Лабораторная работа №3

Визуализация данных с помощью matplotlib. Создание графиков с использованием «научной графики».

Contents

Щью

[Введение 2](#_Toc146144962)

[Архитектура matplotlib 2](#_Toc146144963)

[Расширенные возможности построения графиков 8](#_Toc146144964)

[Демо-пример 9](#_Toc146144965)

[Два графика на диаграмме в индивидуальных масштабах 13](#_Toc146144966)

[Несколько графиков в одном и в разных графических окнах 16](#_Toc146144967)

[figure() и axes() 17](#_Toc146144968)

[Установить свой диапазон осей 20](#_Toc146144969)

[Перенос координатных осей в центр графика 20](#_Toc146144970)

[Надписи на окнах диаграмм и окнах Windows, изменение размера окна 23](#_Toc146144971)

[Параметрический график 25](#_Toc146144972)

[Полярные координаты 27](#_Toc146144973)

[График рассеяния 28](#_Toc146144974)

[Настройки в стиле LATEX 30](#_Toc146144975)

[Модифицированные маркеры 32](#_Toc146144976)

[Экспериментальные данные 33](#_Toc146144977)

[Гистограмма 35](#_Toc146144978)

[Круговая диаграмма 38](#_Toc146144979)

[Текст и надписи 39](#_Toc146144980)

[Контурные графики 40](#_Toc146144981)

[Images (пиксельные картинки) 43](#_Toc146144982)

[Построение 3D графиков. Работа с mplot3d Toolkit 45](#_Toc146144983)

[Линейный график 45](#_Toc146144984)

[Каркасная поверхность 46](#_Toc146144985)

[Поверхность, залитая цветом 47](#_Toc146144986)

[Задание 48](#_Toc146144987)

[Источник: 49](#_Toc146144988)

**Цель работы:** изучить и применить основные принципы визуализации данных с использованием библиотеки **matplotlib**, а также овладеть навыками создания научных графиков для наглядного представления и анализа научных данных. В рамках лабораторной работы студенты должны освоить различные типы графиков, настроить их параметры для максимальной информативности, и применить полученные навыки к конкретным данным, обеспечивая их наглядное и понятное представление.

# Теоретические сведения и ход работы:

## Введение

**matplotlib**-набордополнительныхмодулей(библиотек)языкаPython.

Предоставляетсредствадляпостроениясамыхразнообразных2Dграфикови

диаграммданных.Достоинства этой библиотеки -простотаиспользования.

Для построения весьма мудреных и красочно оформленных диаграмм достаточно нескольких строк кода. При этом качество получаемых изображений

более чем достаточно для их опубликования. Работа этого модуля обычно

подразумевает так же использование модуля NumPy.

Перед началом работы необходимо установить библиотеку **matplotlib** (python -m pip install matplotlib) либо проверить ее наличие.

## Архитектура matplotlib

Одна из основных задач, которую выполняет **matplotlib** — предоставление набора функций и инструментов для представления и управления **Figure** (так называется основной объект) вместе со всеми внутренними объектами, из которого он состоит. Но в **matplotlib** есть также инструменты для обработки событий и, например, анимации. Благодаря им эта библиотека способна создавать интерактивные графики на основе событий по нажатию кнопки или движению мыши.

Архитектура matplotlib логически разделена на три слоя, расположенных на трех уровнях. Коммуникация непрямая — каждый слой может взаимодействовать только с тем, что расположен под ним, но не над.

Вот эти слои:

·           Слой сценария

·           Художественный слой

·           Слой бэкенда

**Слой бэкенда**

Слой Backend является нижним на диаграмме с архитектурой всей библиотеки. Он содержит все API и набор классов, отвечающих за реализацию графических элементов на низком уровне.

·        **FigureCanvas** — это объект, олицетворяющий область рисования.

·        **Renderer** — объект, который рисует по FigureCanvas.

·        **Event** — объект, обрабатывающий ввод от пользователя (события с клавиатуры и мыши)

**Художественный слой**

Средним слоем выступает художественный (artist). Все элементы, составляющие график, такие как название, метки осей, маркеры и так далее, являются экземплярами этого объекта. Каждый из них играет свою роль в иерархической структуре.

Есть два художественных класса: примитивный и составной.

·           **Примитивный** — это объекты, которые представляют собой базовые элементы для формирования графического представления графика, например, Line2D, или геометрические фигуры, такие как прямоугольник круг или даже текст.

·           **Составные** — объекты, состоящие из нескольких базовых (примитивных). Это оси, шкалы и диаграммы.

На этом уровне часто приходится иметь дело с объектами, занимающими высокое положение в иерархии: график, система координат, оси. Поэтому важно полностью понимать, какую роль они играют. Ниже перечислены три основных художественных (составных объекта), которые часто используются на этом уровне.

·           **Figure** — объект, занимающий верхнюю позицию в иерархии. Он соответствует всему графическому представлению и может содержать много систем координат.

·           **Axes** — это тот самый график. Каждая система координат принадлежит только одному объекту Figure и имеет два объекта Axis (или три, если речь идет о трехмерном графике). Другие объекты, такие как название, метки x и y, принадлежат отдельно осям.

·           **Axis** учитывает числовые значения в системе координат, определяет пределы и управляет обозначениями на осях, а также соответствующим каждому из них текстом. Положение шкал определяется объектом Locator, а внешний вид — Formatter.

