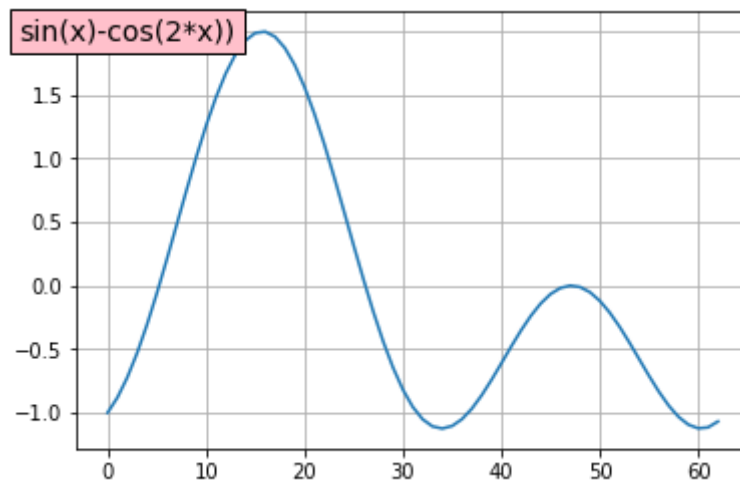


Текст в рамке

```
In [15]: ➤ xz = a*(np.sin(x) - np.cos(2*x))

fig = plt.figure()
plt.plot(xz)

plt.text(2., 2., 'sin(x)-cos(2*x))', fontsize=14,
        # выравнивание по вертикали и по горизонтали
        horizontalalignment='center', verticalalignment='center',
        bbox=dict(facecolor='pink', alpha=1.0))
plt.grid()
plt.show()
```

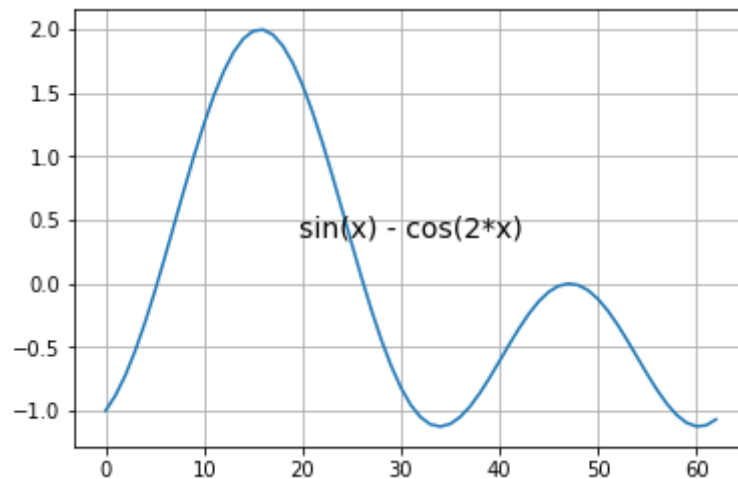


Текст в относительных координатах области рисования ax

```
In [17]: ➤ xz = a*(np.sin(x) - np.cos(2*x))

fig = plt.figure()
plt.plot(xz)

# область рисования ax
ax = fig.add_subplot(111)
plt.text(0.5, 0.5, 'sin(x) - cos(2*x)', fontsize=14,
        horizontalalignment='center', verticalalignment='center',
        transform=ax.transAxes)
plt.grid()
plt.show()
```



Построение графиков с помощью matplotlib

Подключение библиотеки

```
In [ ]: ➤ import matplotlib.pyplot as plt
# %matplotlib inline
```

Основные графические команды

Графические команды - это функции, которые, принимая некоторые параметры, возвращают какой-то графический результат. Это может быть текст, линия, график, диаграмма и др. Рассмотрим графические команды, которые создают графику высокого уровня: графики или диаграммы.

В Matplotlib заложены как простые графические команды, так и достаточно сложные. Доступ к ним через `pyplot` означает использование синтаксиса вида `"plt.название_команды()"`.

Наиболее распространённые команды для создания научной графики в `matplotlib`:

1. Самые простые графические команды

- `plt.scatter()` - маркер или точечное рисование;
- `plt.plot()` - ломаная линия;
- `plt.text()` - нанесение текста.

2. Диаграммы

- `plt.bar()`, `plt.barh()`, `plt.barbs()`, `broken_barh()` - столбчатая диаграмма;
- `plt.hist()`, `plt.hist2d()`, `plt.hlines` - гистограмма;
- `plt.pie()` - круговая диаграмма;
- `plt.boxplot()` - "ящик с усами" (`boxwhisker`);
- `plt.errorbar()` - оценка погрешности, "усы".

3. Изображения в изолиниях

- `plt.contour()` - изолинии;
- `plt.contourf()` - изолинии с послойной окраской.

4. Отображения

- `plt.pcolor()`, `plt.pcolormesh()` - псевдоцветное изображение матрицы (2D массива);
- `plt.imshow()` - вставка графики (пиксели + сглаживание);
- `plt.matshow()` - отображение данных в виде квадратов.

5. Заливка

- `plt.fill()` - заливка многоугольника;

- `plt.fill_between()`, `plt.fill_betweenx()` - заливка между двумя линиями.

6. Векторные диаграммы

- `plt.streamplot()` - линии тока;
- `plt.quiver()` - векторное поле.

В списке нет команд для рисования различных геометрических фигур (круги, многоугольники и т.д.). Это связано с тем, что в `matplotlib` они вызываются через **`matplotlib.patches`**.

Ниже показаны примеры графики, которую `matplotlib` создаёт "по умолчанию" при вызове той или иной графической команды.

1. Простые графические команды

```
In [18]: ▶ import matplotlib.pyplot as plt

fig = plt.figure()
```

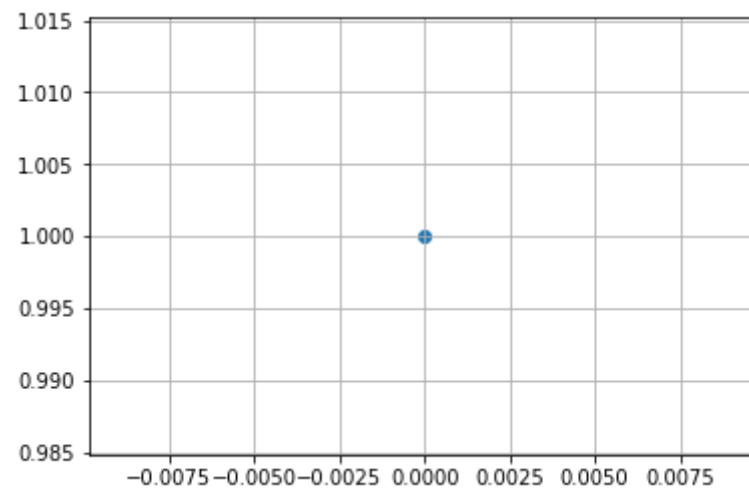
<Figure size 432x288 with 0 Axes>

Добавление на рисунок прямоугольной (по умолчанию) области рисования

```
In [19]: scatter1 = plt.scatter(0.0, 1.0)
print('Scatter: ', type(scatter1))

grid1 = plt.grid(True) # линии вспомогательной сетки
plt.show()
```

Scatter: <class 'matplotlib.collections.PathCollection'>



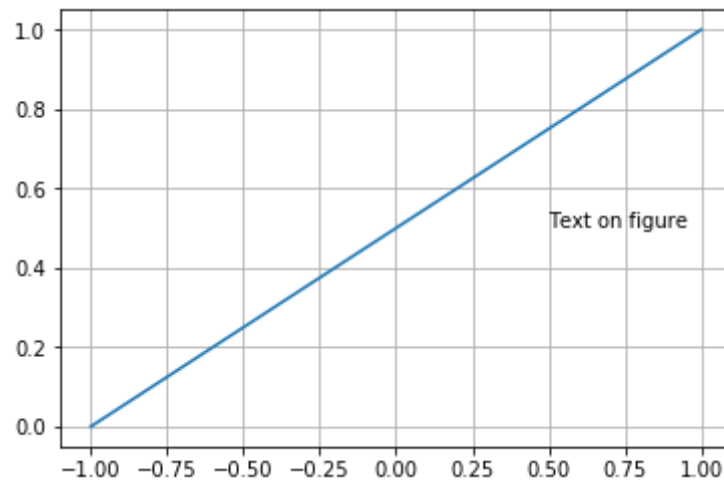

```
In [20]: ▶ graph1 = plt.plot([-1.0, 1.0], [0.0, 1.0])
print('Plot: ', len(graph1), graph1)

text1 = plt.text(0.5, 0.5, 'Text on figure')
print('Text: ', type(text1))

grid1 = plt.grid(True) # линии вспомогательной сетки
plt.show()
```

Plot: 1 [<matplotlib.lines.Line2D object at 0x7fe15fdc5eb8>]

Text: <class 'matplotlib.text.Text'>



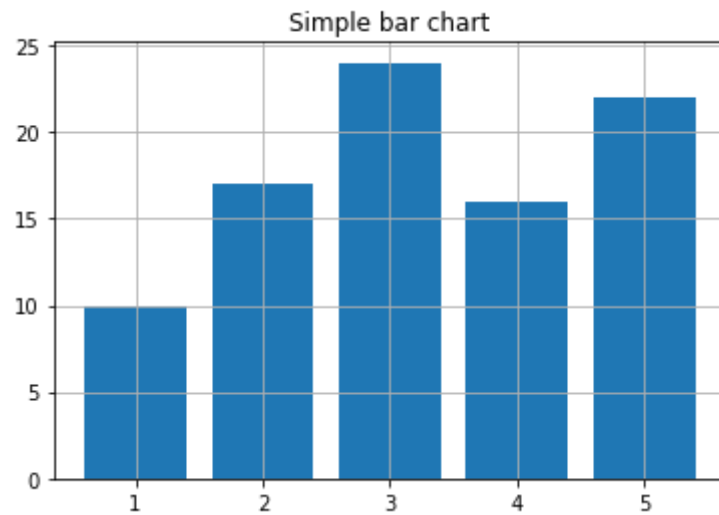
2. Диаграммы

```
In [21]: ▶ import matplotlib.pyplot as plt
import numpy as np

s = ['one', 'two', 'three ', 'four' , 'five']
x = [1, 2, 3, 4, 5]
z = np.random.random(100)
z1 = [10, 17, 24, 16, 22]
z2 = [12, 14, 21, 13, 17]
```

