**Тема 5. Специализированные графики**

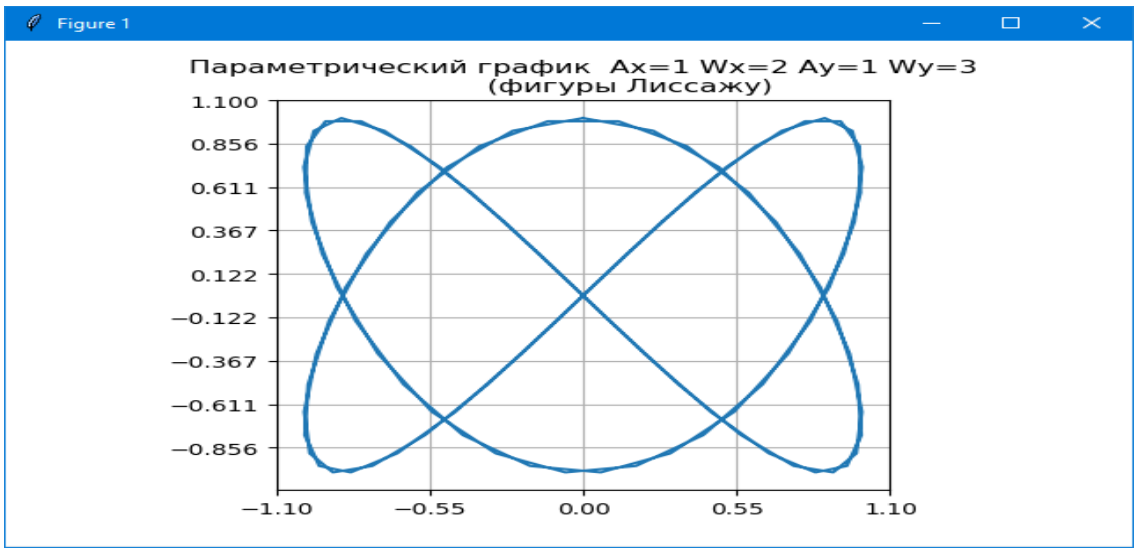
# П9. Параметрический график

Массив *x* не обязан быть монотонно возрастающим. Можно строить любую параметрическую линию `x=x(t), y=y(t)`.

Разрешающие способности монитора по разным осям могут быть разными. Поэтому чтобы окружности выглядели как окружности, а не как эллипсы, (а квадраты как квадраты, а не как прямоугольники), нужно установить параметр - aspect ratio (обычно равный 1).

Построим `x=Ax*cos(Wx*t), y=Ay*sin(Wy*t)`.

```
import matplotlib.pyplot as plt
import numpy as np
Ax=1; Ay=1; Wx=2; Wy=3
# независимая переменная
t=np.linspace(0,4*np.pi,100)
# установим параметр - aspect(ratio)
axx=plt.subplot(111)
axx.set_aspect(1)
# так можно изменить шаг сетки на графике
plt.xticks([i for i in   \
              np.linspace(-1.1*Ax, 1.1*Ax, 5)])
plt.yticks([i for i in \
             np.linspace(-1.1*Ay, 1.1*Ay, 10)])
# Задаем заголовок диаграммы
plt.title("Параметрический график " \
          " Ax={0} Wx={1} Ay={2} Wy={3}\n" \
          "           (фигуры Лиссажу)"  \
               .format(Ax, Wx, Ay, Wy) )
plt.plot(Ax*np.sin(Wx*t),Ay*np.cos(Wy*t))
plt.grid(True)
# визуализируем графики
plt.show()
```

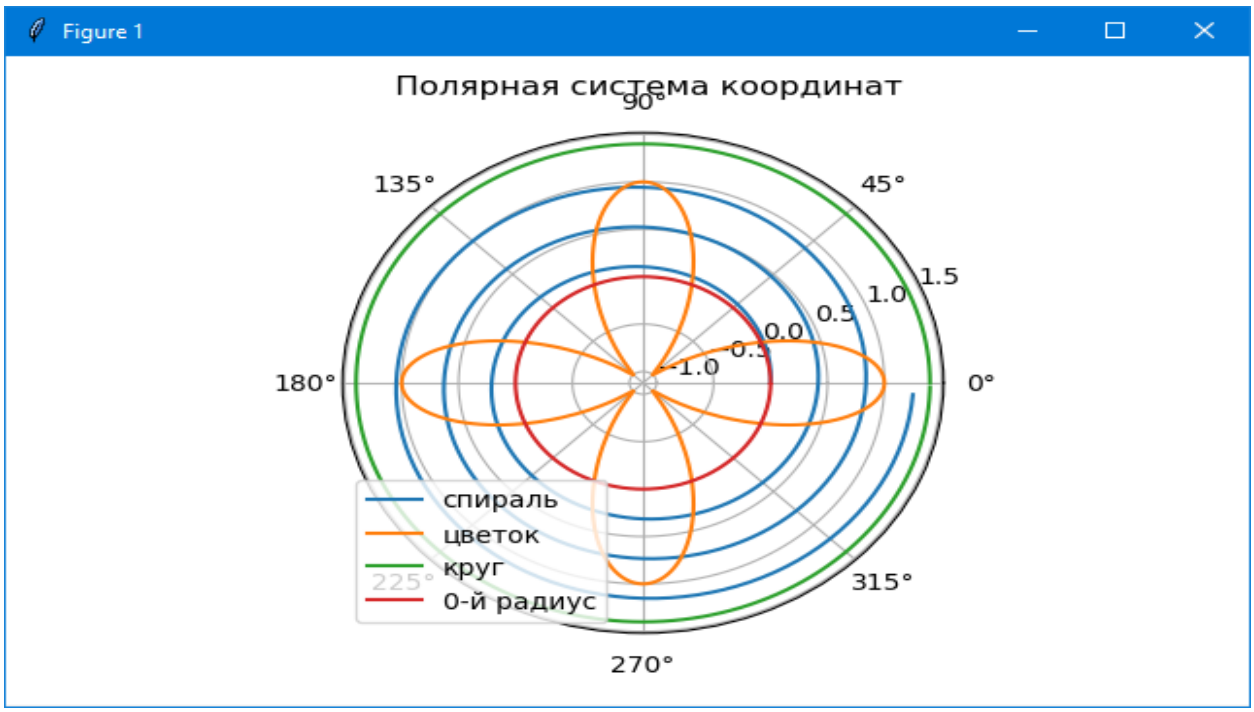



## П10. Полярные координаты

Кроме наиболее часто используемой декартовой системы координат, довольно широко применяется и полярная система координат, удобная в различных радиальных задачах, координаты точек в ней задается с помощью радиус-вектора **ρ**, идущего из начала координат и угла **θ**. Угол может быть задан в радианах или градусах, `matplotlib` использует градусы.

Четыре графика в полярных координатах:

```
import matplotlib.pyplot as plt
import numpy as np
theta = np.arange(0., 2., 1./180.)*np.pi
#plt.title('Поляная система координат')
plt.polar(3*theta, theta/5, label="спираль");
plt.polar(theta, 0+np.cos(4*theta), label="цветок");
plt.polar(theta, [1.4]*len(theta), label="круг");
plt.polar(theta, 0*theta, label="0-й радиус");
plt.title(" Полярная система координат")
lt.grid(True)
plt.legend(loc='lower left')
# визуализируем графики
plt.show()
```