**Слой сценария (pyplot)**

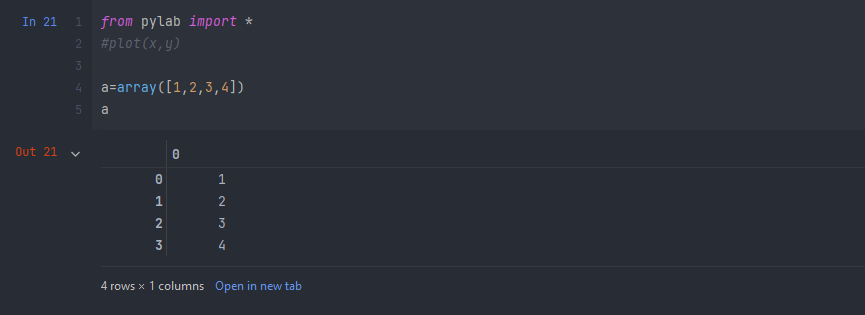
Художественные классы и связанные с ними функции (API matplotlib) подходят всем разработчикам, особенно тем, кто работает с серверами веб-приложений или разрабатывает графические интерфейсы. Но для вычислений, в частности для анализа и визуализации данных, лучше всего подходит слой сценария. Он включает интерфейс pyplot.

**pylab и pyplot**

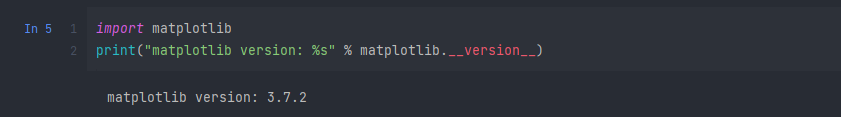
С точки зрения пользователя существуют две библиотеки: **pylab** и **pyplot**. **Pylab** — это модуль, устанавливаемый вместе с matplotlib, а **pyplot** — внутренний модуль matplotlib. На оба часто ссылаются в скриптах:

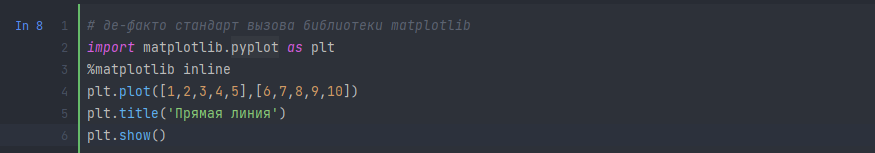
**from pylab import \***

Pylab объединяет функциональность pyplot с возможностями NumPy в одном пространстве имен, поэтому отдельно импортировать NumPy не нужно. Более того, при импорте pylab функции из pyplot и NumPy можно вызывать без ссылки на модуль (пространство имен).

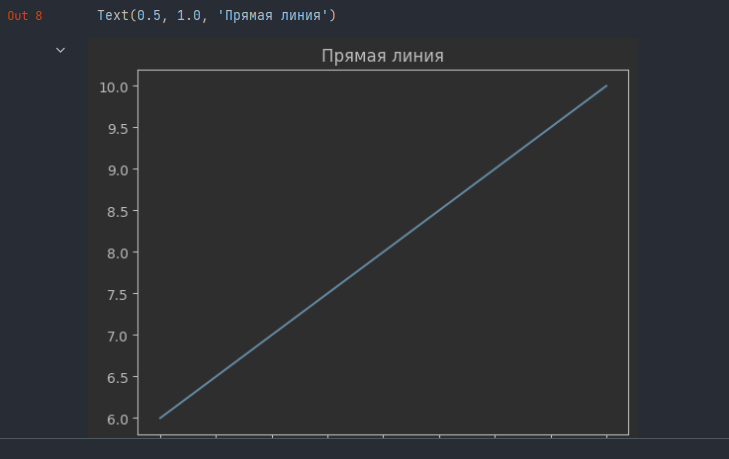


**Проверка версии matplotlib:**





**Результат:**



Модуль **pyplot** содержит функции (похожие на команды) создания

диаграмм и изменения свойств их элементов.

Функция **plot()** строит прямоугольные двумерные диаграммы (графики) в

координатах X - Y.

Если функции plot передан один аргумент (например – список

 [1,2,3]), она рассматривает его как совокупность значений

откладываемых по оси Y, тогда по оси X ему будет соответствовать

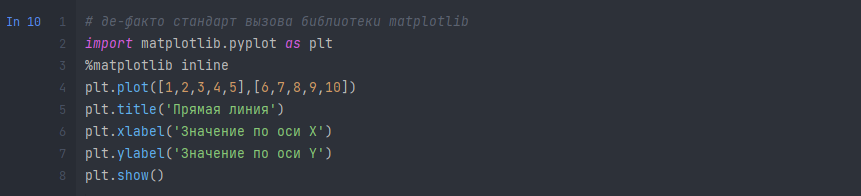
автоматически сгенерированный набор чисел 0,1,2...N - 1, где N - число

элементов в переданном списке.

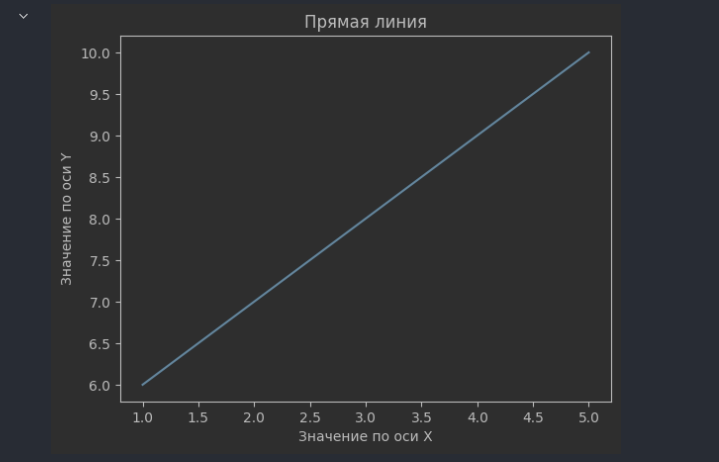
Функция **title()** задает заголовок диаграммы, а функция **show()**

выводит интерактивное окно диаграммы при работе в консоли.

