**Работа с библиотекой Matplotlib**

**Теоретическая часть**

**Контурные диаграммы**

В научном мире часто встречается тип контурных графиков или контурных карт. Они подходят для представления трехмерных поверхностей с помощью контурных карт, составленных из кривых, которые, в свою очередь, являются точками на поверхности на одном уровне (с одним и тем же значением z).

Во-первых, нужна функция *z = f(x, y)* для генерации трехмерной поверхности. Затем, имея значения x и y, определяющие площадь карты для вывода, можно вычислять значения *z* для каждой пары *x* и *y* с помощью функции *f(x, y)*, которая и используется для этих целей. Благодаря функции *contour()* можно сгенерировать контурную карту на поверхности. Также часто требуется добавить цветную карту. Это площади, ограниченные кривыми уровней, заполненные цветным градиентом. Например, следующее изображения показывает отрицательные значения с помощью темных оттенков синего, а приближение к желтому и красному указывает на более высокие значения.

dx = 0.01; dy = 0.01

x = np.arange(-2.0,2.0,dx)

y = np.arange(-2.0,2.0,dy)

X,Y = np.meshgrid(x,y)

def f(x,y):

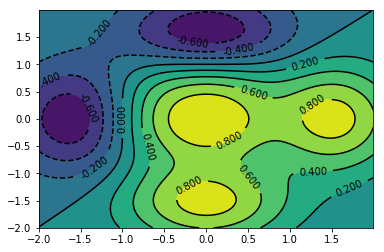
return (1 - y\*\*5 + x\*\*5)\*np.exp(-x\*\*2-y\*\*2)

C = plt.contour(X,Y,f(X,Y),8,colors='black')

plt.contourf(X,Y,f(X,Y),8)

plt.clabel(C, inline=1, fontsize=10)

plt.show()



Пример реализации диаграмма двух переменных:

from scipy import stats

means = (0.5, 0.9)

covar = [

[1., 0.6],

[0.6, 1.]

]

mvn = stats.multivariate\_normal(means, covar)

x, y = np.meshgrid(

np.linspace(-3, 3, 80),

np.linspace(-2, 4, 80)

)

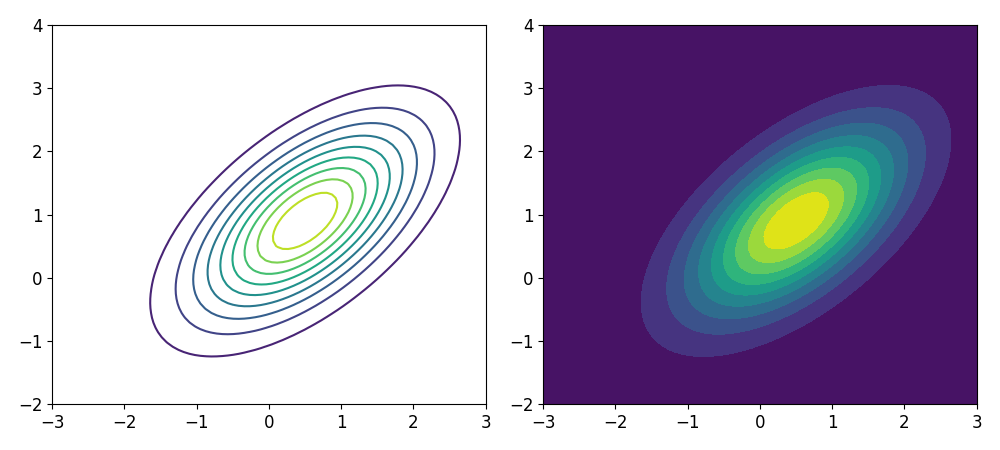
data = np.dstack((x, y))

*# левая диаграмма — без заливки цветом*

plt.contour(x, y, mvn.pdf(data), levels=10)

*# правая диаграмма — с заливкой цветом*

plt.contourf(x, y, mvn.pdf(data), levels=10)



Аргумент levels задает количество контуров. По умолчанию контуры отрисовываются равномерно между максимальным и минимальным значениями. В аргумент levels также можно передать список уровней, на которых следует провести контуры.