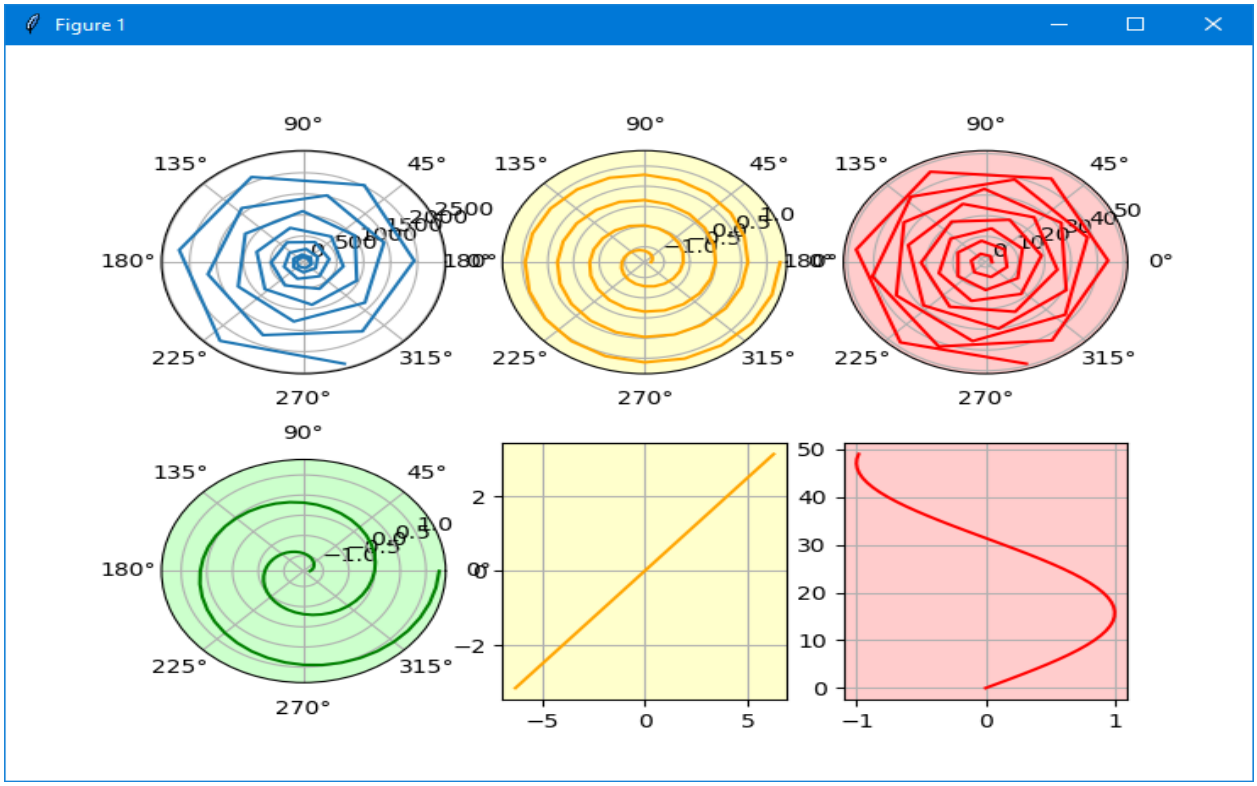

Для построения в полярных координат используется функция `polar()`. В аргументах первыми идут углы, потом радиусы. В примере используется один и тот же массив для углов и радиус-векторов и рисуется четыре разные кривые.

Первая образует спираль, поскольку с каждым новым углом меняется и радиус, вторая рисует цветок в соответствии с уравнением розы, третья и четвертая - окружности ввиду неизменности радиуса для всех углов (в данном случае он равен 1.4 и 0).

Еще один пример:

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

fig = plt.figure(figsize=(8., 6.))
x = np.arange(50)
y = x**2 + 2.5

lag = 0.05*np.pi
r = np.arange(0.0, 5.0, 0.1)
#phi = r*np.pi
phi=np.arange(-2*np.pi, 2*np.pi + lag, lag)
r=phi*0.2
# x и y
ax1 = fig.add_subplot(231, projection='polar')
ax1.plot(x, y)
ax1.grid(True)

ax2 = fig.add_subplot(232, projection='polar',
facecolor ='#FFFFCC')
```

```
ax2.plot(phi*2., r, 'orange')        #
facecolor='#FFFFCC') # axisbg
ax2.grid(True)

ax3 = fig.add_subplot(233, projection='polar',
facecolor='#FFCCCC')
ax3.plot(x, x, 'r')
ax3.grid(True)

ax4 = fig.add_subplot(234, projection='polar',
facecolor='#CCFFCC')
ax4.plot(phi, 0.2*phi, 'g') #(-1.5*phi, r, 'g')
ax4.grid(True)

ax5 = fig.add_subplot(235, facecolor='#FFFFCC')
ax5.plot(phi, phi*0.5, 'orange')
ax5.grid(True)

ax6 = fig.add_subplot(236, facecolor='#FFCCCC')
ax6.plot(np.sin(0.1*x), x, 'r')
ax6.grid(True)
###plt.legend(loc='lower left')
plt.show()
```

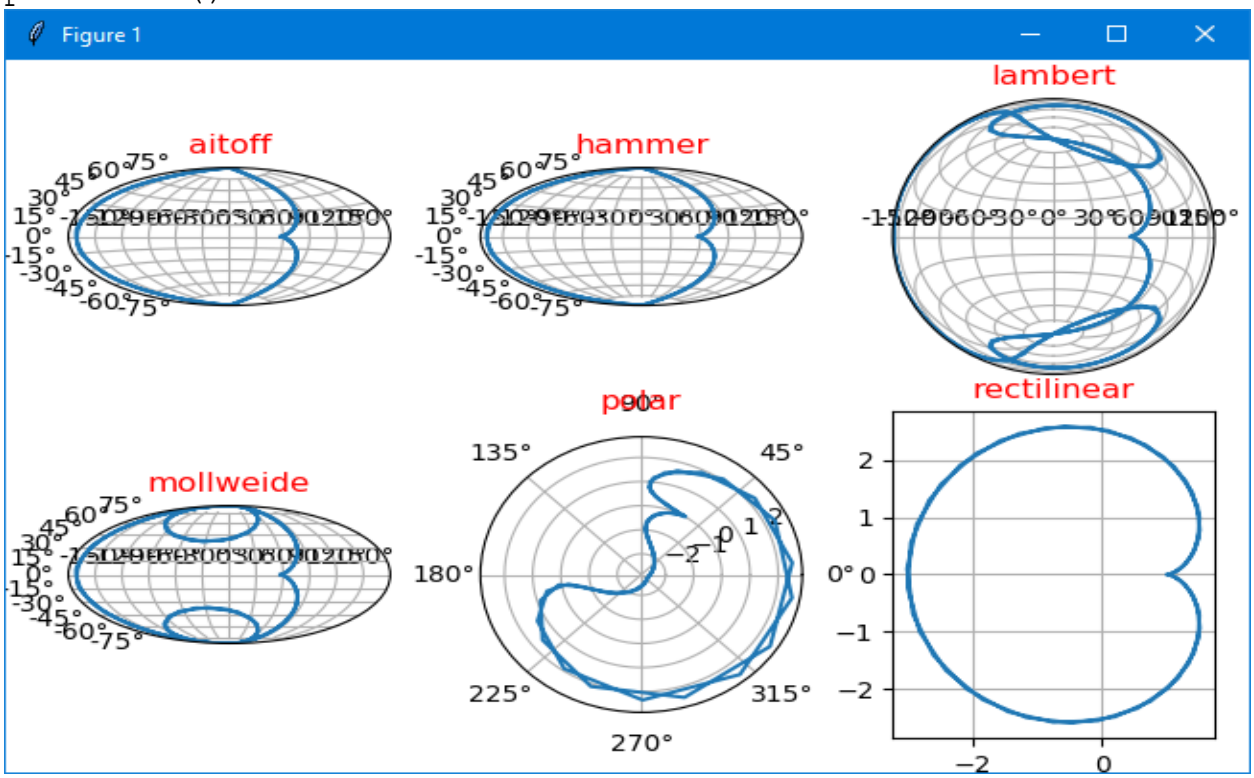



matplotlib позволяет рисовать не только в полярной системе координат, но и в других. За это отвечает параметр `projection`, который в случае равенства значению `'polar'` аналогичен значению параметра `polar=True`.

В matplotlib поддерживаются следующие проекции:
'aitoff', 'hammer', 'lambert', 'mollweide', 'polar', 'rectilinear'.

Чаще всего используются два последних варианта, остальные являются экзотическими для рутинных задач в научной графике.

```
import matplotlib.pyplot as plt
import numpy as np
a = 1.
x = np.arange(-2*np.pi, 2*np.pi, 0.2)
y = np.sin(x) * np.cos(x)
# Уравнение кардиойды
xz = a*(2*np.cos(x) - np.cos(2*x))
yz = a*(2*np.sin(x) - np.sin(2*x))
label = ['aitoff', 'hammer', 'lambert', 'mollweide',
'polar', 'rectilinear']
fig = plt.figure(figsize=(10,8))