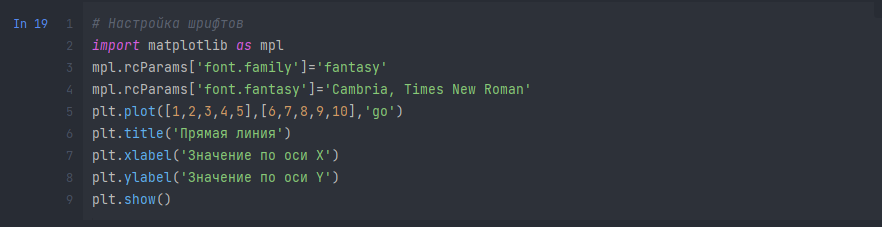
**Добавление подписей к осям**

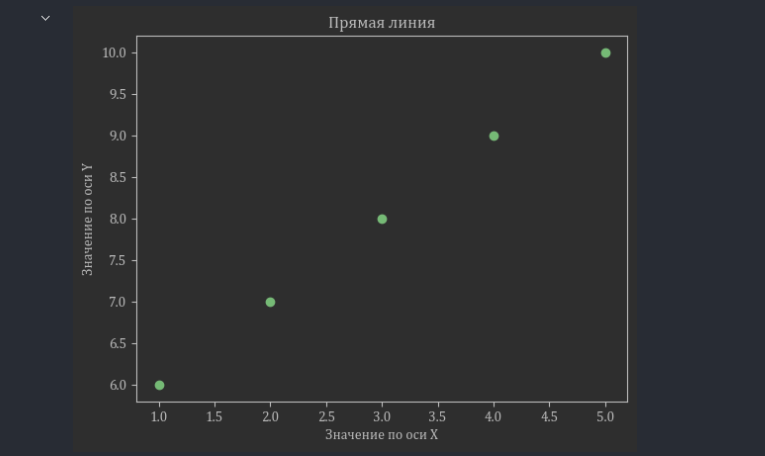


**Результат**:

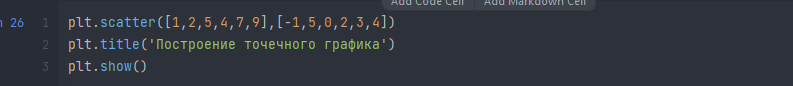


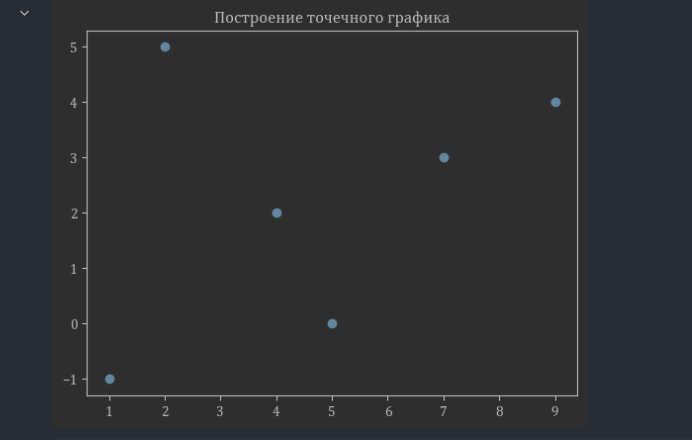
**Настройка шрифтов**





**Построение точечного графика**





## Расширенные возможности построения графиков

Следующий код строит несколько графиков в одном окне и сохраняет результат в графический файл .png

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

*# независимая переменная*x = np.linspace(-4\*np.pi, 4\*np.pi, 100)

*# название осей*xlab='время сек'  
ylab='амплитуда сигнала'

*# первый график*y1= np.sin(2\*x)\*np.exp(-0.1\*x)

*# легенда*leg1='exp(-0.1\*t)\*sin(2\*t)'

*# второй график*y2=10\*np.sin(0.5\*x)+np.cos(10\*x)  
leg2='смешанный сигнал'

*# третий график*y3=np.sin(x)  
leg3='опорный сигнал'

*# Включаем сетку по оси X и оси Y.  
# Задаем цвет толщину сетки*plt.grid(color = 'b',linewidth = 1)

*# Задаем подписи к осям X и оси Y и размер шрифта*plt.xlabel(xlab, fontsize = 'x-large')  
plt.ylabel(ylab, fontsize = 'x-large')

*# Задаем заголовок диаграммы*plt.title('Гармонические сигналы ')

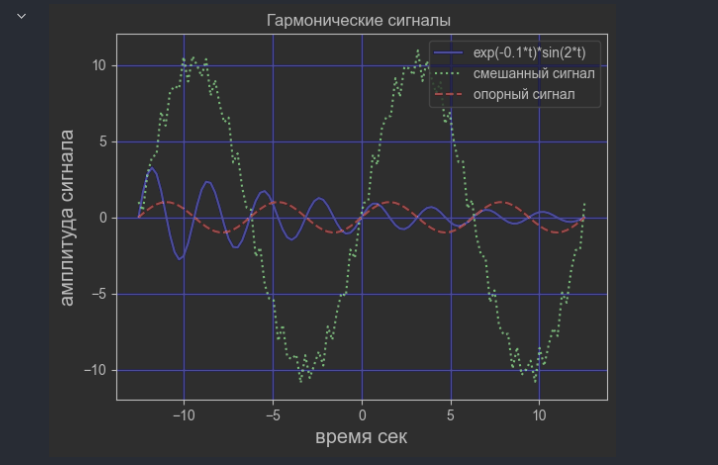
*# строим графики*plt.plot(x,y1,'b-',label=leg1)  
plt.plot(x,y2,'g:',label=leg2)  
plt.plot(x,y3,'r--',label=leg3)

*# задаем вывод легенды и ее расположение*plt.legend(loc='best')

*# Включаем сетку*plt.grid(*True*)

*# Сохраняем построенную диаграмму в файл  
# Задаем имя файла и его тип*plt.savefig('signal3.png', format = 'png')  
*# визуализируем графики*plt.show()