Столбчатая диаграмма

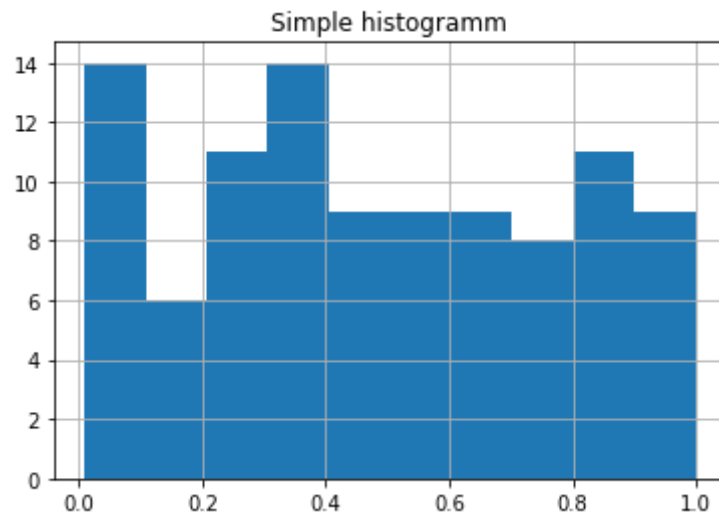
```
In [22]: ▶ fig = plt.figure()  
plt.bar(x, z1)  
plt.title('Simple bar chart')  
  
plt.grid(True)  
plt.show()
```



Гистограмма

```
In [23]: ▶ fig = plt.figure()
plt.hist(z)
plt.title('Simple histogramm')

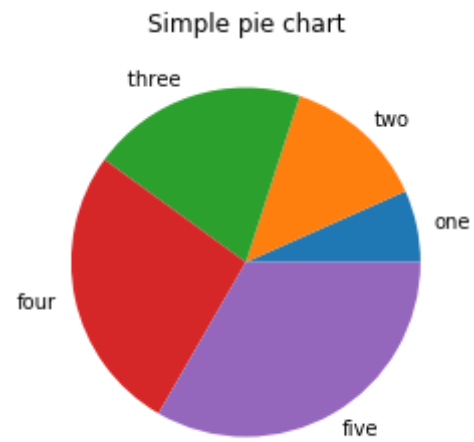
plt.grid(True)
plt.show()
```



Круговая диаграмма

```
In [24]: ▶ fig = plt.figure()
plt.pie(x, labels=s)

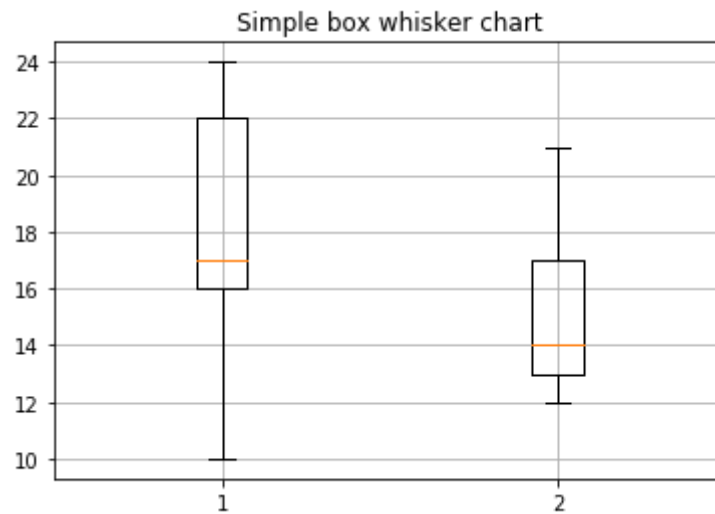
plt.title('Simple pie chart')
plt.show()
```



"Ящик с усами" (boxwhisker)

```
In [25]: ▶ fig = plt.figure()
plt.boxplot([z1, z2])
plt.title('Simple box whisker chart')

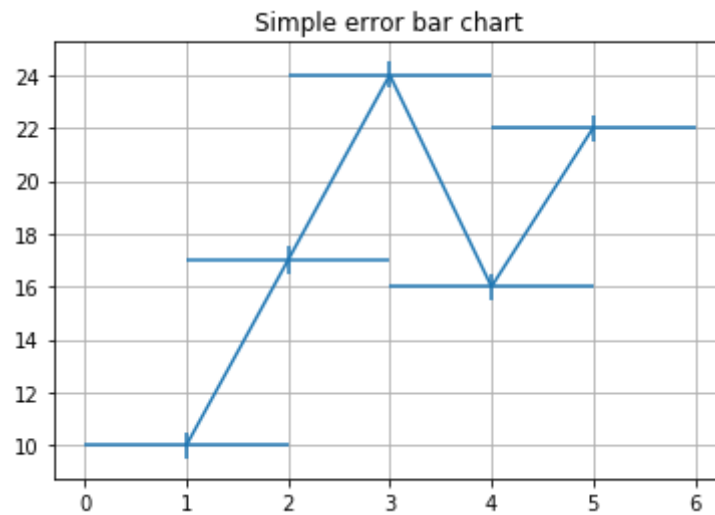
plt.grid(True)
plt.show()
```



Оценка погрешности, "усы"

```
In [26]: ▶ fig = plt.figure()
plt.errorbar(x, z1, xerr=1, yerr=0.5)
plt.title('Simple error bar chart')

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



3. Изображения в изолиниях

```
In [27]: ▶ import matplotlib.pyplot as plt
import numpy as np

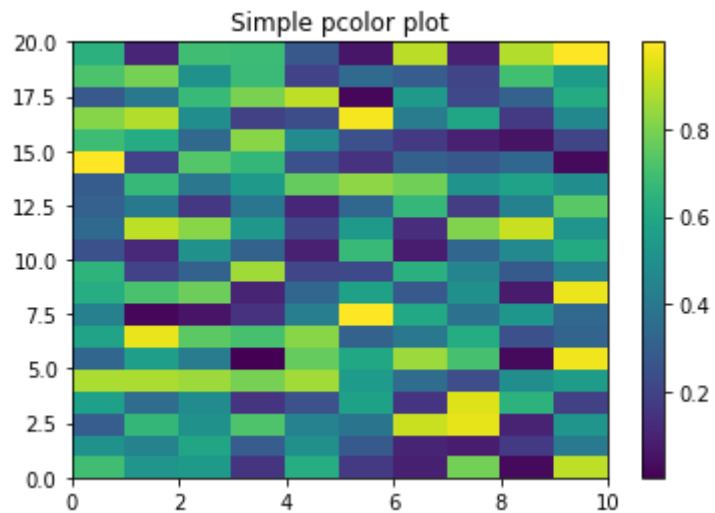
# создаём матрицу значений
dat = np.random.random(200).reshape(20,10)

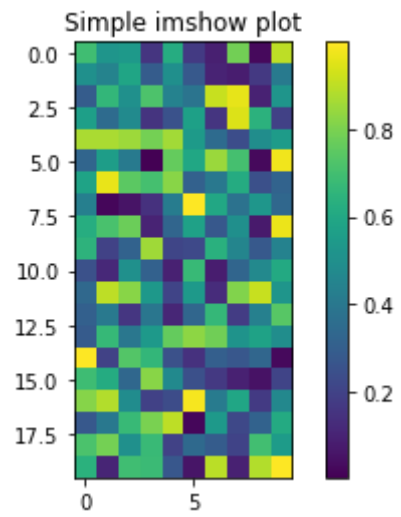
fig = plt.figure()

# метод псевдографики pcolor
pc = plt.pcolor(dat)
plt.colorbar(pc)
plt.title('Simple pcolor plot')

fig = plt.figure()
me = plt.imshow(dat)
plt.colorbar(me)
plt.title('Simple imshow plot')

plt.show()
```





4. Методы отображения


```
In [28]: ▶ import matplotlib.pyplot as plt
import numpy as np

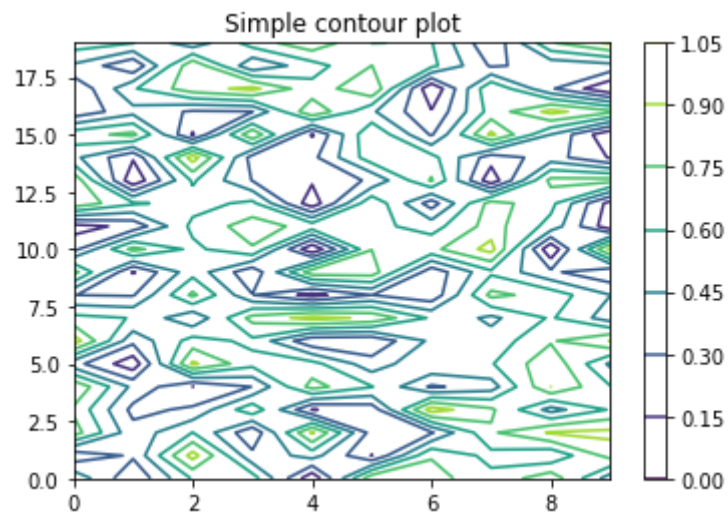
# матрица значений
dat = np.random.random(200).reshape(20,10)

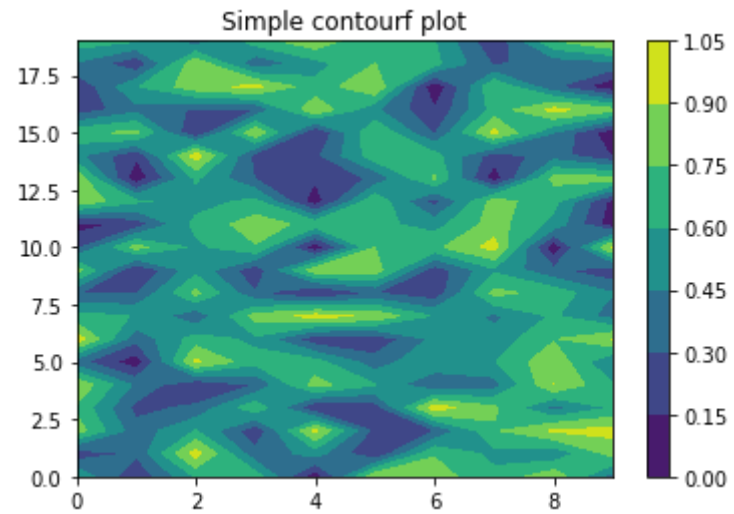
fig = plt.figure()
cr = plt.contour(dat)
plt.colorbar(cr)
plt.title('Simple contour plot')

fig = plt.figure()
cf = plt.contourf(dat)
plt.colorbar(cf)
plt.title('Simple contourf plot')