for i in range(len(label)):
    ax = fig.add_subplot(231+i, projection=label[i])
    ax.plot(xz, yz)
    ax.set_title(label[i], color='r')
    ax.grid(True)
plt.tight_layout()
plt.show()
```




У полярной системы координат есть специальные функции, которые позволяют настраивать внешний вид рисунка.

Для радиуса R:

1. `ax.set_rlabel_position(phi)` - перемещает ось ординат (радиус) по кругу на угол phi (в ГРАДУСАХ) от положения нуля;
2. `ax.set_rmax(R)` - позволяет ограничить область изменения радиуса R на рисунке;
3. `ax.set_rmin(R)` - позволяет ограничить область изменения радиуса R на рисунке;
4. `ax.set_rlim()` - позволяет ограничить область изменения радиуса R на рисунке;
5. `ax.set_rscale()` - позволяет сделать шкалу радиусов логарифмической;
6. `ax.set_rgrid()` - позволяет настроить для оси радиуса вспомогательную сетку, положения делений, формат подписей к ним и т.д.

Для угла **phi** (в matplotlib он называется **theta**):

1. `ax.set_theta_zero_location(loc)` - перемещает положение нуля на определённое положение. `Loc` принимает значения `'N', 'NW', 'W', 'SW', 'S', 'SE', 'E'` или `'NE'`. По умолчанию положение нуля находится в положении "восток" или "3 часа";
2. `ax.set_theta_offset(phi)` - перемещает положение нуля на угол phi (в радианах) . По умолчанию положение нуля находится в положении "восток" или "3 часа";
3. `ax.set_theta_direction(loc)` - определяет направление обхода. `Loc` может быть либо -1 или по часовой стрелке и 1 или против часовой стрелки;
4. `ax.set_thetagrids()` - позволяет настроить для оси углов вспомогательную сетку, положения делений, формат подписей к ним и т.д.

## П11. График рассеяния

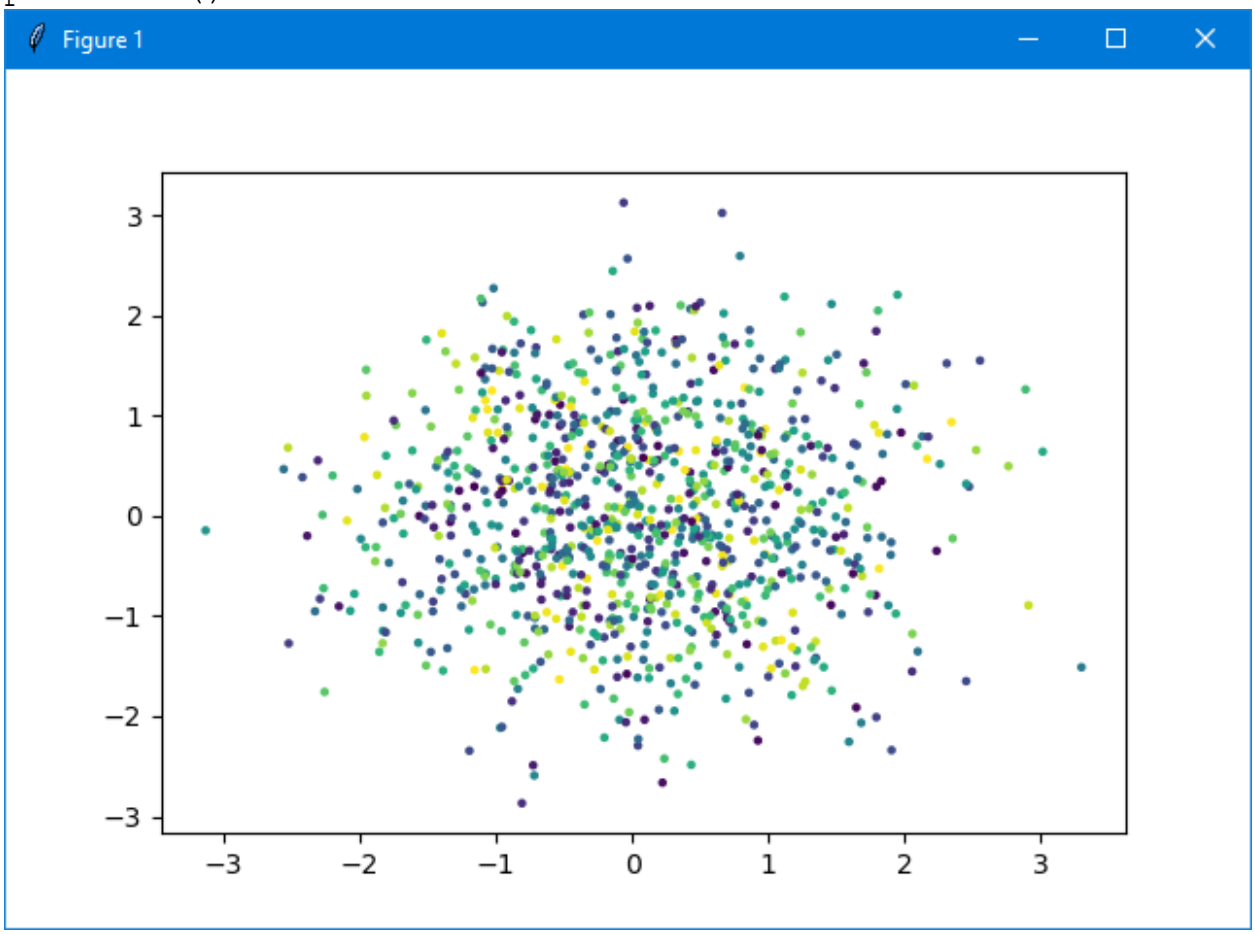
Такой тип графиков позволяет изображать одновременно два множества данных, которые не образуют кривой, а именно двухмерное множество точек. Каждая точка имеет две координаты. График рассеяния часто используется для определения связи между двумя величинами и позволяет определить более точные пределы измерений.

В модуле `matplotlib.pyplot` имеется своя функция, для графика рассеяния (`scatter plot`) это функция `scatter()`.

Она принимает две последовательности и изображает их на плоскости в виде маркеров, по умолчанию они круглые и синие. Но естественно, с ними можно поработать с помощью `keywords` той же функции:

- **s** задает размер маркеров и может быть как одним числом для всех, так и представлять массив значений
- **c** задает цвет маркеров, также либо один для всех, либо множество
- **marker** определяет тип маркера.

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

x = np.random.randn(1000)
y = np.random.randn(1000)

size = 5
colors = np.random.rand(1000)

plt.scatter(x, y, s=size, c=colors)
plt.show()
```




## П12. Настройки в стиле LaTEX

Вот пример настройки почти всего, что можно настроить.

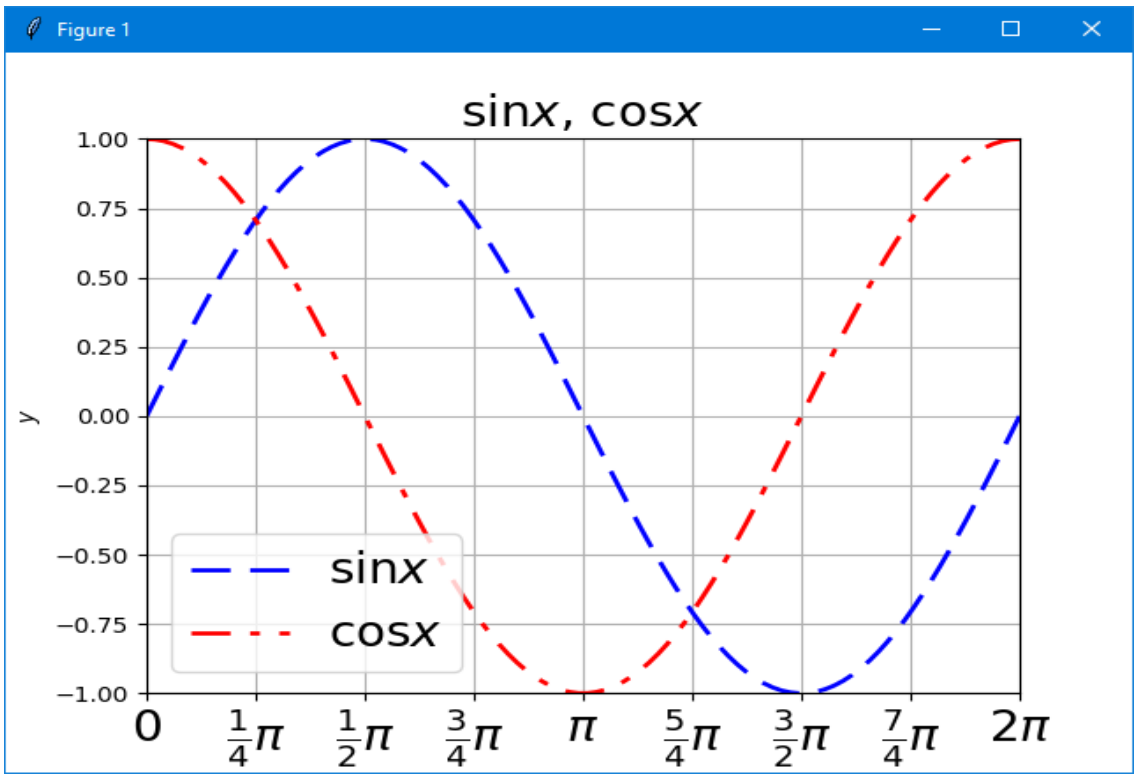
Можно задать последовательность засечек на оси $x$ (и $y$) и подписи к ним (в них, как и в других текстах, можно использовать *LaTEX*-овские обозначения).

Задать подписи осей *x* и *y* и заголовок графика.

Во всех текстовых элементах можно задать размер шрифта. Можно задать толщину линий и штрихи (так, на графике косинуса рисуется штрих длины 8, потом участок длины 4 не рисуется, потом участок длины 2 рисуется, потом участок длины 4 опять не рисуется, и так по циклу; поскольку толщина линии равна 2, эти короткие штрихи длины 2 фактически выглядят как точки).