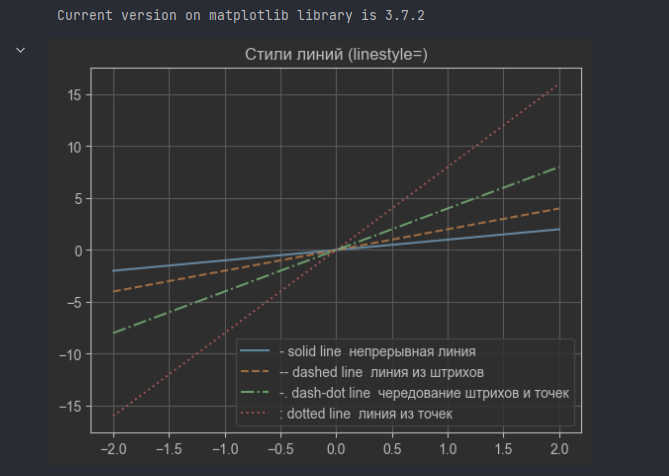
Результат:

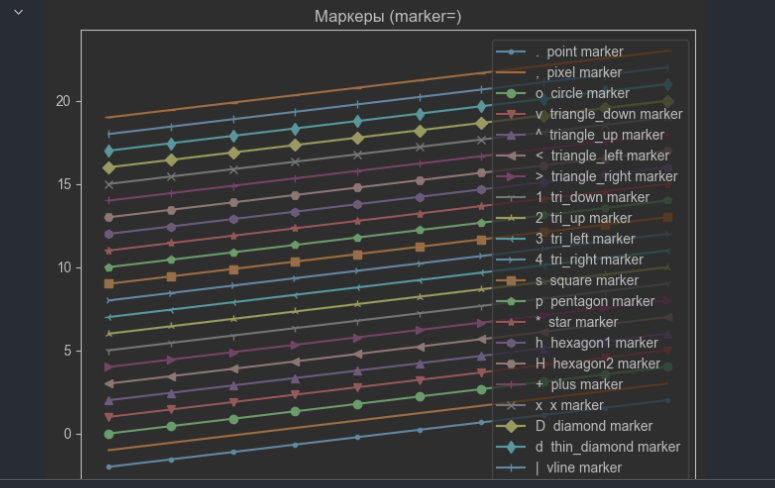


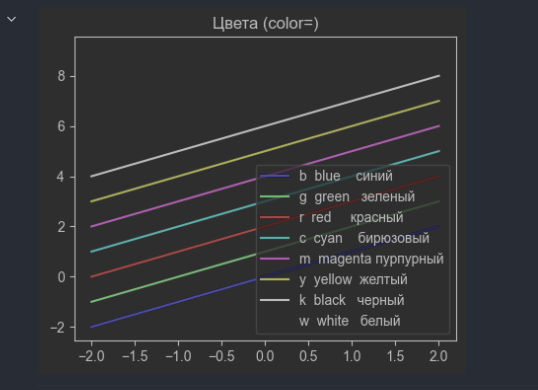
## Демо-пример

*import* sys  
*import* math  
*import* os  
*import* matplotlib.pyplot *as* plt  
*import* numpy *as* np  
*##sys.exit(0)  
import* matplotlib *as* mpl  
*# Вывод на экран текущей версии библиотеки matplotlib*print ('Current version on matplotlib library is',mpl.\_\_version\_\_)  
x = np.linspace(-2, 2, 10)  
y1=1\*x  
y2=2\*x  
y3=4\*x  
y4=8\*x  
s1= ['-', 'solid line ', 'непрерывная линия']  
s2= ['--', 'dashed line ', 'линия из штрихов']  
s3= ['-.', 'dash-dot line ', 'чередование штрихов и точек']  
s4= [':', 'dotted line ', 'линия из точек']  
leg1=s1[0]+' '+s1[1]+' '+s1[2]  
leg2=s2[0]+' '+s2[1]+' '+s2[2]  
leg3=s3[0]+' '+s3[1]+' '+s3[2]  
leg4=s4[0]+' '+s4[1]+' '+s4[2]  
  
ax=plt.subplot(111)  
*#можно поменть размеры окна графика в граф. окне*box=ax.get\_position()  
ax.set\_position([box.x0, box.y0,box.width\*1.0 , box.height\*1.0])  
p1=plt.plot(x,y1,linestyle=s1[0], label=leg1)  
p2=plt.plot(x,y2,linestyle=s2[0], label=leg2)  
p3=plt.plot(x,y3,linestyle=s3[0], label=leg3)  
p4=plt.plot(x,y4,linestyle=s4[0], label=leg4)  
  
plt.title('Стили линий (linestyle=)')  
*#ax.legend(loc=(1.0,0.5), mode='expand' )  
#ax.legend(mode='expand', bbox\_to\_anchor=(1, 0.05),loc #='upper center' ) #left best*ax.legend(loc='lower right')  
*# Включаем сетку*plt.grid()  
plt.show()  
*#sys.exit(0)  
# marker*plt.close()  
mar=[  
'.', 'point marker',  
',', 'pixel marker',  
'o', 'circle marker',  
'v', 'triangle\_down marker',  
'^', 'triangle\_up marker',  
'<', 'triangle\_left marker',  
'>', 'triangle\_right marker',  
'1', 'tri\_down marker',  
'2', 'tri\_up marker',  
'3', 'tri\_left marker',  
'4', 'tri\_right marker',  
's', 'square marker',  
'p', 'pentagon marker',  
'\*', 'star marker',  
'h', 'hexagon1 marker',  
'H', 'hexagon2 marker',  
'+', 'plus marker',  
'x', 'x marker',  
'D', 'diamond marker',  
'd', 'thin\_diamond marker',  
'|', 'vline marker',  
'\_', 'hline marker'  
]  
leg=[]  
*for* i *in* range(0,len(mar),2):  
 leg.append(mar[i]+' '+mar[i+1])  
mpl.rcParams['figure.figsize'] = (8.0, 6.0)  
kk=int( len(mar)/2 )  
*for* i *in* range(kk):  
 plt.plot(x, x+i, marker=leg[i][0], label=leg[i])  
plt.legend(loc='lower right')  
plt.title('Маркеры (marker=)')  
plt.show()  
*#color*col=[  
'b', 'blue синий',  
'g', 'green зеленый',  
'r', 'red красный',  
'c', 'cyan бирюзовый',  
'm', 'magenta пурпурный',  
'y', 'yellow желтый',  
'k', 'black черный',  
'w', 'white белый'  
]  
leg=[]  
*for* i *in* range(0,len(col),2):  
 leg.append(col[i]+' '+col[i+1])  
mpl.rcParams['figure.figsize'] = (5.0, 4.0)  
kk=int( len(col)/2 )  
*for* i *in* range(kk):  
 plt.plot(x, x+i, color=leg[i][0], label=leg[i])  
plt.legend(loc='lower right')  
plt.title('Цвета (color=)')  
plt.show()