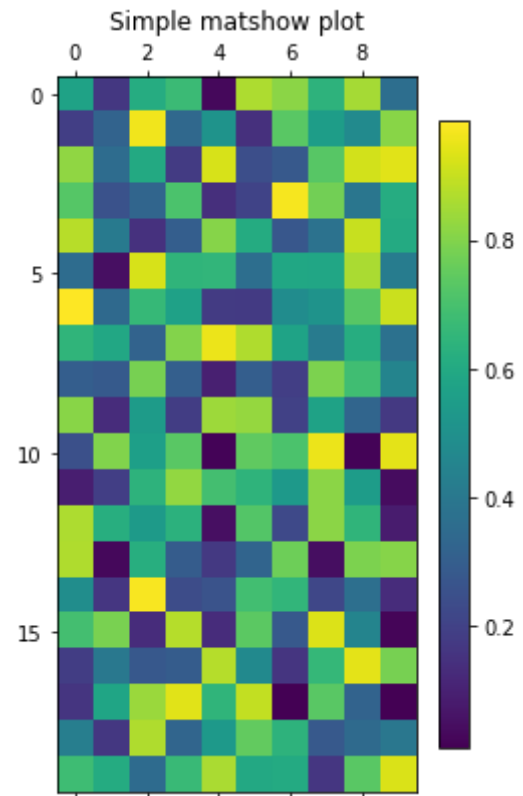
fig = plt.figure()
cf = plt.matshow(dat)
plt.colorbar(cf, shrink=0.7)
plt.title('Simple matshow plot')

plt.show()
```





<Figure size 432x288 with 0 Axes>



5. Методы заливки

```
In [29]: ▶ import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 4*np.pi+0.1, 0.1)
y = np.sin(x)
z = np.sin(2*x)

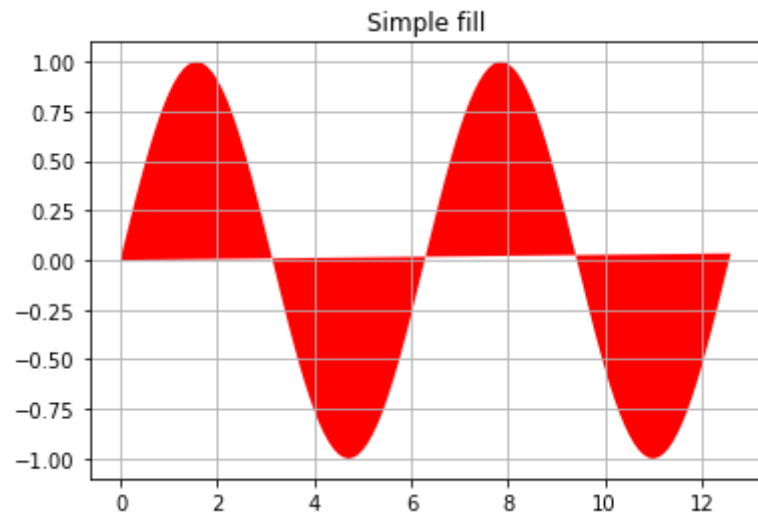
x2 = np.arange(20)
y2 = -1.5*x2 + 2.33
z2 = 0.7*x2 - 8.5
```

Заливка многоугольника

```
In [30]: ▶ fig = plt.figure()

# метод псевдографики pcolor
plt.fill(x, y, 'r')
plt.title('Simple fill')

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



Заливка между двумя линиями

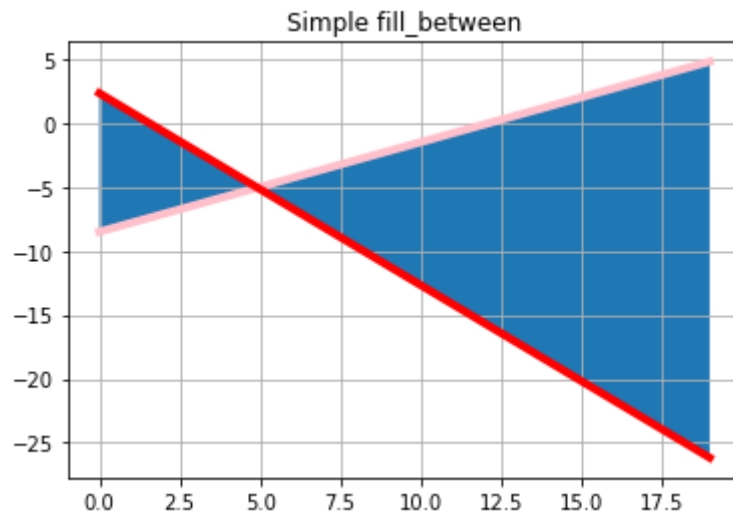
```
In [31]: ▶ # fill_between()
fig = plt.figure()
plt.plot(x2, z2, color='pink', linewidth=4.0)
plt.plot(x2, y2, color='r', linewidth=4.0)
plt.fill_between(x2, y2, z2)
plt.title('Simple fill_between')

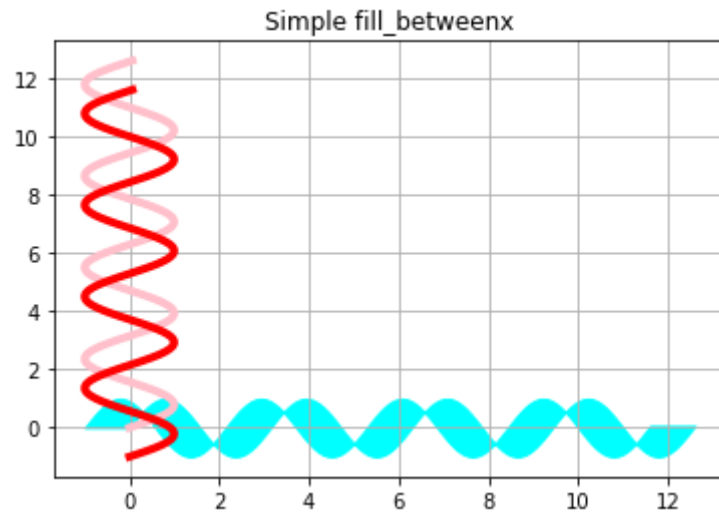
plt.grid(True)

# fill_betweenx()
fig = plt.figure()
plt.plot(z, x, color='pink', linewidth=4.0)
plt.plot(z, x-1.0, color='r', linewidth=4.0)
plt.fill_betweenx(z, x, x-1.0, color='cyan')
plt.title('Simple fill_betweenx')

plt.grid(True)

plt.show()
```





6. Векторные диаграммы

```
In [32]: ▶ import matplotlib.pyplot as plt
import numpy as np

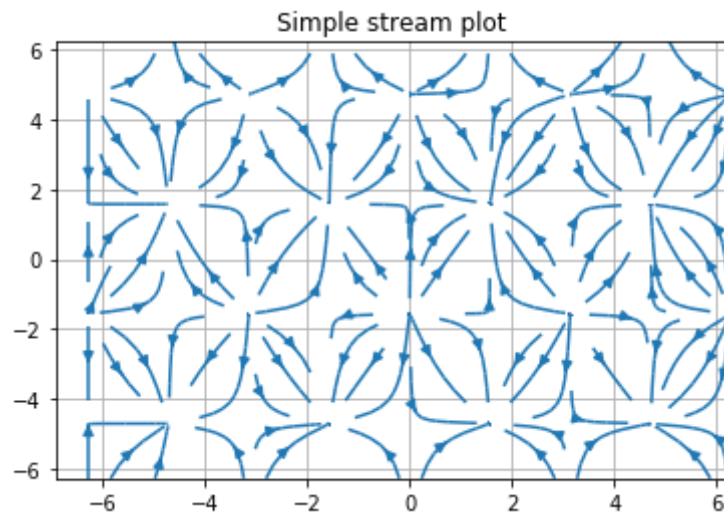
x = np.arange(-2*np.pi, 2*np.pi, 0.1)
u = np.sin(x)*np.cos(x)
v = np.cos(x)
uu, vv = np.meshgrid(u,v)

N = 100
x1 = np.random.random(N).reshape((10, 10))
y1 = np.random.random(N).reshape((10, 10))
```

Линии тока

```
In [33]: ▶ fig = plt.figure()
plt.streamplot(x, x, uu, vv)
plt.title('Simple stream plot')

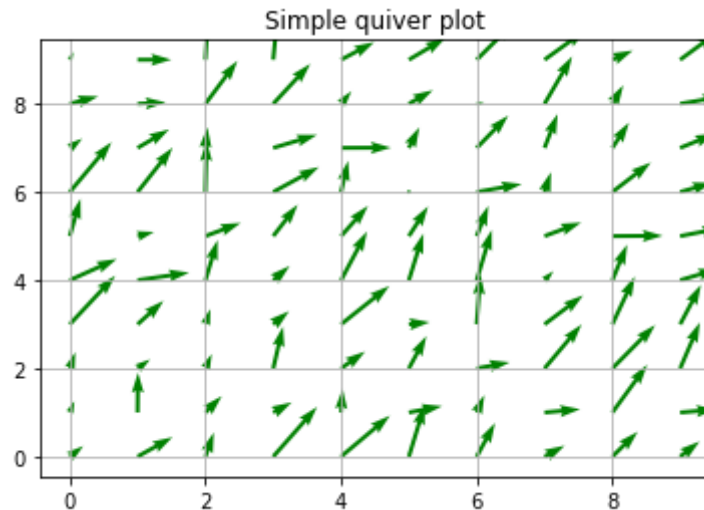
plt.grid(True)
plt.show()
```