Можно задать подписи к кривым (legend); где разместить эти подписи тоже можно регулировать

```
import matplotlib.pyplot as plt
import numpy as np
plt.axis([0,2*np.pi,-1,1])

plt.xticks(np.linspace(0,2*np.pi,9),
       ('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'$\frac{7}{4}\pi$',r'$2\pi$'),
      fontsize=20)

plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.title(r'$\sin x$, $\cos x$',fontsize=20)

x=np.linspace(0,2*np.pi,100)
plt.plot(x,np.sin(x),linewidth=2,
                    color='b',dashes=[8,4],
                                label=r'$\sin x$')
plt.plot(x,np.cos(x),linewidth=2,
                   color='r',dashes=[8,4,2,4],
                                label=r'$\cos x$')
plt.legend(fontsize=20)
plt.grid(True)
# визуализируем графики
plt.show()
```

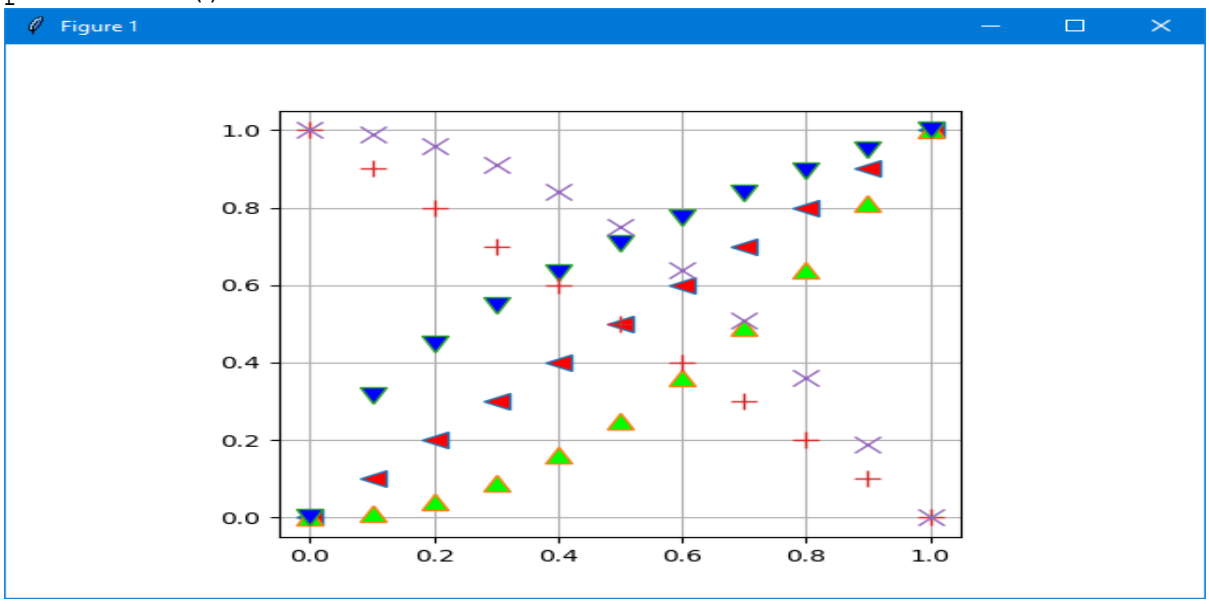



## П13. Модифицированные маркеры

Если `linestyle=''`, то точки не соединяются линиями. Сами точки рисуются маркерами разных типов. Тип определяется строкой из одного символа, который чем-то похож на нужный маркер. В добавок к стандартным маркерам, можно определить самодельные.

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(0,1,11)
# установим параметр - aspect(ratio)
axx=plt.subplot(111)
axx.set_aspect(1)
plt.axis([-0.05,1.05,-0.05,1.05])
plt.plot(x,x,linestyle='',marker='<',markersize=10,
        markerfacecolor='#FF0000')
plt.plot(x,x**2,linestyle='',marker='^',markersize=10,
        markerfacecolor='#00FF00')
plt.plot(x,x**(1/2),linestyle='',marker='v',markersize=
10,
        markerfacecolor='#0000FF')
plt.plot(x,1-x,linestyle='',marker='+',markersize=10,
        markerfacecolor='#0F0F00')
plt.plot(x,1-
x**2,linestyle='',marker='x',markersize=10,
     markerfacecolor='#0F000F')
```

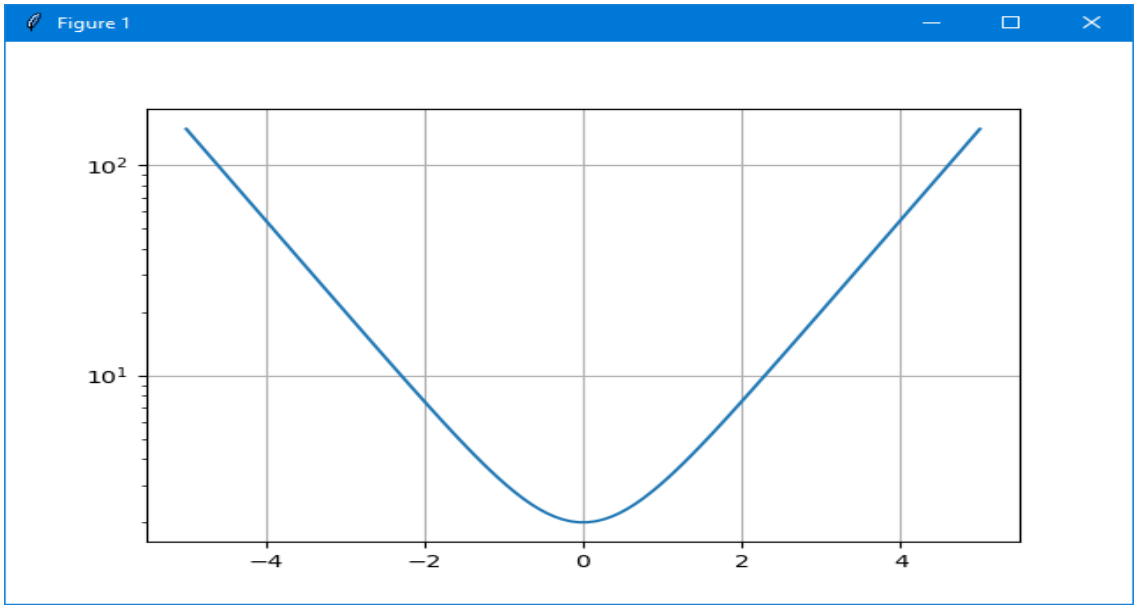
```
plt.grid(True)
# визуализируем графики
plt.show()
```




## П14. Логарифмический масштаб

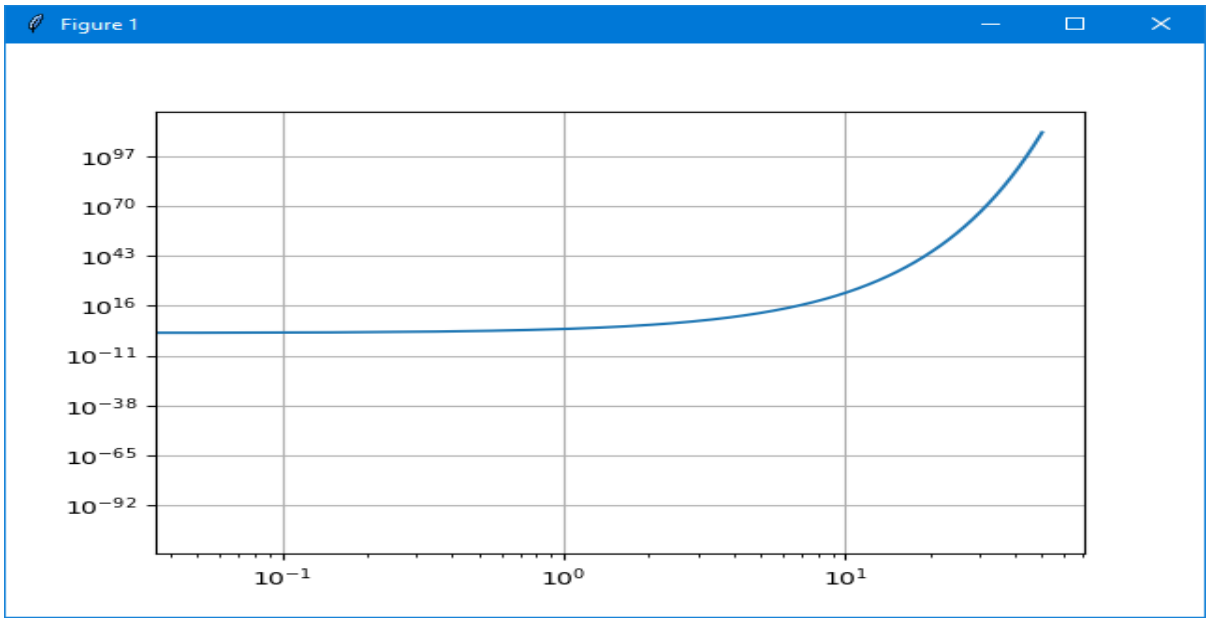
Если **y** меняется на много порядков, то удобно использовать логарифмический масштаб по **y**

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-5,5,100)
plt.yscale('log')
plt.plot(x,np.exp(x)+np.exp(-x))
plt.grid(True)
# визуализируем графики
plt.show()
```



Можно задать логарифмический масштаб по обоим осям.