**Результат**:







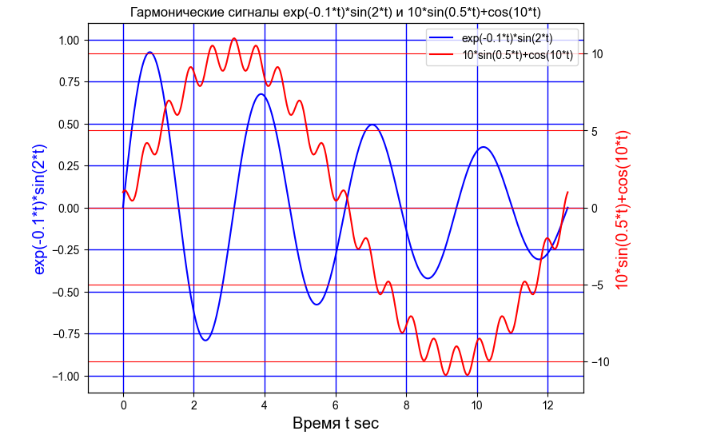
## Два графика на диаграмме в индивидуальных масштабах

Обратитевниманиена«запятую»послеименипеременной

**line\_01,**=ax\_01.plot(X,Y\_01,'b-')

*# -\*- coding: UTF-8 -\*-  
  
import* numpy *as* np  
  
*import* matplotlib *as* mpl  
  
*import* matplotlib.pyplot *as* plt  
  
*# Значения по оси X время, t*X = np.linspace(0, 4\*np.pi, 400)  
  
*# Значения по осям Y\_01 & Y\_02*leg\_01='exp(-0.1\*t)\*sin(2\*t)'  
  
Y\_01 = np.sin(2\*X)\*np.exp(-0.1\*X)  
  
*# "модулированный" сигнал*leg\_02="10\*sin(0.5\*t)+cos(10\*t)"  
  
Y\_02=10\*np.sin(0.5\*X)+np.cos(10\*X)  
  
*# Зададим размеры изображения диаграммы*mpl.rcParams['figure.figsize'] = (8.0, 6.0)  
  
*# Строим диаграмму  
  
# Получаем ссылку на объект типа matplotlib.axes.AxesSubplot, текущую диаграмму*ax\_01 = plt.axes()  
  
*# Задаем исходные данные для первой линии диаграммы  
# (линии степени разложения), внешний вид линии и  
# маркера.  
  
# Функция plot() возвращает ссылку на list,  
  
# первый элемент, которого есть объект класса  
# matplotlib.lines.Line2D*line\_01, = ax\_01.plot(X, Y\_01, 'b-') *#, label=leg\_01)  
  
# если надо в дальнейшем использовать объект класса  
# matplotlib.lines.Line2D  
  
# то применяют ДВА варианта  
  
# 1) line\_01, = ax\_01.plot(X, Y\_01, 'b-')  
# прием первого элемента списка  
  
# 2) line\_01 (без запятой)= … использовать line\_01[0]  
  
# Задаем интервалы значений по осям X и  
# основной оси Y*ax\_01.axis([-1, 13, -1.1, 1.1])  
  
*# Включаем сетку по оси X и основной оси Y.  
# Задаем цвет сетки*ax\_01.grid(color = 'b',linewidth = 1)  
  
*# Задаем подписи к осям X и основной оси Y  
# и размер фонта*ax\_01.set\_xlabel('Время t sec', fontsize = 'x-large')  
  
ax\_01.set\_ylabel(leg\_01, color = 'b',  
 fontsize = 'x-large')  
  
*# Задаем заголовок диаграммы*ax\_01.set\_title('Гармонические сигналы '+  
 leg\_01+" и "+ leg\_02)  
  
*#ax\_01.legend( loc='upper right')  
  
# Включаем дополнительную ось Y*ax\_02 = ax\_01.twinx()  
  
*# Задаем исходные данные для второй линии диаграммы,  
# внешний вид линии и маркера.  
  
# Функция plot() возвращает ссылку на объект класса  
# matplotlib.lines.Line2D (см выше)*line\_02, = ax\_02.plot(X, Y\_02, 'r-') *#, label=leg\_02)  
  
# Задаем интервалы значений по осям X  
# и дополнительной оси Y*ax\_02.axis([-1, 13, -12, 12])  
  
*# Задаем подпись к дополнительной оси Y*ax\_02.set\_ylabel(leg\_02,color='r',  
 fontsize = 'x-large')  
  
*# Задаем исходные данные для легенды и  
# место ее размещение*ax\_02.legend((line\_01, line\_02), (leg\_01, leg\_02),  
 loc = 'best')  
  
*#ax\_02.legend(loc='lower left')  
  
# Включаем сетку по дополнительной оси Y.  
# Задаем цвет сетки*ax\_02.grid(color = 'r')  
  
*# Сохраняем построенную диаграмму в файл  
  
# Задаем имя файла и его тип  
  
#plt.savefig('trigo.png', format = 'png')  
  
# визуализируем*plt.show()

**Результат:**



## Несколько графиков в одном и в разных графических окнах

Работас**matplotlib**основананаиспользованииграфическихокони

осей(осипозволяютзадатьнекоторуюграфическуюобласть).Все

построенияприменяютсяктекущимосям.Этопозволяетизображать

несколькографиковводномграфическомокне.