Векторное поле

```
In [34]: ▶ fig = plt.figure()
plt.quiver(x1, y1, color='green')
plt.title('Simple quiver plot')

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



Многообразие и удобство создания графики в matplotlib обеспечивается за счёт созданных графических команд и богатого арсенала по конфигурации типовых форм. Эта настройка включает в себя работу с цветом, формой, типом линии или маркера, толщиной линий, степенью прозрачности элементов, размером и типом шрифта и другими свойствами.

Параметры, которые определяют эти свойства в различных графических командах, обычно имеют одинаковый синтаксис, то есть называются одинаково. Стандартным способом задания свойств какого либо создаваемого объекта (или методу) является передача по ключу: ключ = значение.

Наиболее часто встречаемые названия параметров изменения свойств графических объектов:

- color/colors/c - цвет;
- linewidth/linewidths - толщина линии;
- linestyle - тип линии;
- alpha - степень прозрачности (от полностью прозрачного 0 до непрозрачного 1);
- fontsize - размер шрифта;

- `marker` - тип маркера;
- `s` - размер маркера в методе `plt.scatter`(только цифры);
- `rotation` - поворот строки на X градусов.

Различные формы области рисования

In [35]: `import matplotlib.pyplot as plt`

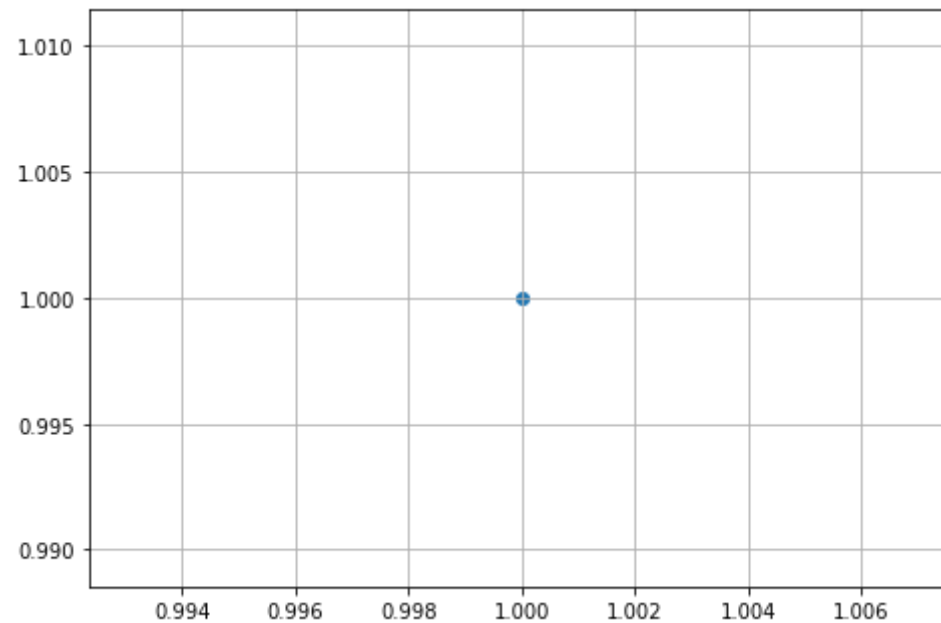
Добавление на рисунок прямоугольной (по умолчанию) области рисования

```
In [36]: ► fig = plt.figure()

ax = fig.add_axes([0, 0, 1, 1])
print (type(ax))
plt.scatter(1.0, 1.0)

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

<class 'matplotlib.axes._axes.Axes'>

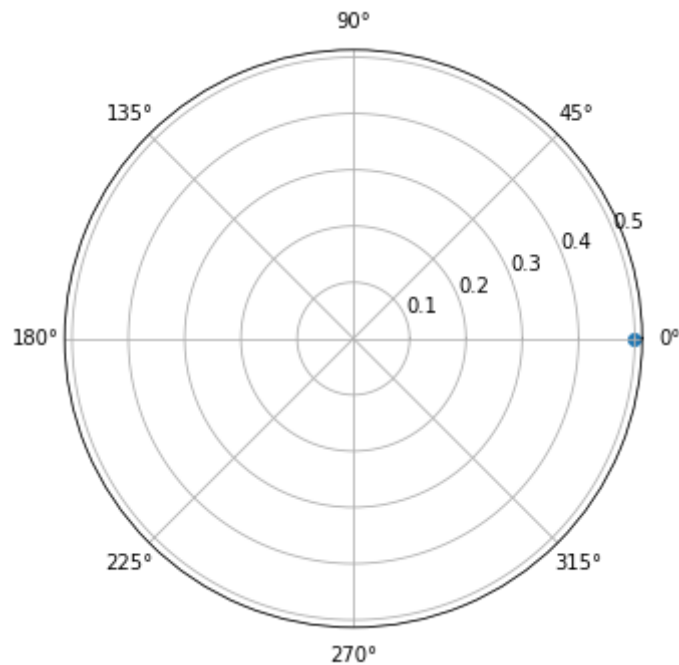


Добавление на рисунок круговой области рисования

```
In [37]: ▶ fig = plt.figure()

ax = fig.add_axes([0, 0, 1, 1], polar=True)
plt.scatter(0.0, 0.5)

plt.show()
```



Элементы рисунка Artists

Всё пространство рисунка Figure (прямоугольной или иной формы) можно использовать для нанесения других элементов рисунка, например, контейнеров Axes, графических примитивов в виде линий, фигур, текста и так далее. В любом случае каждый рисунок можно структурно представить следующим образом:

1. Область рисования Axes
 - Заголовок области рисования -> `plt.title();`
2. Ось абсцисс Xaxis
 - Подпись оси абсцисс OX -> `plt.xlabel();`

3. Ось ординат Yaxis
 - Подпись оси ординат OY -> `plt.ylabel();`
4. Легенда -> `plt.legend();`
5. Цветовая шкала -> `plt.colorbar()`
 - Подпись горизонтальной оси абсцисс OY -> `cbar.ax.set_xlabel();`
 - Подпись вертикальной оси абсцисс OY -> `cbar.ax.set_ylabel();`
6. Деления на оси абсцисс OX -> `plt.xticks();`
7. Деления на оси ординат OY -> `plt.yticks();`

Для каждого из перечисленных уровней-контейнеров есть возможность нанести заголовок (title) или подпись (label). Подписи к рисунку облегчают понимание того, в каких единицах представлены данные на графике или диаграмме.

Также на рисунок можно нанести линии вспомогательной сетки (grid). В pyplot она вызывается командой `plt.grid()`. Вспомогательная сетка связана с делениями координатных осей (ticks), которые определяются автоматически исходя из значений выборки.

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. В плане настройки главные деления ничем не отличаются от вспомогательных.

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 "OX") и ось ординат (ось "OY"). В полярной системе координат координатными осями являются радиус и угол наклона, что выражается в виде своеобразного тригонометрического круга.

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

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.

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 `haxis` и `uaxis`. Чтобы обратиться к экземпляру `axis`, отвечающему, например, за ось ординат, нужно обратиться к контейнеру `uaxis` соответствующей области рисования `ax`. Причём запись `"uax=ax.get_yaxis()"` идентична записи `"uax=ax.yaxis"`.

Каждый экземпляр `axis` содержит атрибут подписей (`label`) координатной оси и список главных (`major ticks`) и вспомогательных (`minor ticks`) делений, а также является хранилищем для экземпляров делений (`ticks`).

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`.

Хотя все графические примитивы, с помощью которых создаётся облик координатной оси содержатся в делениях **ticks**, у экземпляров `axis` есть средства для управления линиями делений (`tick lines`), подписями делений (`tick labels`), а также местоположением делений (`tick locations`).

```
In [38]: ► import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()

ax = fig.add_subplot(111)

x = np.arange(100)
y = -np.exp(-0.3*x) + 2.33

ax.plot(x, y, 'k')

xax = ax.xaxis

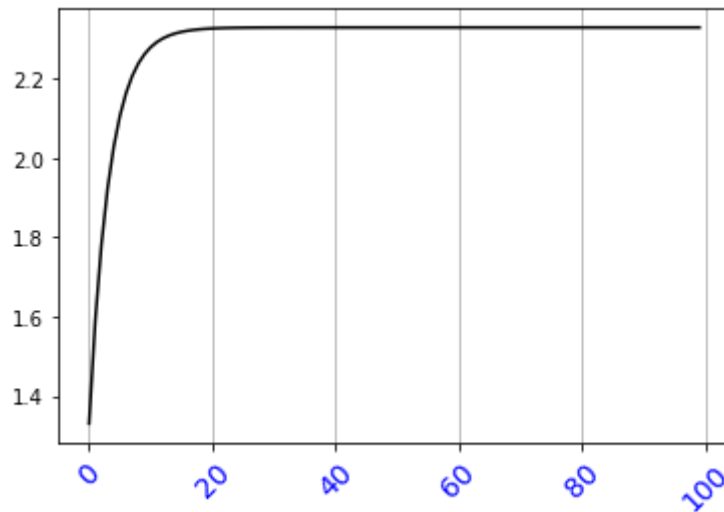
xlocs = yax.get_ticklocs()
print ('Major X-ticks locations:', xlocs)
xlabels = yax.get_ticklabels()
print ('Major X-ticks labels:', xlabels)
xlines = yax.get_ticklines()
print ('Major X-ticks tick lines:', xlines)

# Линии вспомогательной сетки (главные деления) только по оси абсцисс
xax.grid(True)

for label in xlabels:
    # цвет подписи делений оси OX
    label.set_color('blue')
    # поворот подписей делений оси OX
    label.set_rotation(45)
    # размер шрифта подписей делений оси OX
    label.set_fontsize(14)

plt.show()
```

```
Major X-ticks locations: [-20.  0.  20.  40.  60.  80. 100. 120.]
Major X-ticks labels: <a list of 8 Text major ticklabel objects>
Major X-ticks tick lines: <a list of 16 Line2D ticklines objects>
```



Главные и вспомогательные деления

В matplotlib деления на координатной оси могут быть главными (major ticks) или вспомогательными (minor ticks). По умолчанию наносятся главные деления, к ним привязывается нанесение вспомогательной сетки grid, за которую отвечает параметр `axes.grid.which`: 'major' из `rcParams`.

Работа со вспомогательными делениями ничем не отличается от работы с главными делениями: используются те же методы, но указывается тип делений через определённый атрибут. Обычно этот атрибут называется `which` (принимает значения 'major'/'minor') или булевый атрибут `minor`.

Ниже представлен список некоторых методов Axis. Они применяются к экземплярам `haxis` или `yaxis`. Так как при работе с атрибутами поддерживается идеология "set-get", то списке указаны методы только с приставкой "set_". Для получения значений атрибутов какой-либо характеристики, нужно в методе заменить приставку "set_" на "get": `set_scale` -> `get_scale`; `get_data_interval` -> `set_data_interval` и так далее.