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-50,50,1000)
plt.xscale('log')
plt.yscale('log')
plt.plot(x,np.exp(5*x+3))
plt.grid(True)
# визуализируем графики
plt.show()
```




## П13. Экпериментальные данные

Допустим, имеется теоретическая кривая (резонанс без фона).

```
xt=np.linspace(-4,4,101)
yt=1/(xt**2+1)
```

Поскольку реальных экспериментальных данных под рукой нет, мы их сгенерируем. Пусть они согласуются с теорией, и все статистические ошибки равны 0.1.

```
xe=np.linspace(-3,3,21)
yerr=0.1*np.ones(21)
ye=1/(xe**2+1)+yerr*np.random.normal(size=21)
```

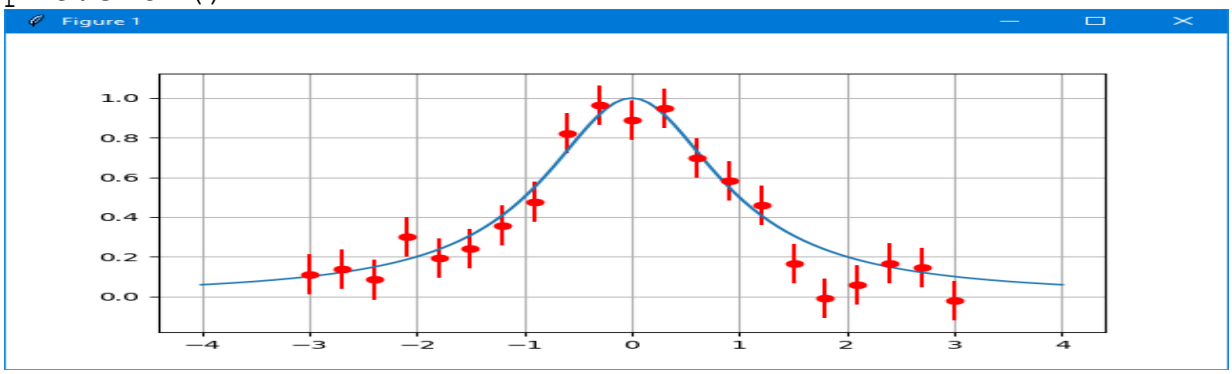
***Экспериментальные точки с усами*** и теоретическая кривая на одном графике.

```
plt.plot(xt,yt)
plt.errorbar(xe,ye,fmt='ro',yerr=yerr)
```

Весь скрипт:

```
import math
```

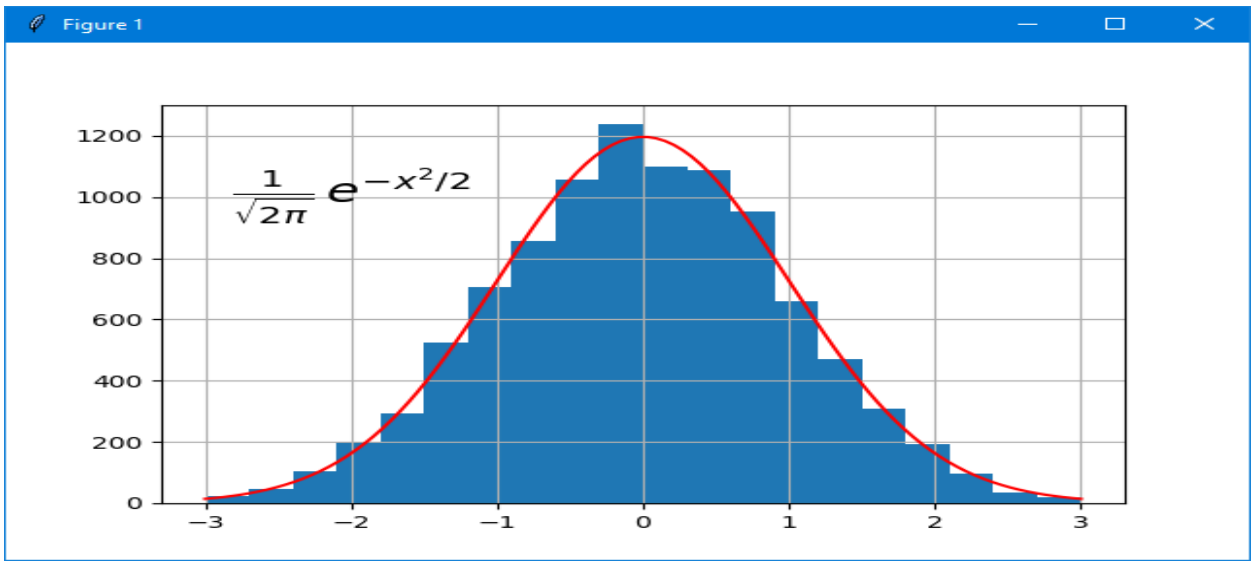
```
import numpy as np
import matplotlib.pyplot as plt
xt=np.linspace(-4,4,101)
yt=1/(xt**2+1)
xe=np.linspace(-3,3,21)
yerr=0.1*np.ones(21)
ye=1/(xe**2+1)+yerr*np.random.normal(size=21)
plt.plot(xt,yt)
plt.errorbar(xe,ye,fmt='ro',yerr=yerr)
plt.grid(True)
# визуализируем графики
plt.show()
```




## П15. Гистограмма

Сгенерируем $N$ случайных чисел с нормальным (гауссовым) распределением (среднее 0, среднеквадратичное отклонение 1), и раскидаем их по 20 бинам от −3 до 3 (точки за пределами этого интервала отбрасываются). Для сравнения, вместе с гистограммой нарисуем Гауссову кривую в том же масштабе. И даже напишем формулу Гаусса.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as ml
N=10000
r=np.random.normal(size=N)
n,bins,patches=plt.hist(r,range=(-3,3),bins=20)
x=np.linspace(-3,3,100)
plt.plot(x,N/np.sqrt(2*np.pi)*0.3*
                              np.exp(-0.5*x**2),'r')
plt.text(-2,1000,r'$\frac{1}{\sqrt{2\pi}}
                                      \,e^{-x^2/2}$',
     fontsize=20,horizontalalignment='center',
     verticalalignment='center')
plt.grid(True)
# визуализируем графики
plt.show()
```

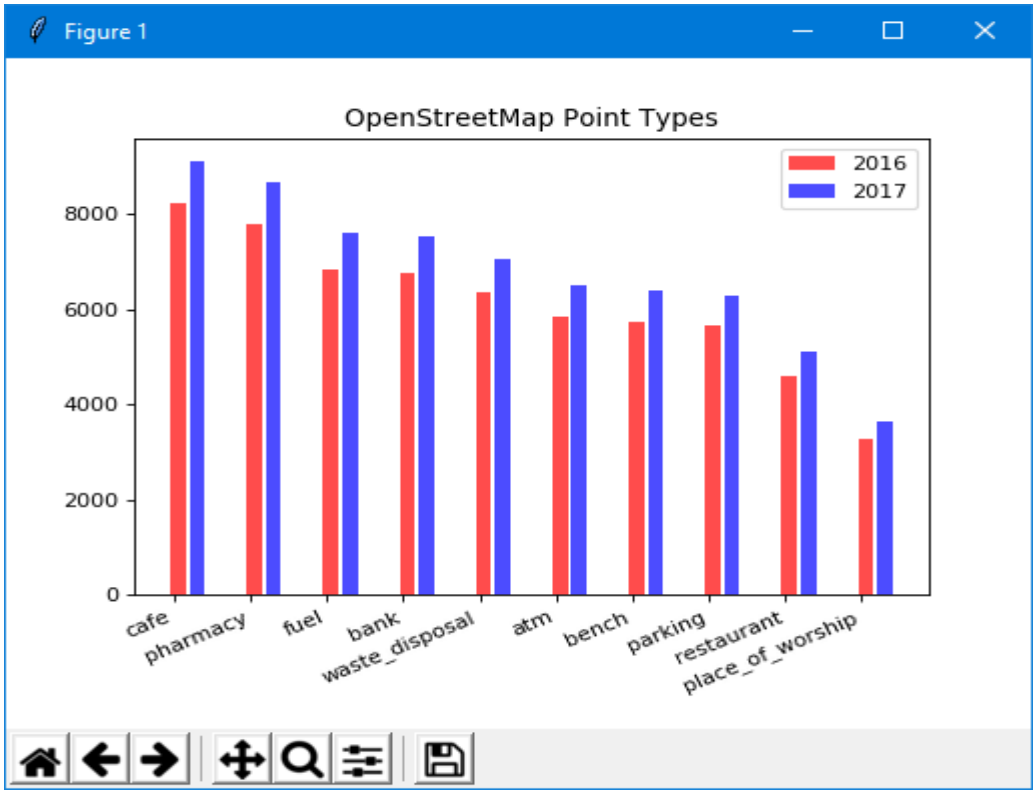



Рассмотрим несколько примеров (https://eax.me/python-matplotlib/) построения специальных диаграммы.