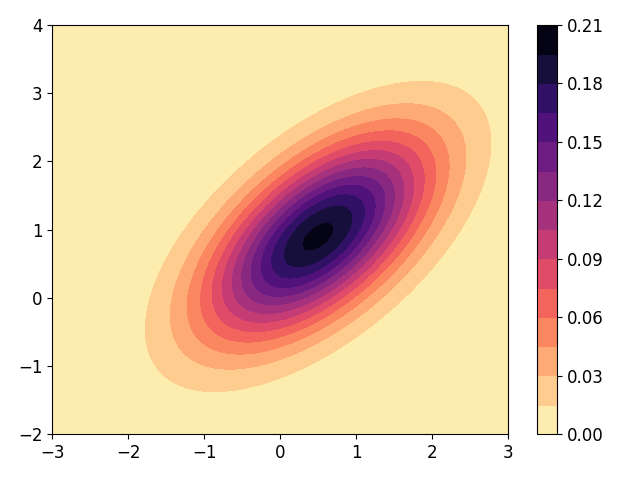
Обратите внимание на использование функций numpy.meshgrid и numpy.dstack в этом примере.

Контурную диаграмму можно дополнить цветовой полосой colorbar, вызвав функцию pyplot.colorbar:

cs = plt.contourf(x, y, mvn.pdf(data), levels=15,

cmap=matplotlib.cm.magma\_r)

cbar = plt.colorbar(cs)



**Расположение нескольких осей в одном окне**

В одном окне (объекте Figure) можно разместить несколько осей (объектов axis.Axis). Функция pyplot.subplots создает объект Figure, содержащий регулярную сетку объектов axis.Axis:

{% raw %}

import numpy as np

from scipy import stats

import matplotlib.pyplot as plt

fig, axes = plt.subplots(ncols=3, nrows=2, /media//media/figsize=(12, 8))

x = np.linspace(0.01, 25, 250)

for idx, row in enumerate(axes):

for jdx, ax in enumerate(row):

ndf = idx \* 3 + jdx + 1

y = stats.chi2.pdf(x, ndf)

ax.plot(x, y, label=fr'$\chi^2\_*{*{ndf={ndf}}*}*(x)$')

ax.set\_xlabel(r'$x$', fontsize=16)

ax.set\_ylim([0, 1.05\*y.max()])

ax.minorticks\_on()

ax.legend(fontsize=16)