Вспомогательный метод -> Описание

- `set_scale` -> определяет характер изменений значений на оси: линейный 'linear' (по умолчанию) или 'log' логарифмический;
- `set_view_interval` -> экземпляр интервала области отображения по оси axis;
- `set_data_interval` -> экземпляр интервала области изменения данных по оси axis;
- `set_gridlines` -> список линий сетки для Axis;

- `set_label` -> подпись оси, экземпляр `Text`;
- `set_ticklabels` -> подписи делений, список экземпляров `Text`. Тип делений определяет параметр `minor=True/False`;
- `set_ticklines` -> конфигурация делений, список экземпляров `Line2D`. Тип делений определяет параметр `minor=True/False`;
- `set_ticklocs` -> положение делений. Тип делений определяет параметр `minor=True/False`;
- `set_major_locator` -> экземпляр `matplotlib.ticker.Locator` для главных (`major`) делений;
- `set_major_formatter` -> экземпляр `matplotlib.ticker.Formatter` для главных делений;
- `set_major_ticks` -> список экземпляров `Tick` для главных делений;
- `set_minor_locator` -> экземпляр `matplotlib.ticker.Locator` для вспомгательных (`minor`) делений;
- `set_minor_formatter` -> экземпляр `matplotlib.ticker.Formatter` для вспомогательных делений;
- `set_minor_ticks` -> список экземпляров `Tick` для вспомогательных делений;
- `grid` -> определяет будет ли отображаться линии сетки для главной или вспомогательных делений (параметр `which='major'/'minor'`);


```
In [39]: ► fig = plt.figure()

ax = fig.add_subplot(111)
rect = ax.patch
rect.set_facecolor('lightslategray')
rect.set_alpha(0.25)

x = np.arange(100)
y = np.exp(-0.3*x) + 2.33

ax.plot(x, y, 'green')

# экземпляр xaxis
xax = ax.xaxis

print ('Major X-ticks tick lines:', yax.get_ticklines())
print ('Major X-ticks location:', yax.get_ticklocs())
print ('Major X-ticks labels:', yax.get_ticklabels())

print ('Major X-ticks tick lines:', yax.get_ticklines(minor=True))
print ('Minor X-ticks location:', yax.get_ticklocs(minor=True))
print ('Minor X-ticks labels:', yax.get_ticklabels(minor=True))

# вспомогательная сетка для главных делений
ax.grid(True, which='major', color='w', linewidth=2., linestyle='solid')

# вспомогательная сетка для вспомогательных делений
ax.grid(True, which='minor', color='b')

# Ось абсцисс
for label in yax.get_ticklabels():
    # Label - это экземпляр текста Text
    label.set_color('red')
    label.set_rotation(-45)
    label.set_fontsize(15)

# Ось ординат
for line in yax.get_ticklines():
    # Line - это экземпляр плоской линии Line2D
    line.set_color('blue') # задаём цвет линии деления
    line.set_markersize(14) # задаём длину линии деления
```

```

line.set_markeredgewidth(3)  # задаём толщину линии деления

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

plt.show()

```

Major X-ticks tick lines: <a list of 16 Line2D ticklines objects>

Major X-ticks location: [-20. 0. 20. 40. 60. 80. 100. 120.]

Major X-ticks labels: <a list of 8 Text major ticklabel objects>

Major X-ticks tick lines: <a list of 0 Line2D ticklines objects>

Minor X-ticks location: []

Minor X-ticks labels: <a list of 0 Text minor ticklabel objects>

Тип изменения оси: linear

Область изменения отображения данных: [-4.95 103.95]

Область изменения отображения данных: [0. 99.]

Линии сетки: <a list of 8 Line2D gridline objects>

Подпись оси: Text(0.5, 0, '')

Подписи делений: <a list of 8 Text major ticklabel objects>

Линии делений: <a list of 16 Line2D ticklines objects>

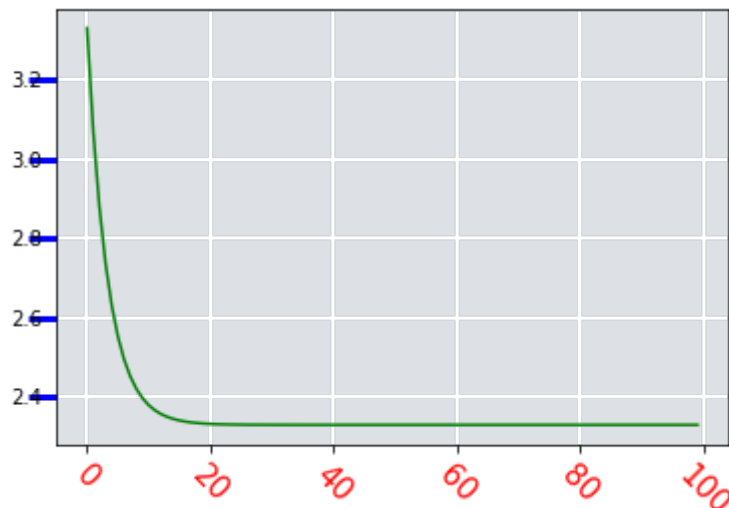
Расположение делений: [-20. 0. 20. 40. 60. 80. 100. 120.]

locator главных делений: <matplotlib.ticker.AutoLocator object at 0x7fe15fd442e8>

formatter главных делений: <matplotlib.ticker.ScalarFormatter object at 0x7fe15fd44358>

Список главных делений оси: [<matplotlib.axis.XTick object at 0x7fe15fcc3ba8>, <matplotlib.axis.XTick object at 0x7fe15fd449e8>, <matplotlib.axis.XTick object at 0x7fe15fd68668>, <matplotlib.axis.XTick object at 0x7fe15fd689e8>, <matplotlib.axis.XTick object at 0x7fe15fc9fe80>, <matplotlib.axis.XTick object at 0x7fe15fd68c50>, <matplotlib.axis.XTick object at 0x7fe15fd68d50>, <matplotlib.axis.XTick object at 0x7fe15fd68e50>]

```
s.XTick object at 0x7fe15fc9fa90>, <matplotlib.axis.XTick object at 0x7fe15fc9f208>]
locator вспомогательных делений: <matplotlib.ticker.NullLocator object at 0x7fe15fc80160>
formatter вспомогательных делений: <matplotlib.ticker.NullFormatter object at 0x7fe15fd44da0>
Список вспомогательных делений оси: []
-----
```



Деления координатной оси Ticks

Деления (ticks) неотделимы от координатной оси на которой они находятся. Однако свойства самих делений (цвет, длина, толщина и др.), и их подписей (кегель, 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 Ticks

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 атрибутов 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()`

Атрибуты:

Атрибут `Tick` -> Описание

- `tick1line` - 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 снизу.

```
In [40]: import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)

rect = ax.patch
rect.set_facecolor('w')

N = 31
x = np.arange(N)
y = 100*np.random.rand(N)

ax.plot(x, y, color='red', linewidth=2.0)

for tick in ax.yaxis.get_major_ticks():
    print ('Major ticks on Y-axis %s' % type(tick))
    tick.label10n = True
    tick.label1.set_color('green')
    tick.label20n = True
    tick.label2.set_color('blue')
    # серые деления на оси OY слева
    tick.tick1line.set_color('grey')
    tick.tick1line.set_markeredgewidth(2)
    tick.tick1line.set_markersize(25)
    # линии вспомогательной сетки для оси OX
    tick.grid0n = True
    tick.gridline.set_color('grey')
    tick.gridline.set_linewidth(1.5)

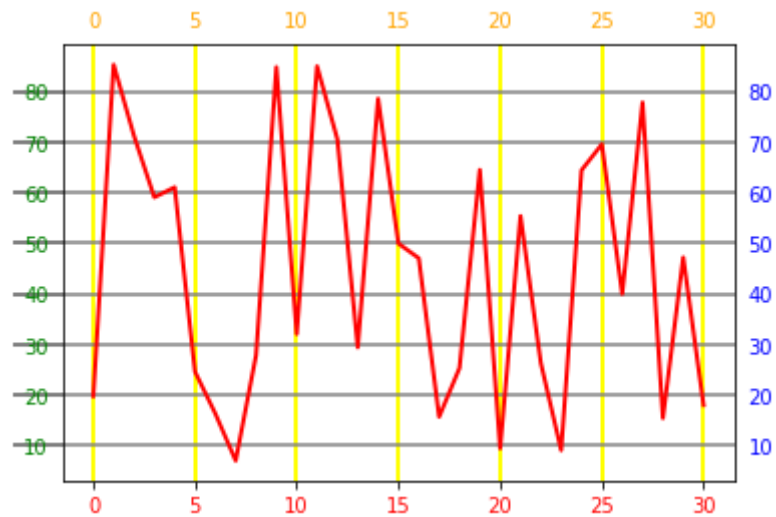
for tick in ax.xaxis.get_major_ticks():
    print ('Major ticks on X-axis %s' % type(tick))
    tick.label10n = True
    tick.label1.set_color('red')
    tick.label20n = True
    tick.label2.set_color('orange')

    # линии вспомогательной сетки для оси OY
    tick.grid0n = True
    tick.gridline.set_color('yellow')
```

```
tick.gridline.set_linewidth(2.)
```

```
plt.show()
```

```
Major ticks on Y-axis <class 'matplotlib.axis.YTick'>
Major ticks on Y-axis <class 'matplotlib.axis.YTick'>
Major ticks on Y-axis <class 'matplotlib.axis.YTick'>
Major ticks on Y-axis <class 'matplotlib.axis.YTick'>
Major ticks on Y-axis <class 'matplotlib.axis.YTick'>
Major ticks on Y-axis <class 'matplotlib.axis.YTick'>
Major ticks on Y-axis <class 'matplotlib.axis.YTick'>
Major ticks on Y-axis <class 'matplotlib.axis.YTick'>
Major ticks on Y-axis <class 'matplotlib.axis.YTick'>
Major ticks on Y-axis <class 'matplotlib.axis.YTick'>
Major ticks on X-axis <class 'matplotlib.axis.XTick'>
Major ticks on X-axis <class 'matplotlib.axis.XTick'>
Major ticks on X-axis <class 'matplotlib.axis.XTick'>
Major ticks on X-axis <class 'matplotlib.axis.XTick'>
Major ticks on X-axis <class 'matplotlib.axis.XTick'>
Major ticks on X-axis <class 'matplotlib.axis.XTick'>
Major ticks on X-axis <class 'matplotlib.axis.XTick'>
Major ticks on X-axis <class 'matplotlib.axis.XTick'>
```



Locator и Formatter

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 нужно передать какой-либо экземпляр класса из модуля `matplotlib.ticker`, например, `matplotlib.ticker.MultipleLocator`.

In [42]: `from matplotlib.ticker import MultipleLocator, FormatStrFormatter`

```
majorLocator = MultipleLocator(5)
majorFormatter = FormatStrFormatter('%d')
```

```
print(type(majorLocator))
print(type(majorFormatter))
```

```
<class 'matplotlib.ticker.MultipleLocator'>
<class 'matplotlib.ticker.FormatStrFormatter'>
```

Класс **Locator** - базовый класс для всех производных классов-локаторов, отвечающих за расположение делений. Локаторы работают с автомасштабированием 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. Субкласс `AutoMinorLocator()` позволяет автоматически определить положение вспомогательных делений.

Класс **Formatter** является базовым для всех производных классов-форматоров, отвечающих за форматирование делений. Форматоры в качестве входящих данных принимают строку формата, например `"%d"`, которая применяется к подписи каждого деления. Разные субклассы предоставляют разный функционал. Субкласс `NullFormatter()` позволяет скрыть все подписи на выбранной оси.