Поумолчаниюсоздаетсяоднографическоеокно**figure(1)**иодна

графическаяобласть**subplot(111)**вэтомокне.

Команда**subplot**позволяетразбитьграфическоеокнонанесколько

областей.Онаимееттрипараметра:**nr; nc; np**:

·        параметры**nr**и**nc**определяютколичествострокистолбцовна

которыеразбиваетсяграфическаяобласть

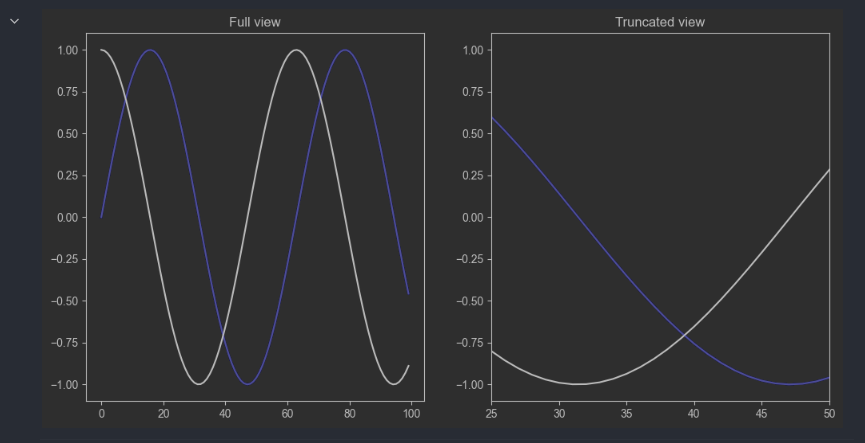
·        параметр**np**определяетномертекущейобласти(npпринимает

значенияот1доnr\*nc).

Если**nr\*nc<10**,топередаватьпараметры**nr,nc,np**можнобез

использованиязапятой.Например,допустимыформы**subplot(2,2,1)**и**subplot(221).**

*import* matplotlib.pyplot *as* plt  
*import* numpy *as* np  
  
fig = plt.figure(figsize=(12, 6))  
  
x = np.arange(0, 10, 0.1)  
y = np.sin(x)  
z = np.cos(x)  
  
ax = fig.add\_subplot(121)  
ax2 = fig.add\_subplot(122)  
  
ax.set\_title('Full view')  
ax.plot(y, color='blue', label='Sine wave')  
ax.plot(z, color='black', label='Cosine wave')  
  
ax2.set\_title('Truncated view')  
ax2.plot(y, color='blue', label='Sine wave')  
ax2.plot(z, color='black', label='Cosine wave')  
  
ax2.set\_xlim([25, 50])  
  
plt.show()



## figure() и axes()

Естьвозможностьрасположитьоси(подокна)вручную,напримернена

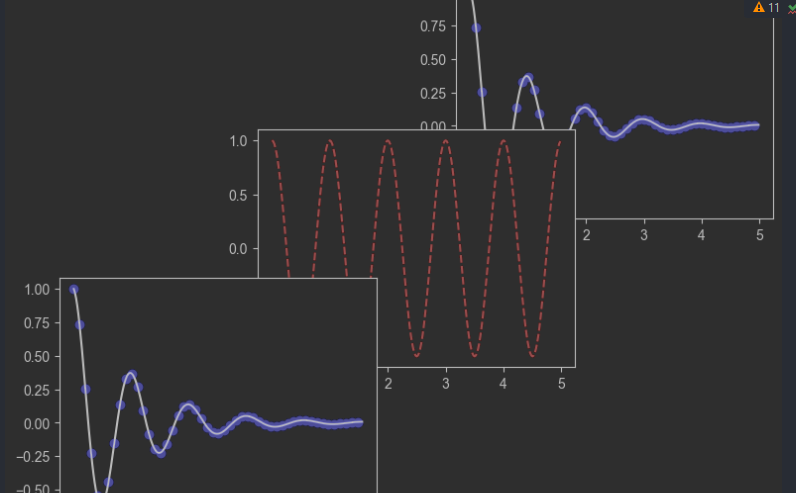
прямоугольнойсетке,акак- нибудьёлочкой,используйтекоманду***axes()***,котораяпозволяетопределить

положениеосейкак

***axes***([left,bottom,width,height]),

гдевсезначенияизменяютсяот0до1.Выглядетьэтоможеттак:

*import* math  
  
*import* numpy *as* np  
  
*import* matplotlib.pyplot *as* plt  
  
*# Импортируем пакет со вспомогательными функциями  
  
import* matplotlib.mlab *as* ml  
  
*import* numpy *as* np  
  
*import* matplotlib.pyplot *as* plt  
  
*def* f(t):  
  
 *return* np.exp(-t) \* np.cos(2\*np.pi\*t)  
  
  
  
t1 = np.arange(0.0, 5.0, 0.1)  
  
t2 = np.arange(0.0, 5.0, 0.02)  
  
plt.figure(1)  
  
plt.axes([0.55, 0.55, 0.4, 0.4])  
  
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')  
  
plt.axes([0.3, 0.3, 0.4, 0.4])  
  
plt.plot(t2, np.cos(2\*np.pi\*t2), 'r--')  
  
plt.axes([0.05, 0.05, 0.4, 0.4])  
  
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')  
  
*# Покажем окно с нарисованным графиком*plt.show()



Имеетсявозможностьсоздатьнесколькоокон,вызывая**figure()**с

увеличивающимсяномеромокна.Конечно,каждоеокноможетсодержать

несколькоосейиподокон.