В качестве примера построим диаграмму, отображающую, сколько точек на карте к какому типу (заправка, кафе и так далее) относятся. Чтобы было чуть интереснее, сделаем вид, что в прошлом году точек каждого вида было на 10% меньше, и попытаемся отразить это изменение:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import datetime as dt
import csv
data_names = ['cafe', 'pharmacy', 'fuel', 'bank',
'waste_disposal',
              'atm', 'bench', 'parking', 'restaurant',
              'place_of_worship']
data_values = [9124, 8652, 7592, 7515, 7041, 6487,
6374, 6277,
               5092, 3629]
dpi = 80
fig = plt.figure(dpi = dpi, figsize = (512 / dpi, 384 /
dpi) )
mpl.rcParams.update({'font.size': 10})
plt.title('OpenStreetMap Point Types')
#ax = plt.axes()
#ax.yaxis.grid(True, zorder = 1)
xs = range(len(data_names))
plt.bar([x + 0.05 for x in xs], [ d * 0.9 for d in
                                    data_values],
        width = 0.2, color = 'red', alpha = 0.7,
                                   label = '2016',
        zorder = 2)
```

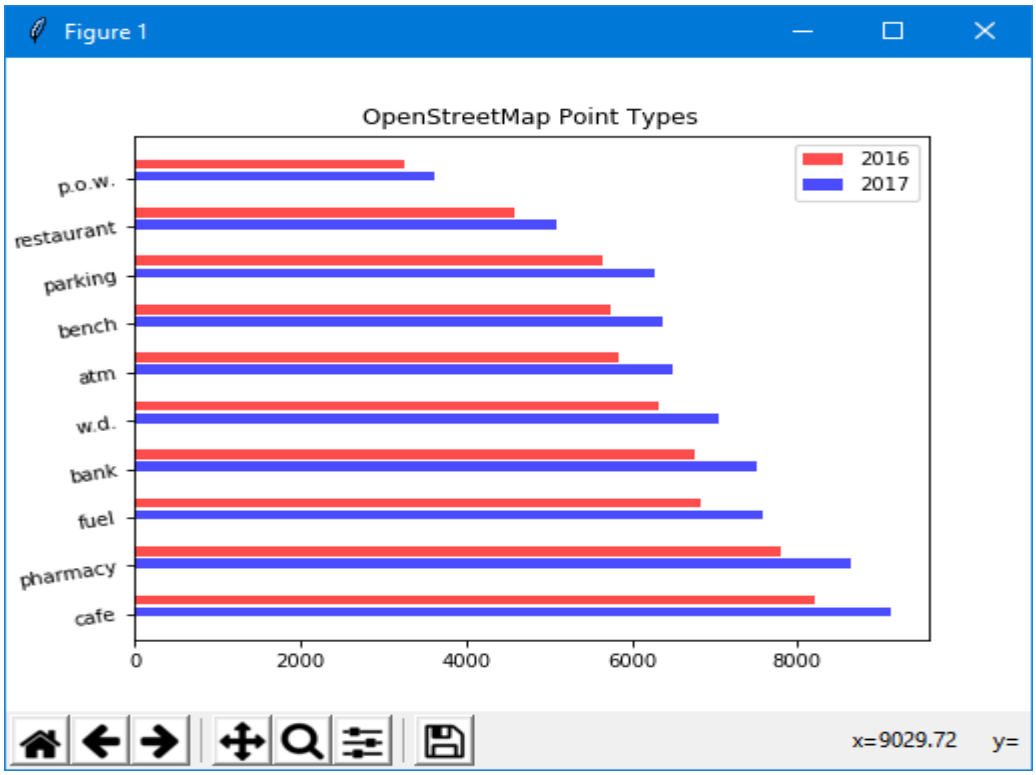
```
plt.bar([x + 0.3 for x in xs], data_values,
        width = 0.2, color = 'blue', alpha = 0.7,
                                        label = '2017',
        zorder = 2)
plt.xticks(xs, data_names)
fig.autofmt_xdate(rotation = 25)
plt.legend(loc='upper right')
#fig.savefig('bars.png')
plt.show()
```




Те же данные можно отобразить, расположив столбики горизонтально:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import datetime as dt
import csv

data_names = ['cafe', 'pharmacy', 'fuel', 'bank',
'w.d.', 'atm','bench', 'parking', 'restaurant',
                                          'p.o.w.']
data_values = [9124, 8652, 7592, 7515, 7041, 6487,
6374, 6277,  5092, 3629]

dpi = 80
fig = plt.figure(dpi = dpi, figsize =
                              (512 / dpi, 384 / dpi) )
```

```
mpl.rcParams.update({'font.size': 9})
plt.title('OpenStreetMap Point Types')
#ax = plt.axes()
#ax.xaxis.grid(True, zorder = 1)
xs = range(len(data_names))
plt.barh([x + 0.3 for x in xs], [ d * 0.9 for d in
data_values],
         height = 0.2, color = 'red', alpha = 0.7,
label = '2016',
         zorder = 2)
plt.barh([x + 0.05 for x in xs], data_values,
         height = 0.2, color = 'blue', alpha = 0.7,
label = '2017',
         zorder = 2)
plt.yticks(xs, data_names, rotation = 10)

plt.legend(loc='upper right')
plt.show()
```




## П16. Круговая диаграмма

Для примера визуализируем распределение кафе по различным городам (пример взят с https://eax.me/python-matplotlib/):

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
```

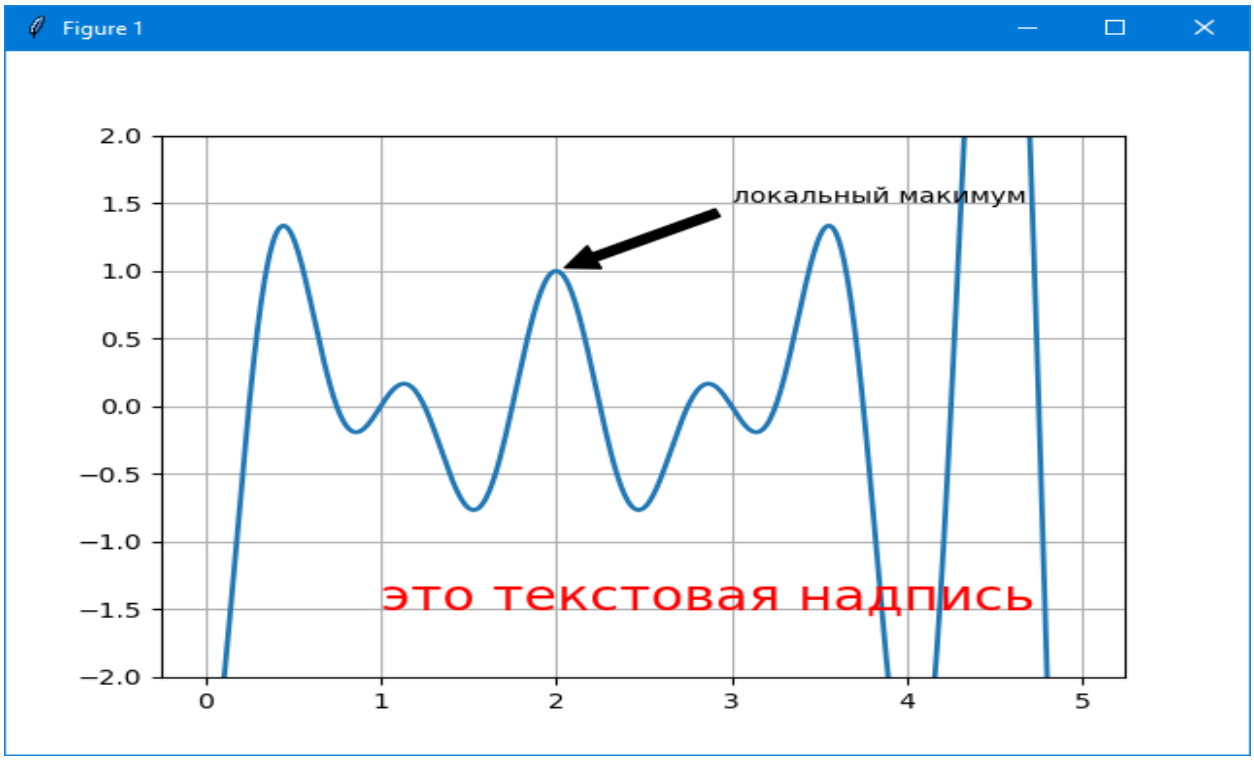
```
import datetime as dt
import csv