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.

Количество созданных делений равно $n-1$.


```
In [43]: import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator, FormatStrFormatter, AutoMinorLocator, NullFormatter

majorLocator = MultipleLocator(5)
# Автоматический подбор промежуточных делений
minorLocator = AutoMinorLocator(n=3)

majorFormatter = FormatStrFormatter('%d')
minorFormatter = FormatStrFormatter('%.2f')

N = 10
x = np.arange(-N, N+1, 1)
y = (np.random.random(len(x))*2.-1)*N

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111)

ax.plot(x, y)

xax = ax.xaxis
yax = ax.yaxis

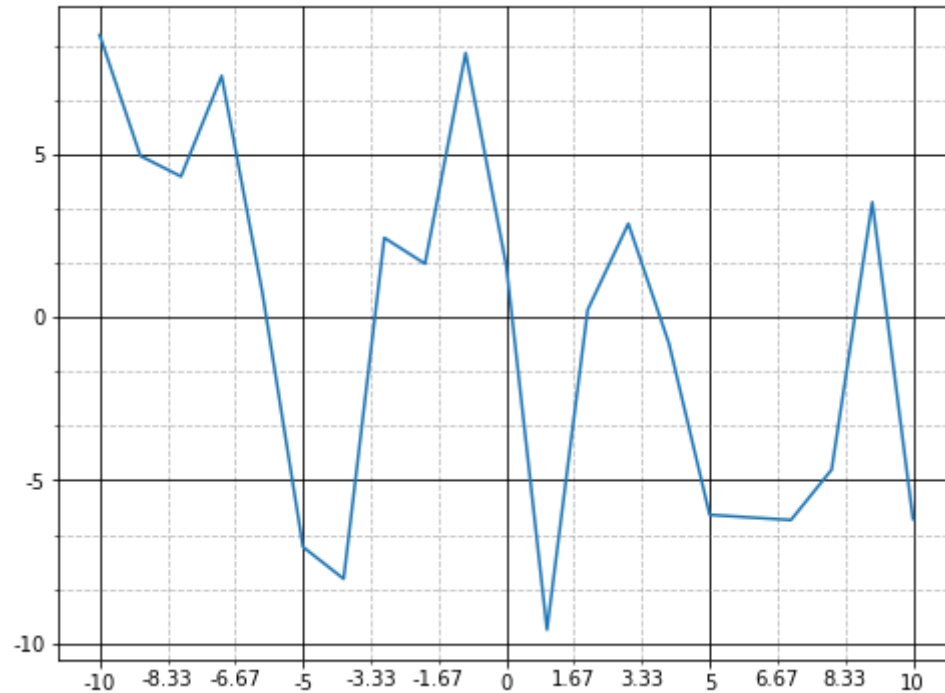
xax.set_major_locator(majorLocator)
xax.set_major_formatter(majorFormatter)
yax.set_major_locator(majorLocator)
yax.set_major_formatter(majorFormatter)

xax.set_minor_locator(minorLocator)
xax.set_minor_formatter(minorFormatter)
yax.set_minor_locator(minorLocator)

# скрываем подписи вспомогательных делений по оси OY
yax.set_minor_formatter(NullFormatter())

ax.grid(True, which='major', color='k', linestyle='solid')

ax.grid(True, which='minor', color='grey', linestyle='dashed', alpha=0.5)
```



Особенности координатных осей

Дополнительная координатная ось

Нанесение на график дополнительной координатной оси необходимо в случае, когда на один рисунок наносится две величины, имеющие общие единицы измерения по одной оси, но разные по другой. Это могут быть временные ряды сильно отличающихся по масштабу величин. Или ряды величин, имеющих разные единицы измерения. В таких случаях удобно нарисовать дополнительную координатную ось (обычно - ординат).

Такую возможность обеспечивает метод `ax.twinx()` для оси OX и метод `ax.twinx()` для оси OY . В результате создаётся ещё одна область рисования, совпадающая по размерам с исходной. На каждой области рисуются соответствующие значения, при этом с помощью пользовательской настройки подписей осей ординат можно добиться, чтобы вспомогательные сетки `grid` обеих областей совпадали.

```
In [44]: ► from matplotlib import rcParams
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-2*np.pi, 2*np.pi, 0.2)
y = np.sin(x) * np.cos(x)
f = np.sin(x) + np.cos(x)

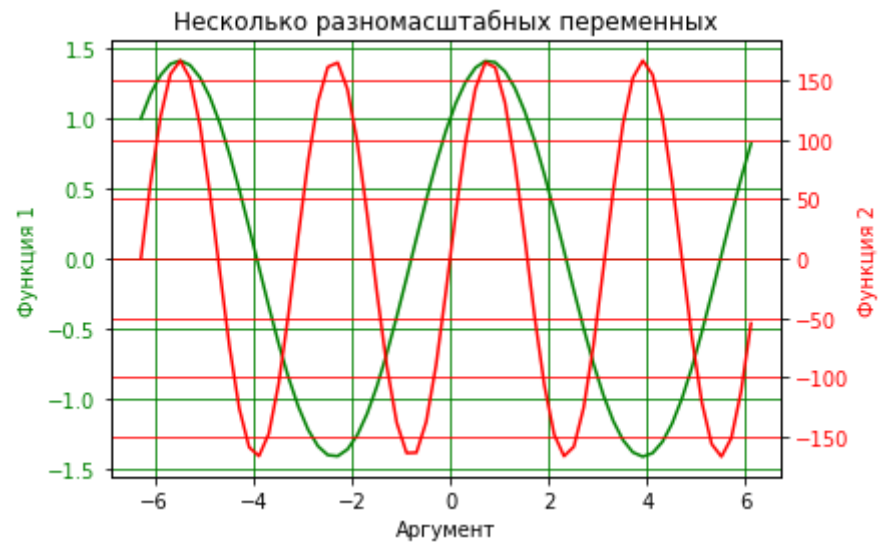
fig = plt.figure()
ax1 = fig.add_subplot(111)
ax2 = ax1.twinx() # Создаём вторую область рисования ax2

ax1.plot(x, f, label = u'Сумма cos и sin', color='green')
ax1.set_xlabel(u'Аргумент')
ax1.set_ylabel(u'Функция 1', color='green')
ax1.grid(True, color='green')
ax1.tick_params(axis='y', which='major', labelcolor='green')

ax2.plot(x, y*333, label = u'Произведение cos и sin', color='red')
ax2.set_ylabel(u'Функция 2', color='red')
ax2.grid(True, color='red')
ax2.tick_params(axis='y', which='major', labelcolor='red')

ax1.set_title(u'Несколько разномасштабных переменных')

plt.show()
```



Работа с легендой. Создание единой легенды.

```
In [45]: ▶ import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-2*np.pi, 2*np.pi, 0.2)
y = np.sin(x) * np.cos(x)
f = np.sin(x) + np.cos(x)

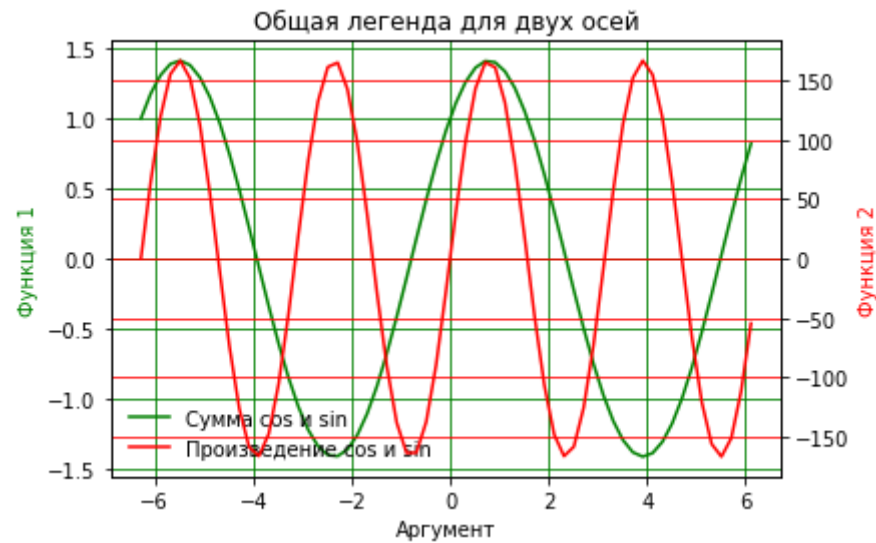
fig = plt.figure()
ax1 = fig.add_subplot(111)
line1 = ax1.plot(x, f, label = u'Сумма cos и sin', color='green')
ax1.set_xlabel(u'Аргумент')
ax1.set_ylabel(u'Функция 1', color='green')
ax1.grid(True, color='green')

ax2 = ax1.twinx() # Создаём вторую шкалу ax2
line2 = ax2.plot(x, y*333, label = u'Произведение cos и sin', color='red')
ax2.set_ylabel(u'Функция 2', color='red')
ax2.grid(True, color='red')

lns = line1 + line2
labs = [l.get_label() for l in lns]
ax1.legend(lns, labs, loc= 3 , frameon=False)

plt.title(u'Общая легенда для двух осей')

plt.show()
```



Графики в полярной системе координат

В некоторых задачах намного нагляднее отображать информацию на круге, а не в прямоугольнике. По умолчанию двумерные графики имеют прямоугольную декартову систему координат, однако matplotlib позволяет строить графики также в полярной системе координат.

В декартовой системе координат, которая используется по умолчанию, положение точки определяется двумя значениями: аргументом x и значением функции y .

В полярной системе координат положение точки также определяется двумя значениями: радиусом r и углом ϕ . Угол обычно отсчитывают против часовой стрелки от "восточного" (3:00) направления.

```
In [46]: ► from math import pi
import matplotlib.pyplot as plt
import numpy as np

lag = pi/4.
angles = np.arange(0, 2*pi , lag)

fig = plt.figure(figsize=(10,5))
xrange = np.arange(1,3,1)
for i in xrange:
    r = np.random.random(len(angles))

ax = fig.add_subplot(120+i, projection='polar')
ax.plot(angles, r, color='r', linewidth=1.)

ax.plot((angles[-1],angles[0]),(r[-1],r[0]), color='r', linewidth=1.)

ax.set_theta_direction(-1)
ax.set_theta_offset(pi/2.0)
ax.set_title(u"Позиция ветров %d" % i, loc='center')

plt.tight_layout()

plt.show()
```



Цвета и цветовая палитра

Цвет может быть задан либо через пропорции трёх базовых цветов: красного, зелёного и синего. Такой способ носит название RGB. Другим стандартным методом является использование шестнадцатичной кодировки (HEX), которая широко применяется в HTML. В не зависимости от способа задания цвета нужно понимать, какой цвет получается при той или иной комбинации.