Очиститьактивноеокноможноприпомощи***clf()***,аактивныеосиприпомощи

***cla()***.

*import* math  
  
*import* numpy *as* np  
  
*import* matplotlib.pyplot *as* plt  
  
*# Импортируем пакет со вспомогательными функциями  
  
import* matplotlib.mlab *as* ml  
  
*import* numpy *as* np  
  
*import* matplotlib.pyplot *as* plt  
  
  
  
plt.figure(1) *# the first figure*plt.subplot(211) *# the first subplot in the first figure*plt.plot([1,2,3])  
  
plt.subplot(212) *# the second subplot in the first figure*plt.plot([4,5,6],[-16,-25,-36])  
  
plt.figure(2) *# a second figure*plt.plot([4,5,6],[16,25,36]) *# creates a subplot(111)  
 #by default*plt.figure(3)  
  
plt.plot([4,5,6],[-16,-25,-36])  
  
plt.show()

**! Протестировать самостоятельно.**

## Установить свой диапазон осей

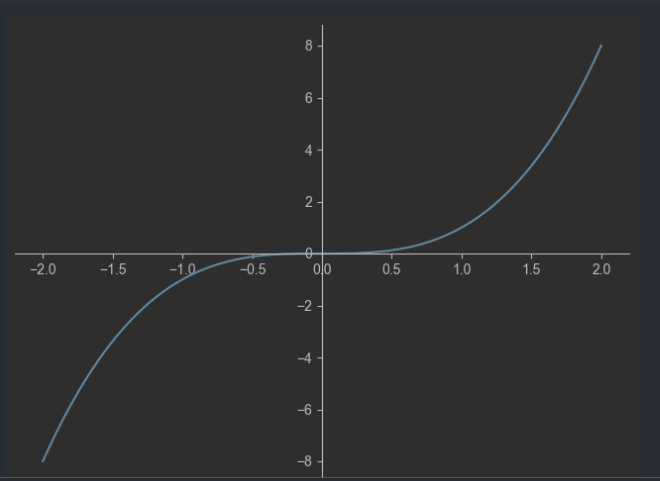
plt.xlim(right=xmax) *#xmax is your value*plt.xlim(left=xmin) *#xmin is your value*plt.ylim(top=ymax) *#ymax is your value*plt.ylim(bottom=ymin) *#ymin is your value*Или так:  
  
plt.ylim(ymin, ymax)  
  
plt.xlim(xmin, xmax)

## Перенос координатных осей в центр графика

Если необходимо перенести координатные оси в «центр рисунка», можно поступить так:

*import* matplotlib.pyplot *as* plt  
  
*import* numpy *as* np  
  
X = np.linspace(-2,2,100)  
  
Y = X\*\*3  
  
plt.plot(X, Y)  
  
ax = plt.gca()  
  
ax.spines['left'].set\_position('center')  
  
ax.spines['bottom'].set\_position('center')  
  
ax.spines['top'].set\_visible(*False*)  
  
ax.spines['right'].set\_visible(*False*)  
  
plt.show()

**Результат**:



Пример переноса осей для **subplot:**

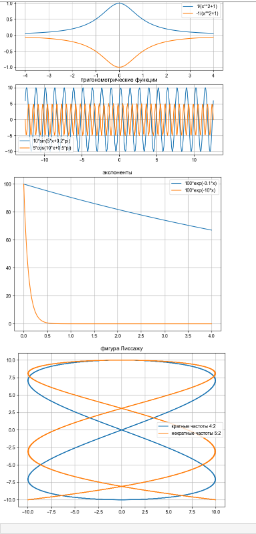
*#Пример получения осей для subplots:  
  
# Two subplots, unpack the axes array immediately*f, (ax1, ax2) = plt.subplots(1, 2, sharey=*True*)  
  
ax1.plot(X, Y)  
  
ax1.set\_title('Sharing Y axis')  
  
ax1.spines['left'].set\_position('center')  
  
ax1.spines['bottom'].set\_position('center')  
  
ax1.spines['top'].set\_visible(*False*)  
  
ax1.spines['right'].set\_visible(*False*)  
  
  
  
ax2.scatter(X, Y)  
  
ax2.spines['left'].set\_position('center')  
  
ax2.spines['bottom'].set\_position('center')  
  
ax2.spines['top'].set\_visible(*False*)  
  
ax2.spines['right'].set\_visible(*False*)  
  
  
  
f, axarr = plt.subplots(2, 2) *# Four axes,  
 # returned as a 2-d array*axarr[0, 0].plot(X, Y)  
  
axarr[0, 0].spines['left'].set\_position('center')  
  
axarr[0, 0].spines['bottom'].set\_position('center')  
  
axarr[0, 0].spines['top'].set\_visible(*False*)  
  
axarr[0, 0].spines['right'].set\_visible(*False*)  
  
axarr[0, 0].set\_title('Axis [0,0]')  
  
  
  
axarr[0, 1].scatter(X, Y)  
  
axarr[0, 1].set\_title('Axis [0,1]')  
  
axarr[1, 0].plot(X, X \*\* 2)  
  
axarr[1, 0].set\_title('Axis [1,0]')  
  
axarr[1, 1].scatter(X, X \*\* 1)  
  
axarr[1, 1].set\_title('Axis [1,1]')  
  
  
  
plt.show()

**! Результат получить самостоятельно.**

## Надписи на окнах диаграмм и окнах Windows, изменение размера окна

*import* math  
  
*import* numpy *as* np  
  
*import* matplotlib.pyplot *as* plt  
  
*# Импортируем пакет со вспомогательными функциями  
  
import* matplotlib *as* mpl  
  
*import* numpy *as* np  
  
*import* matplotlib.pyplot *as* plt  
  
  
x1=np.linspace(-4,4,100)  
  
y1=1/(x1\*\*2+1)  
  
y11=-1/(x1\*\*2+1)  
  