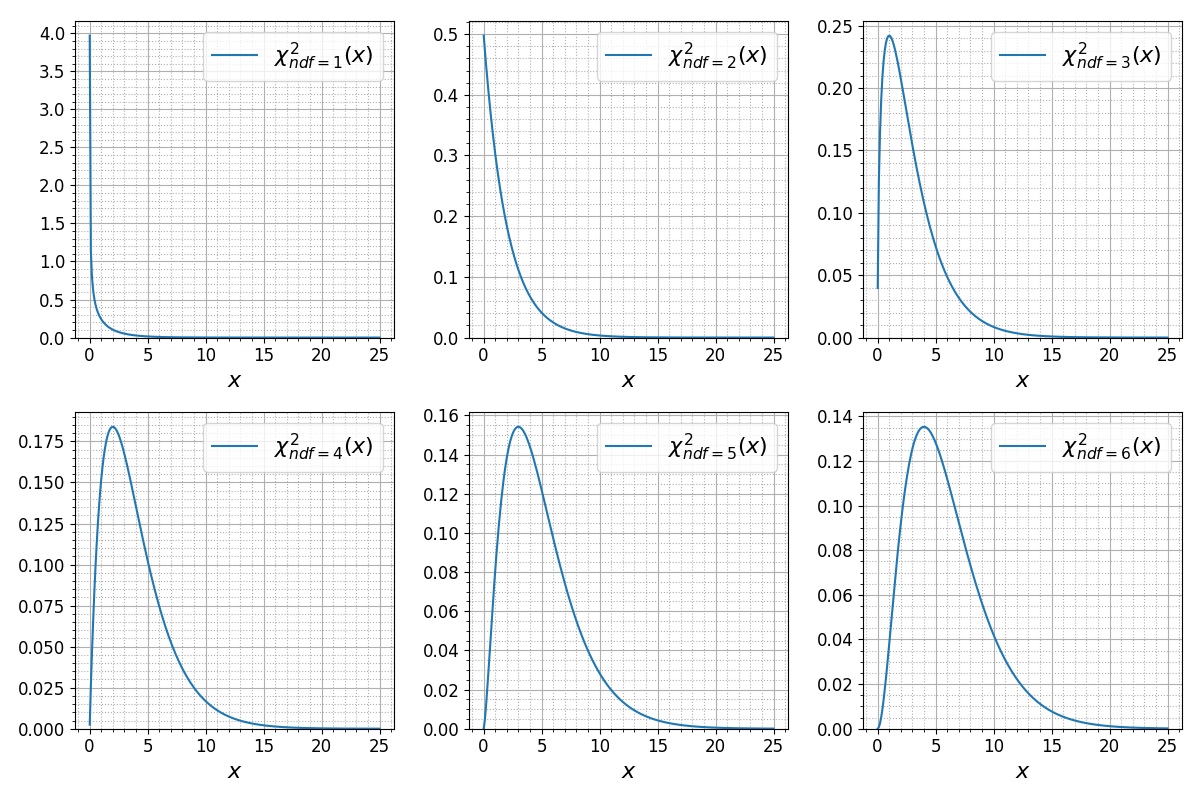
ax.grid(which='major')

ax.grid(which='minor', linestyle=':')

fig.tight\_layout()

plt.show()

{% endraw %}



Количество строк и столбцов, по которым располагаются различные оси, задаются с помощью параметров nrows и ncols, соответственно. Функция pyplot.subplots возвращает объект Figure и двумерный список осей axis.Axis. Обратите внимание на то, что вместо вызовов функций модуля pyplot в этом примере использовались вызовы методов классов Figure и axis.Axis.

В последнем примере горизонтальная ось во всех графиках имеет один и тот же диапазон. Аргумент sharex функции pyplot.subplots позволяет убрать дублирование отрисовки осей в таких случаях:

fig, axes = plt.subplots(ncols=3, nrows=2, /media//media/figsize=(12, 8),

sharex=**True**)