Многообразие сочетаний цветов порождает массу наборов цветов - цветовые палитры. Они нужны в качестве основы для отображения цветовых легенд или цветовых шкал, где каждому оттенку цвета шкалы сопоставляется какое-то значение.

Цвет как декоративный элемент

Matplotlib предоставляет широкие возможности для работы с цветом. Существует ряд предустановленных цветов, которые имеют следующие аббревиатуры:

- Красный - 'red', 'r';
- Оранжевый - 'orange';
- Жёлтый - 'yellow', 'y';
- Зелёный - 'green', 'g';
- Голубой - 'cyan', 'c';
- Синий - 'blue', 'b';
- Фиолетовый - 'magenta', 'm';
- Чёрный - 'black', 'k';
- Белый - 'white', 'w'.

Цветовую гамму можно разнообразить за счёт различной степени прозрачности параметра alpha. Но если нужно задать конкретно, например, алый цвет, то нужна пользовательская настройка.

Matplotlib отображает цвет, заданный как в формате RGB, так и в HEX. Существуют различные [таблицы цветов](https://www.webucator.com/blog/2015/03/python-color-constants-module/) (<https://www.webucator.com/blog/2015/03/python-color-constants-module/>), в которых представленным оттенкам соответствует код в RGB или HEX формате. Ими удобно пользоваться при составлении небольших пользовательских цветовых палитр.

Способы задания цветов. RGB и HEX

Цвет в формате RGB представляет собой триплет или кортеж, состоящий из трёх целых значений от 0 до 255 для красного, зелёного и синего цветов. Различные сочетания этих трёх значений позволяет получить огромное количество оттенков и цветов.

В matplotlib цвета rgb значения задаются в относительных единицах, в диапазоне от [0, 1]. Если необходимый цвет задан в диапазоне [0, 255], то нужно просто поделить их на 255.

Цвет, представленный в формате HEX, передаётся в виде шестнадцатеричной html строки (символ # перед шестизначным значением обязателен).

```
In [47]:  ▶ import matplotlib.pyplot as plt
import numpy as np
```

```
In [48]: fig = plt.figure()

# добавление области рисования ax
ax = fig.add_subplot(111)

N = 10
x = np.arange(1, N+1, 1)
y = 20.*np.random.rand(N)

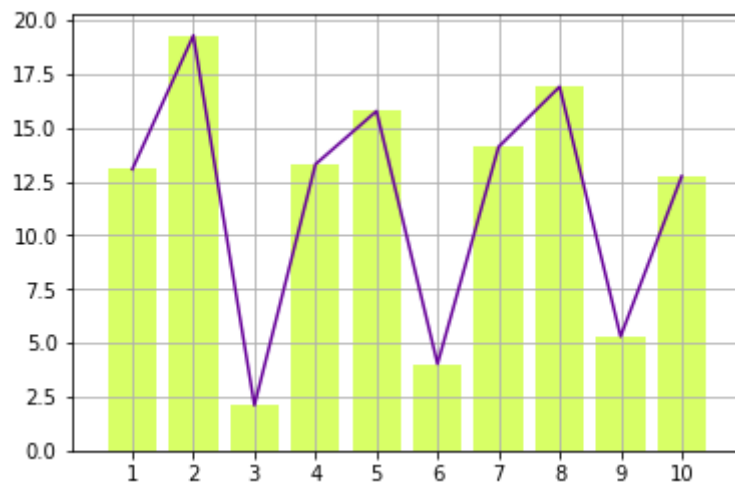
rgb = np.array([204,255,51])/255.
myhex = '#660099'

ax.plot(x, y, color=myhex)
ax.bar(x, y, color=rgb, alpha=0.75, align='center')

# установка делений на оси OX
ax.set_xticks(x)
# ограничение области изменения по оси OX
ax.set_xlim(np.min(x)-1, np.max(x)+1)

ax.grid(True)

plt.show()
```



Цветовая палитра colormap

Последовательность или набор цветов образует цветовую палитру **colormap**. Чаще всего она используется при отрисовке трёхмерных данных. Но автоматический подбор цветов при добавлении каждого нового экземпляра plot также осуществляется из цветовой палитры по умолчанию.

Для получения текущей цветовой палитры можно воспользоваться методом **plt.get_cmap('название палитры')**. Список всех предустановленных палитр можно получить с помощью метода **plt.cm.datad**.

В настройках matplotlibrc можно также изменить цветовую палитру с помощью параметра **image.cmap**. В интерактивном режиме её можно поменять через **plt.rcParams['image.cmap']='имя_палитры'** или через **plt.set_cmap('имя_палитры')**. Последний позволяет изменить палитру текущего рисунка уже после вызова графических команд.

```
In [49]: ▶ import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

# матрица значений
dat = np.random.random(200).reshape(20,10)
```

Создание списка цветовых палитр из словаря

```
In [50]: ▶ maps = [m for m in plt.cm.datad]
```

```
In [52]: ▶ for i, m in enumerate(plt.cm.datad):
            maps.append(m)
            print(u'Предустановленные цветовые палитры:', maps)

fig, axes = plt.subplots(nrows=2, ncols=2, sharex=True, sharey=True)

cmaplist = plt.cm.datad

for ax in fig.axes:
    random_cmap = np.random.choice(maps)
    cf = ax.contourf(dat, cmap=plt.get_cmap(random_cmap))
    ax.set_title('%s colormap' % random_cmap)
    fig.colorbar(cf, ax=ax)

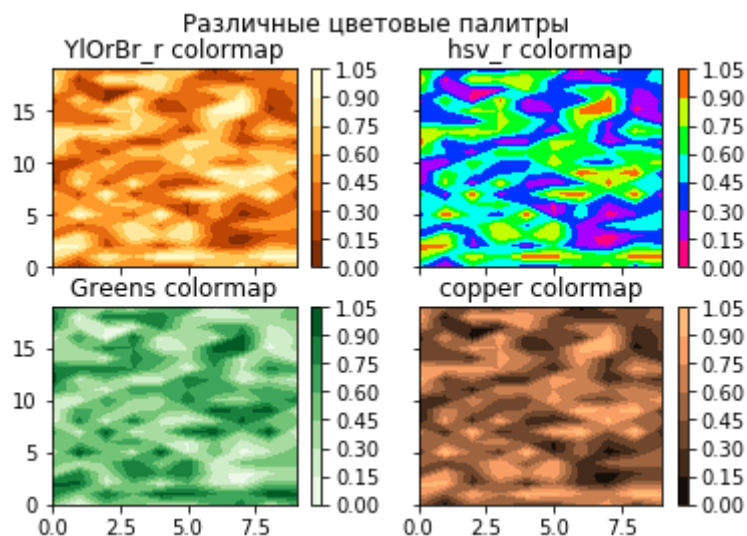
# единый заголовок рисунка
plt.suptitle(u'Различные цветовые палитры')

plt.show()
```

```

_r', 'PuOr_r', 'PuRd_r', 'Purples_r', 'RdBu_r', 'RdGy_r', 'RdPu_r', 'RdYlBu_r', 'RdYlGn_r', 'Reds_r', 'Spectral_r',
'Wistia_r', 'YlGn_r', 'YlGnBu_r', 'YlOrBr_r', 'YlOrRd_r', 'afmhot_r', 'autumn_r', 'binary_r', 'bone_r', 'brg_r', 'b
wr_r', 'cool_r', 'coolwarm_r', 'copper_r', 'cubehelix_r', 'flag_r', 'gist_earth_r', 'gist_gray_r', 'gist_heat_r',
'gist_ncar_r', 'gist_rainbow_r', 'gist_stern_r', 'gist_yarg_r', 'gnuplot_r', 'gnuplot2_r', 'gray_r', 'hot_r', 'hsv_
r', 'jet_r', 'nipy_spectral_r', 'ocean_r', 'pink_r', 'prism_r', 'rainbow_r', 'seismic_r', 'spring_r', 'summer_r',
'terrain_r', 'winter_r', 'Accent_r', 'Dark2_r', 'Paired_r', 'Pastel1_r', 'Pastel2_r', 'Set1_r', 'Set2_r', 'Set3_r',
'tab10_r', 'tab20_r', 'tab20b_r', 'tab20c_r', 'Blues', 'BrBG', 'BuGn', 'BuPu', 'CMRmap', 'GnBu', 'Greens', 'Greys',
'OrRd', 'Oranges', 'PRGn', 'PiYG', 'PuBu', 'PuBuGn', 'PuOr', 'PuRd', 'Purples', 'RdBu', 'RdGy', 'RdPu', 'RdYlBu',
'RdYlGn', 'Reds', 'Spectral', 'Wistia', 'YlGn', 'YlGnBu', 'YlOrBr', 'YlOrRd', 'afmhot', 'autumn', 'binary', 'bone',
'brg', 'bwr', 'cool', 'coolwarm', 'copper', 'cubehelix', 'flag', 'gist_earth', 'gist_gray', 'gist_heat', 'gist_nca
r', 'gist_rainbow', 'gist_stern', 'gist_yarg', 'gnuplot', 'gnuplot2', 'gray', 'hot', 'hsv', 'jet', 'nipy_spectral',
'ocean', 'pink', 'prism', 'rainbow', 'seismic', 'spring', 'summer', 'terrain', 'winter', 'Accent', 'Dark2', 'Paire
d', 'Pastel1', 'Pastel2', 'Set1', 'Set2', 'Set3', 'tab10', 'tab20', 'tab20b', 'tab20c', 'Blues_r', 'BrBG_r', 'BuGn_
r', 'BuPu_r', 'CMRmap_r', 'GnBu_r', 'Greens_r', 'Greys_r', 'OrRd_r', 'Oranges_r', 'PRGn_r', 'PiYG_r', 'PuBu_r', 'Pu
BuGn_r', 'PuOr_r', 'PuRd_r', 'Purples_r', 'RdBu_r', 'RdGy_r', 'RdPu_r', 'RdYlBu_r', 'RdYlGn_r', 'Reds_r', 'Spectral
_r', 'Wistia_r', 'YlGn_r', 'YlGnBu_r', 'YlOrBr_r', 'YlOrRd_r', 'afmhot_r', 'autumn_r', 'binary_r', 'bone_r', 'brg_
r', 'bwr_r', 'cool_r', 'coolwarm_r', 'copper_r', 'cubehelix_r', 'flag_r', 'gist_earth_r', 'gist_gray_r', 'gist_heat
_r', 'gist_ncar_r', 'gist_rainbow_r', 'gist_stern_r', 'gist_yarg_r', 'gnuplot_r', 'gnuplot2_r', 'gray_r', 'hot_r',
'hsv_r', 'jet_r', 'nipy_spectral_r', 'ocean_r', 'pink_r', 'prism_r', 'rainbow_r', 'seismic_r', 'spring_r', 'summer_
r', 'terrain_r', 'winter_r', 'Accent_r', 'Dark2_r', 'Paired_r', 'Pastel1_r', 'Pastel2_r', 'Set1_r', 'Set2_r', 'Set3
_r', 'tab10_r', 'tab20_r', 'tab20b_r', 'tab20c_r']

```



Цветовая шкала colorbar

Это аналог легенды для отображения трёхмерных полей, заданных обычно в виде матрицы или двумерного массива. Цветовая шкала является неотъемлемой частью рисунков, создаваемых с помощью методов `ax.contour()`, `ax.contourf()`, `ax.imshow()`, `ax.pcolor()`, `ax.scatter()`.