data_names = ['Москва', 'Санкт-Петербург', 'Сочи',
'Архангельск',
              'Владимир', 'Краснодар', 'Курск',
'Воронеж',
              'Ставрополь', 'Мурманск']
data_values = [1076, 979, 222, 189, 137, 134, 124, 124,
91, 79]

dpi = 80
fig = plt.figure(dpi = dpi, figsize = (512 / dpi, 384 /
dpi) )
mpl.rcParams.update({'font.size': 9})

plt.title('Распределение кафе по городам России (%)')

xs = range(len(data_names))

plt.pie(
    data_values, autopct='%.1f', radius = 1.1,
    explode = [0.15] + [0 for _ in
range(len(data_names) - 1)] )
plt.legend(
    bbox_to_anchor = (-0.16, 0.45, 0.25, 0.25),
    loc = 'lower left', labels = data_names )
plt.show()
```

Огромное количество примеров можно найти на https://matplotlib.org/gallery.html

## П17. Текст и надписи

Текст и дополнительные надписи (аннатации) размещаются на диаграмме:

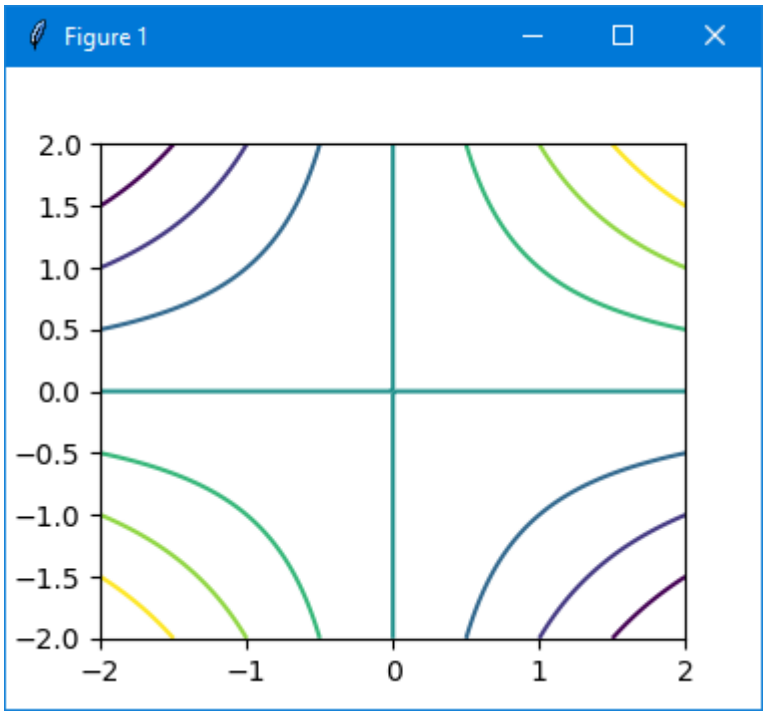
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = (1-(t-2)*(t-2))  * np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)
plt.text(1,-1.5,'это текстовая надпись', size=20,
color='r')
plt.grid(True)
plt.annotate('локальный макимум', xy=(2, 1),
                              xytext=(3, 1.5),
                arrowprops=
                dict(facecolor='black', shrink=0.05), )
plt.ylim(-2,2)
plt.show()
```



# П18. Контурные графики

Пусть необходимо хотим изучить поверхность **z=xy.** Построим её «горизонтали»
(взять с http://www.inp.nsk.su/~grozin/python/python6.html):

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
x=np.linspace(-1,1,50)
y=x
z=np.outer(x,y)
plt.contour(x,y,z)
plt.axes().set_aspect(1)
plt.show()
```

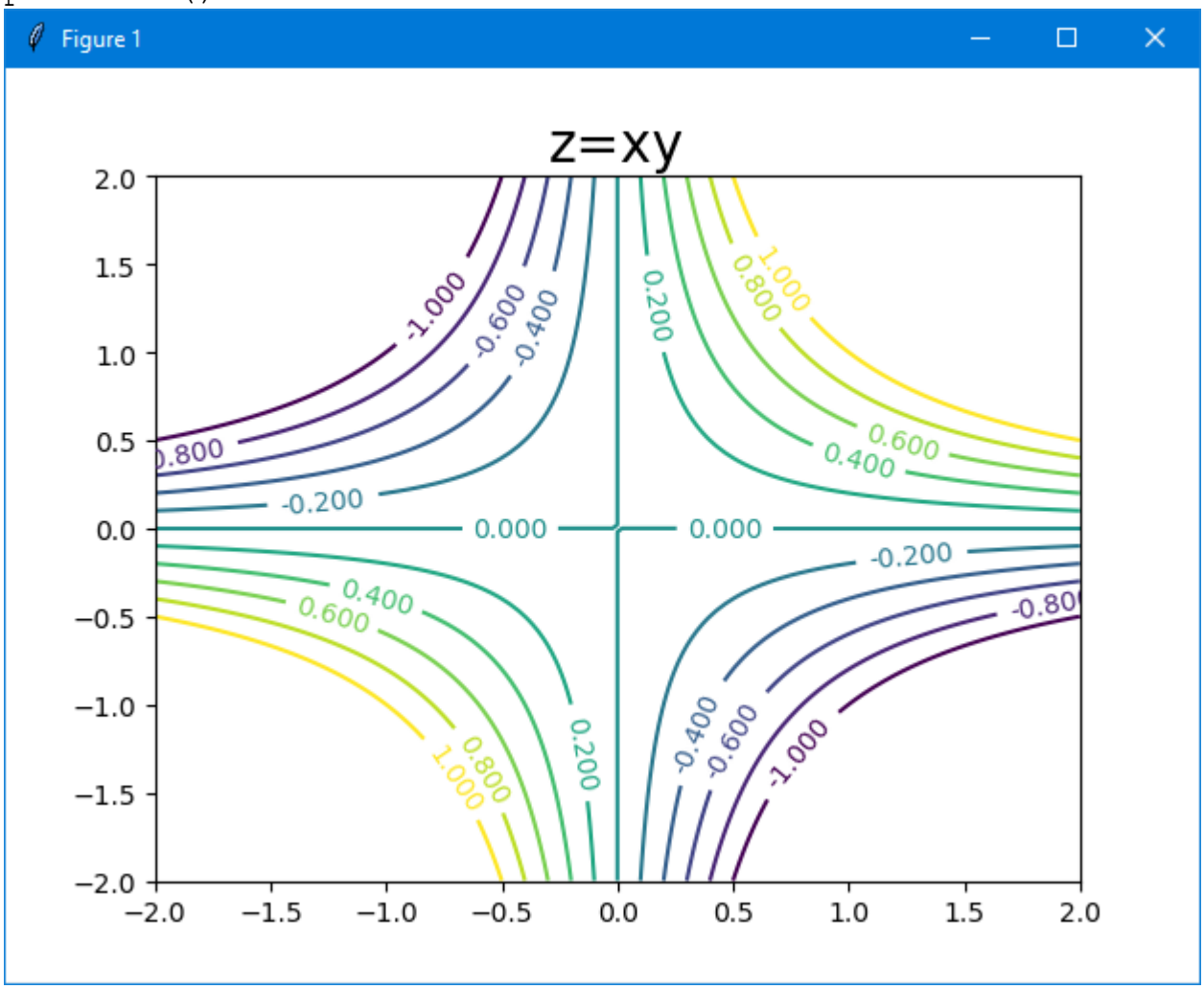



Роль функции **np.outer**(x,y) пакета numpy очевидна из примера:

```
>>> x=np.array([1,2,3])
>>> y=np.array([10,20,30])
>>> x*y
array([10, 40, 90])
>>> np.outer(x,y)
array([[10, 20, 30],
       [20, 40, 60],
       [30, 60, 90]])
```

Горизонтали можно раскрасить и подписать, а так же увеличить их число:

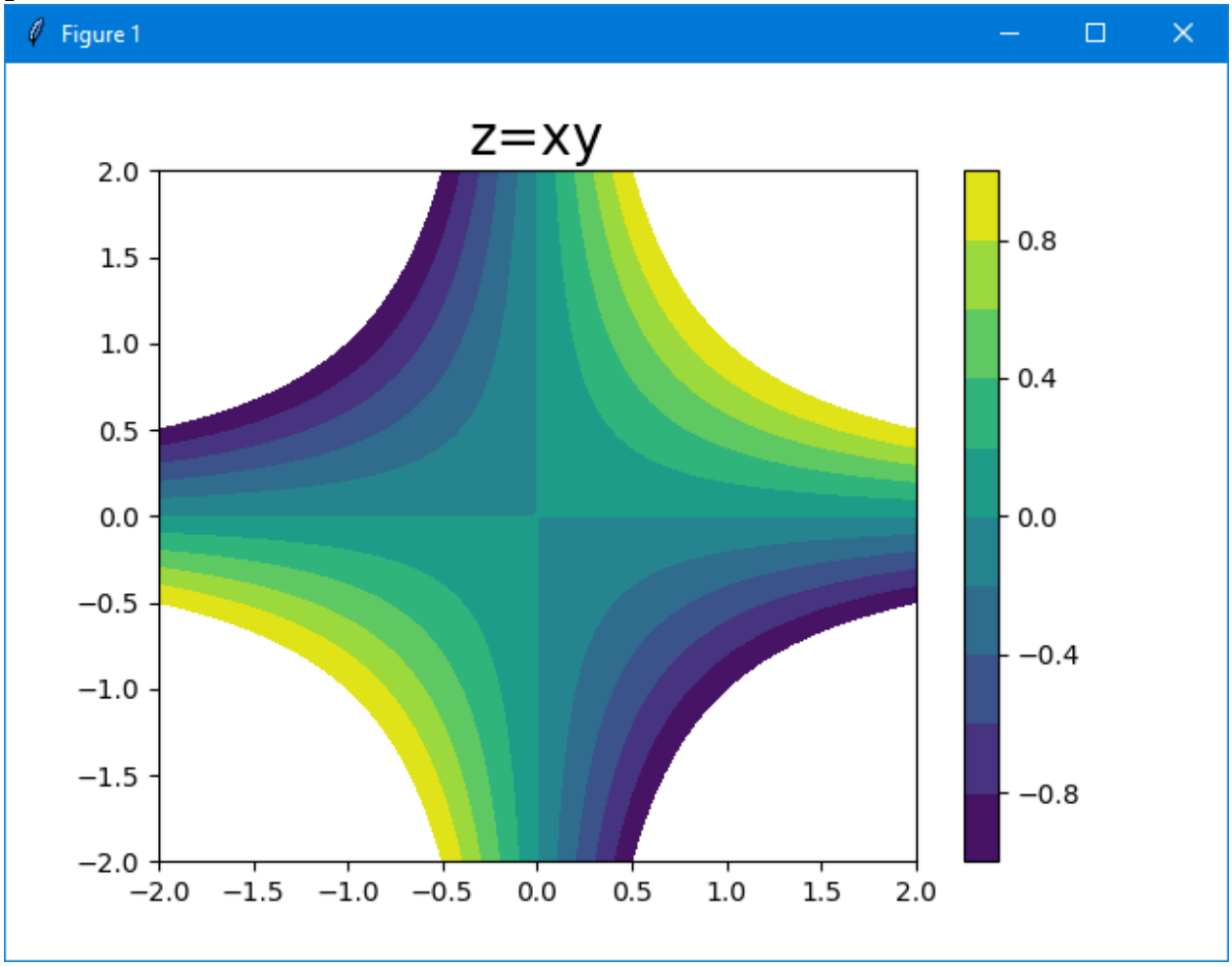
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
x=np.linspace(-2,2,100)
y=x
z=np.outer(x,y)
plt.title('z=xy',fontsize=20)
curves=plt.contour(x,y,z,np.linspace(-1,1,11))
plt.clabel(curves)
plt.show()
```




Высоту (значение Z) можно задать цветом, как на физических географических картах. `colorbar` показывает соответствие цветов и значений *z*.

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
x=np.linspace(-2,2,100)
y=x
```

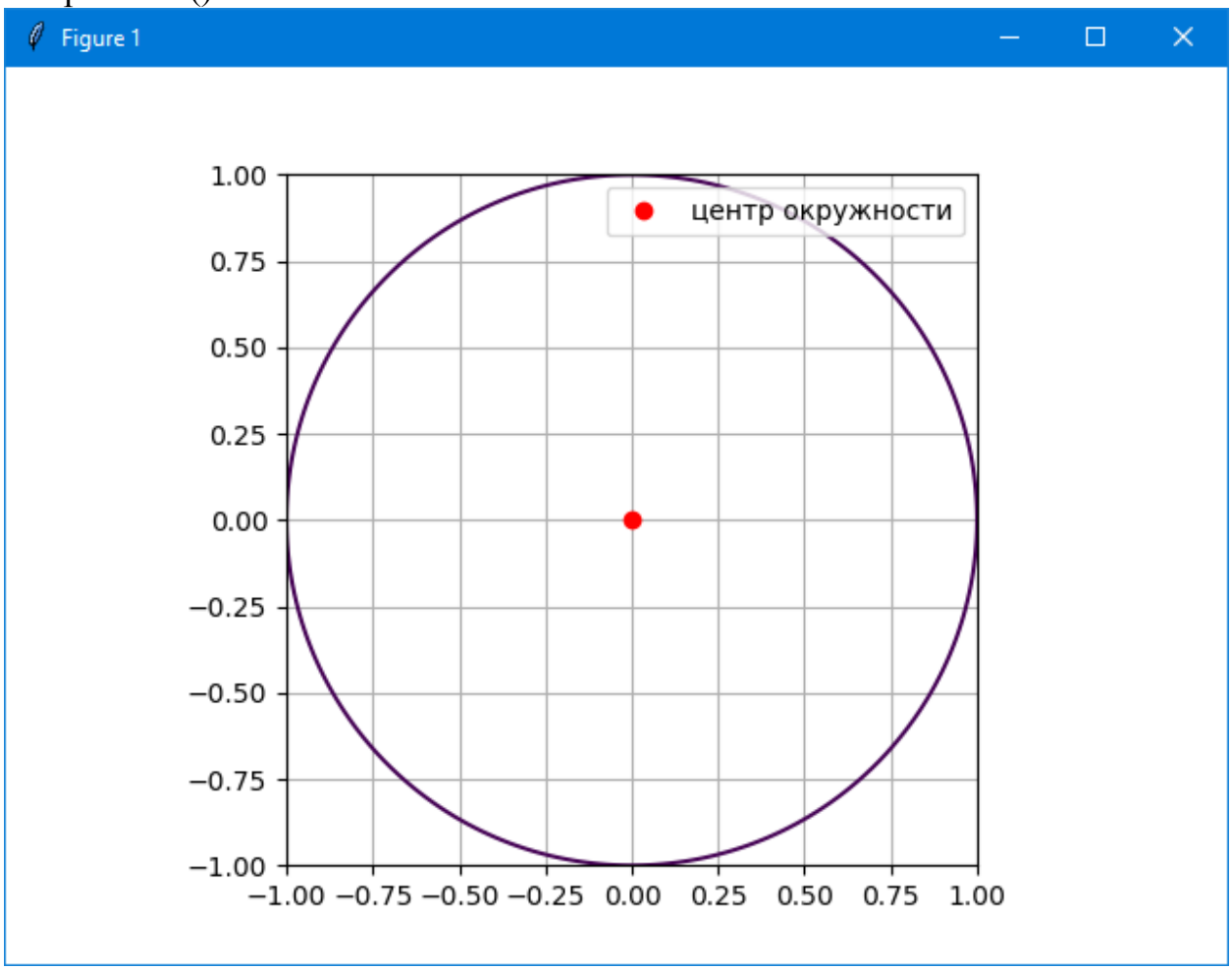
```
z=np.outer(x,y)
plt.title('z=xy',fontsize=20)
plt.contourf(x,y,z,np.linspace(-1,1,11))
plt.colorbar()
#plt.axes().set_aspect(1)
plt.show()
```




Так можно «рисовать» графики функций на плоскости, заданные «неявно» - F(x,y)=0:

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

x = np.linspace(-1.0, 1.0, 100)
y = np.linspace(-1.0, 1.0, 100)
X, Y = np.meshgrid(x,y)
F = X**2 + Y**2 - 1  #0.6
plt.contour(X,Y,F,[0])
plt.plot([0],[0],'ro', label="центр окружности")
plt.gca().set_aspect('equal') #, чтобы рисунок выглядело кругом

# Включаем сетку
plt.grid(True)
```

```
plt.legend(loc='best')
plt.show()
```




## П19. Images (пиксельные картинки)

Картинка задаётся массивом `z`: `z[i,j]` - это цвет пикселя `i,j`, массив из 3 элементов (`r g b`, числа от 0 до 1). Для наглядности в примере формируются «натуральные» цвета в список col:

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

n=8
u=np.linspace(0,1,n)
x,y=np.meshgrid(u,u)
z=np.zeros((n,n,3))

col=[]
for r in range(0,2):
    for g in range(0,2):
        for b in range(0,2):
```

```
            print(r,g,b)
            col.append([r,g,b])
#print (col)

for i in range(n):
    for j in range(n):     #  R G B
        z[i,j]=col[i]

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