*# ...*

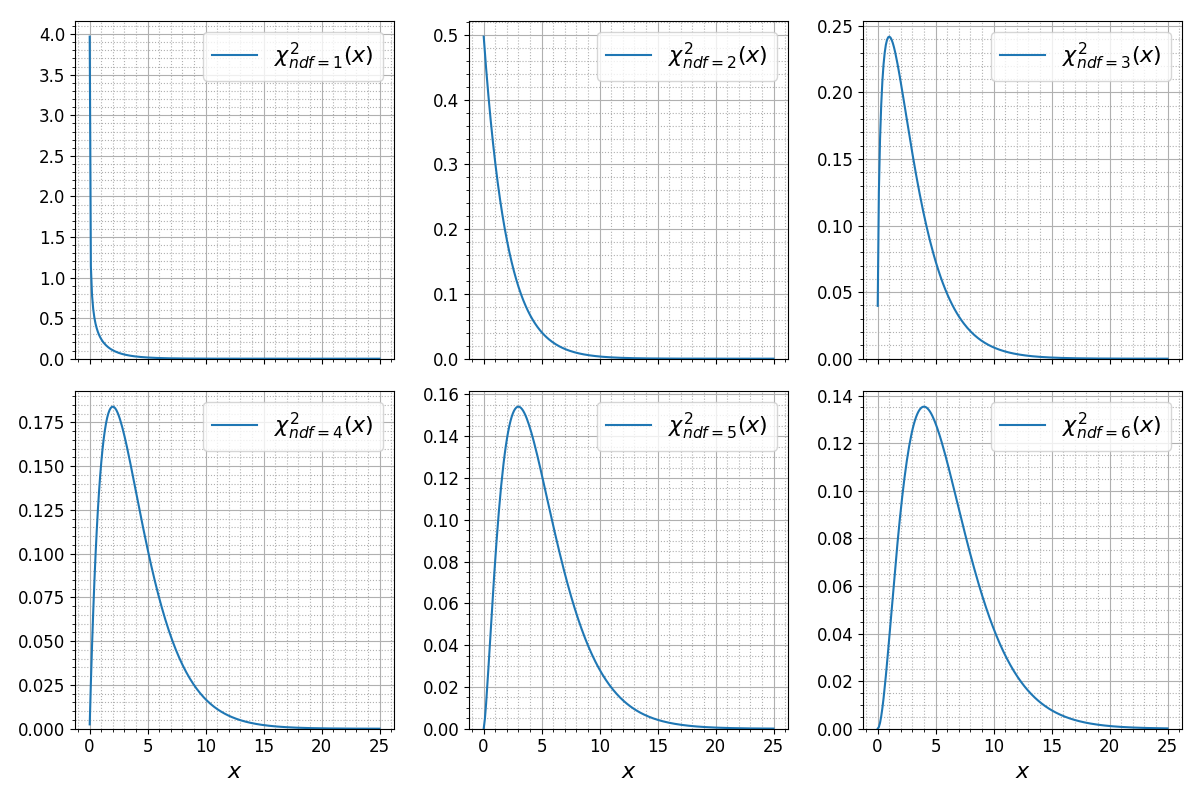
**for** idx, row **in** enumerate(axes):

**for** jdx, ax **in** enumerate(row):

*# ...*

**if** idx:

ax.set\_xlabel(r'$x$', fontsize=16)



Существует аналогичный параметр sharey для вертикальной оси.

Более гибкие возможности регулярного расположения осей предоставляет функция pyplot.subplot.

Функция pyplot.axes позволяет добавлять новые оси в текущем окне в произвольном месте:

import numpy as np

import matplotlib.pyplot as plt

exno = 26

rg = np.random.Generator(np.random.PCG64(5))

x1 = rg.exponential(10, 5000)

x2 = rg.normal(10, 0.1, 100)

*# Строим основную гистограмму*

plt.hist([x1, x2], bins=150, range=(0, 60), stacked=True)

plt.minorticks\_on()

plt.xlim((0, 60))

plt.grid(which='major')

plt.grid(which='minor', linestyle=':')

*# Строим вторую гистограмму в отдельных осях*

plt.axes([.5, .5, .4, .4])

plt.hist([x1, x2], bins=100, stacked=True, range=(9, 11))

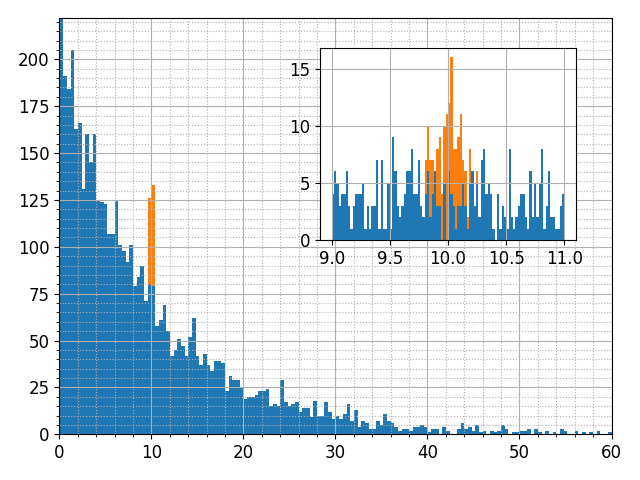
plt.grid(which='major')

plt.tight\_layout()

*# сохраняем диаграмму в файл*

plt.savefig('histograms.png')

plt.show()



В этом примере была использована функция pyplot.savefig, сохраняющая содержимое текущего окна в файл в векторном или растровом формате. Формат задается с помощью аргумента format или автоматически определяется из имени файла (как в примере выше). Набор доступных форматов зависит от окружения, однако в большинстве случаев можно использовать такие форматы как png, jpeg, pdf, svg и eps.