Если на рисунке присутствует так называемый "mappable object", то на рисунке может быть нарисована цветовая шкала (`colorbar`). К шкале также можно делать подписи вдоль разных сторон. При этом сама цветовая может быть расположена как на текущей области рисования `axes`, отбирая у неё некоторую долю, либо может быть размещена на самостоятельной области рисования.

Каждая цветовая шкала является ещё одной областью рисования `Axes`. Это значит, что она имеет такие же методы и атрибуты, как и обычные экземпляры типа `Axes` или `AxesSubplot`. Имеются у неё и свои особенности.

```
In [54]: ► import matplotlib.pyplot as plt
import numpy as np

l = 0.1
x = np.arange(0.0, 2*np.pi+l, l)
y = np.cos(x)

fig = plt.figure()
plt.plot(x, y, label='line', color='blue') # синяя линия

plt.title('1 TITLE') # заголовок
plt.ylabel('3 Ylabel') # подпись оси OY
plt.xlabel('2 Xlabel ') # подпись оси OX

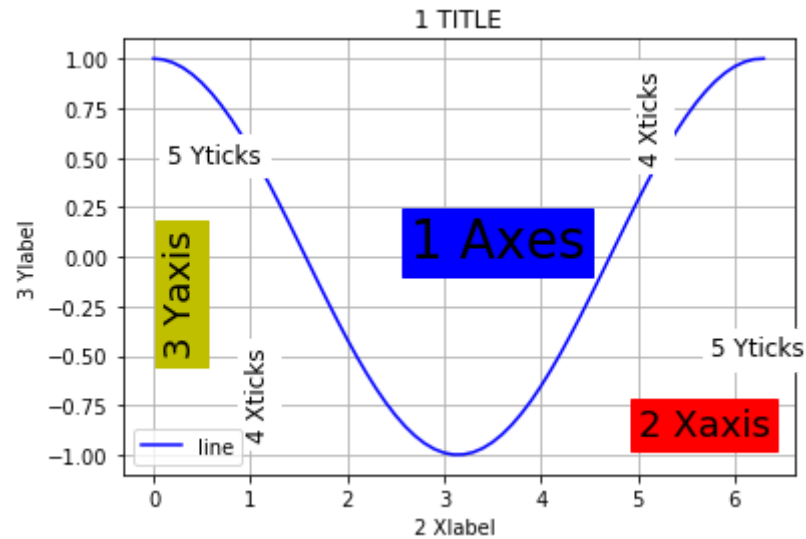
plt.legend() # легенда

plt.text(np.pi-0.5, 0, '1 Axes', fontsize=26, bbox=dict(edgecolor='r', color='b'))
plt.text(0.1, 0, '3 Yaxis', fontsize=18, bbox=dict(edgecolor='w', color='y'), rotation=90)
plt.text(5, -0.9, '2 Xaxis', fontsize=18, bbox=dict(edgecolor='w', color='r'))

plt.text(5, 0.85, '4 Xticks', fontsize=12, bbox=dict(edgecolor='w', color='w'), rotation=90)
plt.text(0.95, -0.55, '4 Xticks', fontsize=12, bbox=dict(edgecolor='w', color='w'), rotation=90)

plt.text(5.75, -0.5, '5 Yticks', fontsize=12, bbox=dict(edgecolor='w', color='w'))
plt.text(0.15, 0.475, '5 Yticks', fontsize=12, bbox=dict(edgecolor='w', color='w'))

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



Выбор формата графиков

При необходимости, например, для масштабирования или печати, можно переключить формат, в котором выводятся графики.

Векторные форматы позволяют сохранить качество при масштабировании и печати, но требуют больше ресурсов. По умолчанию используется растровый формат png.

```
In [ ]:  #векторный формат
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('svg')
```

Линейные графики

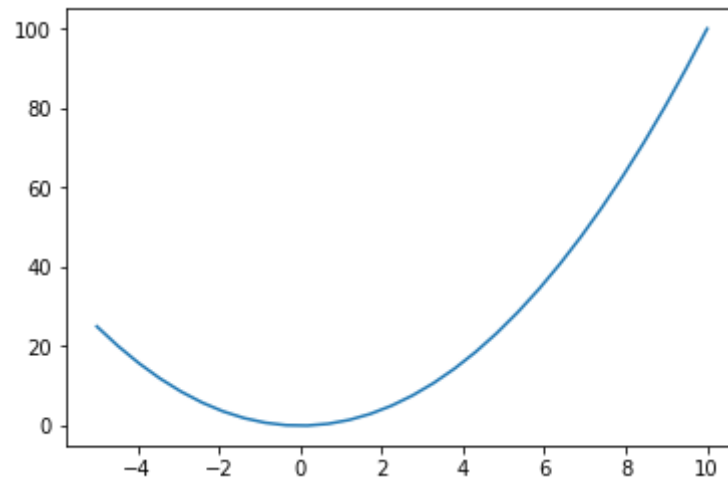
Все графики строятся по точкам, координаты которых хранятся в массивах NumPy. В записи ниже подключается пакет numpy, значению x присваиваем процедуру np.linspace, которая возвращает указанное количество чисел (30) на интервал [-5;10]. Переменная y равна x в квадрате.


```
In [55]: In [55]: ▶ import numpy as np  
x = np.linspace(-5, 10, 30)  
y = x ** 2
```

Для вывода линейных графиков используется функция plot()

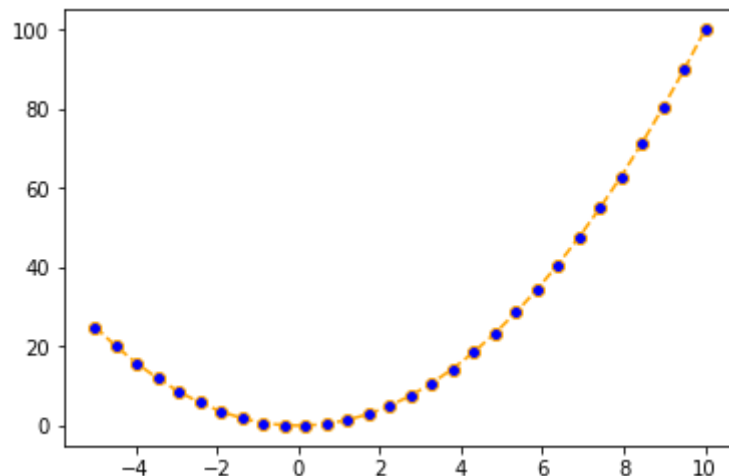
```
In [56]: In [56]: ▶ plt.plot(x,y)
```

Out[56]: [<matplotlib.lines.Line2D at 0x7fe15daecf60>]



```
In [57]: #изменение цвета, типа линии и маркера  
plt.plot(x, y, color='orange', marker='o', linestyle='--', markerfacecolor='blue')
```

```
Out[57]: [<matplotlib.lines.Line2D at 0x7fe15da617f0>]
```

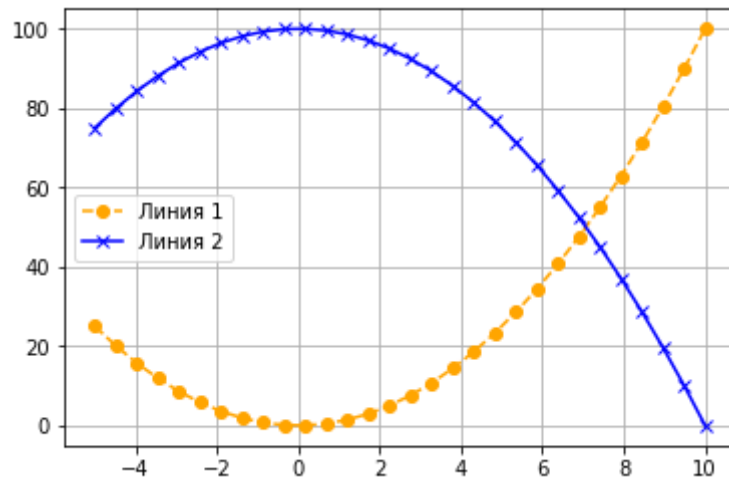


Оформление графиков

В ячейке блокнота можно вызывать функцию `plot()` несколько раз для нанесения нескольких линий на график и использовать различные функции для форматирования графика и добавления легенды `legend()`.

```
In [58]: ▶ # Полная форма указания параметров графика
plt.plot(x, y, color='orange', marker='o', linestyle='--', label='Линия 1')
plt.plot(x, -y + 100, color='blue', marker='x', label='Линия 2')
plt.grid()
# сетка
plt.legend(loc='best')
```

Out[58]: <matplotlib.legend.Legend at 0x7fe15d9c4860>



Вызов `legend()` без аргументов автоматически выбирает маркеры легенды.

С помощью функций `xlabel()`, `ylabel()` можно подписать координатные оси. Заглавие графика можно задать с помощью функции `title()`. В названия можно включать математические символы и формулы, используя LaTeX. Чтобы не возникало ошибок из-за специальных символов, используемых для набора формул, нужно добавлять символ `r` перед строкой с формулой, например: `r'\alpha'`.

```
In [59]: ▶ plt.plot(x, y, color='orange', marker='o', linestyle='--', label='Линия 1')
plt.plot(x, -y + 100, color='blue', marker='x', label='Линия 2')
plt.legend(loc='upper center', ncol=2)
plt.grid()

#Подписи для осей:
plt.xlabel('x', fontsize=14)
plt.ylabel('y', fontsize=14)

#Диапазон оси y:
plt.ylim(0, 120)

#Заголовок:
plt.title(r'График функций $y = x^2$ и $y = 100 - x^2$', fontsize=16, y=1.05);
```