## **Лепестковые диаграммы**

Этот вид диаграмм представляет собой набор круговых секторов, каждый из которых занимает определенный угол. Таким образом можно изобразить два значения, присвоив их величинам, обозначающим диаграмму: радиус r и угол θ. На самом деле, это полярные координаты — альтернативный способ представления функций на осях координат. С графической точки зрения такая диаграмма имеет свойства круговой и столбчатой. Как и в круговой диаграмме каждый сектор здесь указывает на значение в процентах по отношению к целому. Как и в столбчатой, расширение по радиусу — это числовое значение категории.

Есть возможность использовать любую последовательность цветов. Нужно лишь определить список строковых значений, содержащих RGB-коды в формате #rrggbb, которые будут соответствовать нужным цветам.

Странно, но для вывода лепестковой диаграммы нужно использовать функцию bar() и передать ей список углов θ и список радиальных расширений для каждого сектора.

N = 8

theta = np.arange(0.,2 \* np.pi, 2 \* np.pi / N)

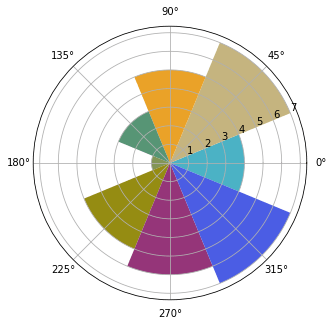
radii = np.array([4,7,5,3,1,5,6,7])

plt.axes([0.025, 0.025, 0.95, 0.95], polar=True)

colors = np.array(['#4bb2c5','#c5b47f','#EAA228','#579575','#839557','#958c12','#953579','#4b5de4'])

bars = plt.bar(theta, radii, width=(2\*np.pi/N), bottom=0.0, color=colors)

plt.show()



В этом примере определена последовательность цветов в формате #rrggbb, но ничто не мешает использовать строки с реальными названиями цветов.

N = 8

theta = np.arange(0.,2 \* np.pi, 2 \* np.pi / N)

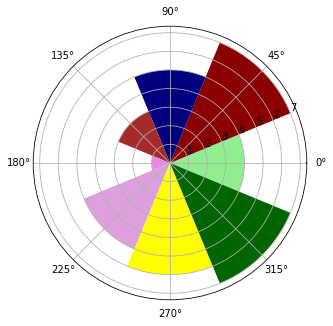
radii = np.array([4,7,5,3,1,5,6,7])

plt.axes([0.025, 0.025, 0.95, 0.95], polar=True)

colors = np.array(['lightgreen','darkred','navy','brown','violet','plum','yellow','darkgreen'])

bars = plt.bar(theta, radii, width=(2\*np.pi/N), bottom=0.0, color=colors)

plt.show()



**Задание (с использованием библиотеки pyplot)**

1. Построить контурную диаграмму для функции двух переменных:

|  |  |
| --- | --- |
| Нечетные варианты | Четные варианты |
|  |  |
| где *n* – последняя цифра в логине | |

Контурные диаграммы нарисовать с цветовой полосой.

Количество уровней – *n*.

Диаграммы построить с заливкой цветом и без нее.

1. Построить на одном рисунке графики функций. Слева – в указанном масштабе, справа – масштаб больше в 5 раз.

|  |  |
| --- | --- |
| Нечетные варианты | Четные варианты |
|  |  |
|  |  |
| где *n* – последняя цифра в логине | |

1. Построить лепестковую диаграмму
   1. показывающую произвольную «розу ветров» по 8 направлением. Выделить цветом различные направления.
   2. По данным:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Юг | Шереметьево | Зименки | Подлипки | Раскольниково | Водянка |
| 90 | 85 | 45 | 75 | 56 | 35 |