*# Зададим размеры изображения диаграммы*mpl.rcParams['figure.figsize'] = (8.0, 6.0)  
  
plt.figure("Заголовок 1-го окна Windows") *# первое  
 # окно windows*plt.subplot(211) *# первые графики в этом окне*plt.plot(x1,y1,label='1/(x\*\*2+1)')  
  
plt.plot(x1,y11,label='-1/(x\*\*2+1)')  
  
plt.title("графики 1/(x\*\*2+1) и x\*\*2+1" )  
  
plt.grid(*True*)  
  
plt.legend()  
  
plt.subplot(212) *# вторые графики в этом окне*x2=np.linspace(-4\*np.pi,4\*np.pi,400)  
  
plt.plot(x2,10\*np.sin(5\*x2+0.2\*np.pi),label='10\*sin(5\*x+0.2\*pi)')  
  
plt.plot(x2,5\*np.cos(10\*x2+0.5\*np.pi),label="5\*cos(10\*x+0.5\*pi)")  
  
plt.title("тригонометрические функции" )  
  
plt.grid(*True*)  
  
plt.legend(loc='best')  
  
  
  
plt.figure("Заголовок 2-го окна Windows") *# a second  
 #figure  
  
# или так  
  
#fig = plt.figure()  
  
#fig.canvas.set\_window\_title("Заголовок 2-го окна Windows")*x3=np.linspace(0,4,400)  
  
plt.plot(x3, 100\*np.exp(-0.1\*x3), label='100\*exp(-0.1\*x)')  
  
plt.plot(x3, 100\*np.exp(-10\*x3), label='100\*exp(-10\*x)')  
  
plt.title("экспоненты" )  
  
plt.grid(*True*)  
  
plt.legend(loc='best')  
  
fig3=plt.figure()  
  
*#fig3.canvas.set\_window\_title("Заголовок 3-го окна Windows")*plt.grid(*True*)  
  
t=np.linspace(0,8\*np.pi,500)  
  
plt.plot(10\*np.sin(4\*t),10\*np.cos(2\*t),  
 label="кратные частоты 4:2")  
  
plt.plot(10\*np.sin(5\*t),10\*np.cos(2\*t),  
 label="некратные частоты 5:2")  
  
plt.title("фигура Лиссажу" )  
  
plt.grid(*True*)  
  
plt.grid(*True*)  
  
plt.legend(loc='best')  
  
  
  
plt.show()

**Результат**:



## Параметрический график

Массив*x*необязанбытьмонотонновозрастающим.Можностроитьлюбую

параметрическуюлиниюx=x(t),y=y(t).

Разрешающиеспособностимониторапоразнымосяммогутбытьразными.Поэтому

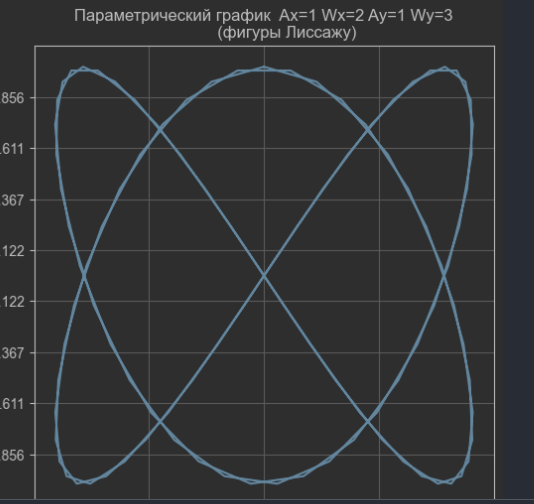
чтобыокружностивыгляделикакокружности,анекакэллипсы,(аквадратыкак

квадраты,анекакпрямоугольники),нужноустановитьпараметр-aspectratio(обычноравный1).

Построимx=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()

**Результат:**

****

## 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.

*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")  
  
plt.grid(*True*)  
  
plt.legend(loc='lower left')  
  
*# визуализируем графики*plt.show()



## 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:

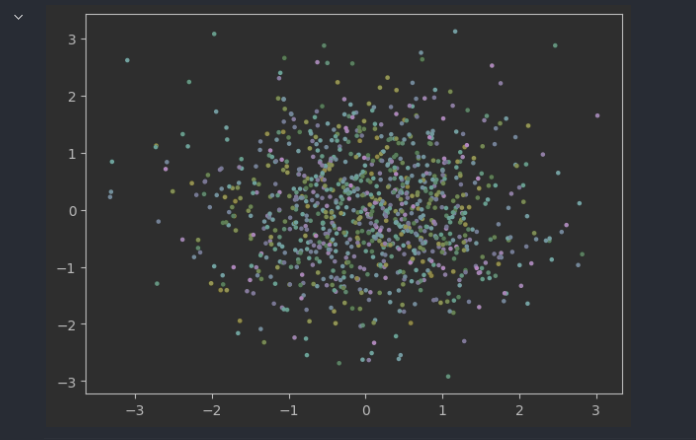
·        **s** задает размер маркеров и, может быть, как одним числом для всех, так и представлять массив значений

·        **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()

**Результат**:



## Настройки в стиле LATEX

Вотпримернастройкипочтивсего,чтоможнонастроить.

Можнозадатьпоследовательностьзасечекнаоси*x*(и*y*)иподписикним(вних,какивдругихтекстах,можноиспользовать*LATEX*-овскиеобозначения).

Задатьподписиосей*x*и*y*изаголовокграфика.

Вовсехтекстовыхэлементахможнозадатьразмершрифта.Можнозадать

толщинулинийиштрихи(так,награфикекосинусарисуетсяштрихдлины8,

потомучастокдлины4нерисуется,потомучастокдлины2рисуется,потом

участокдлины4опятьнерисуется,итакпоциклу;посколькутолщиналинии

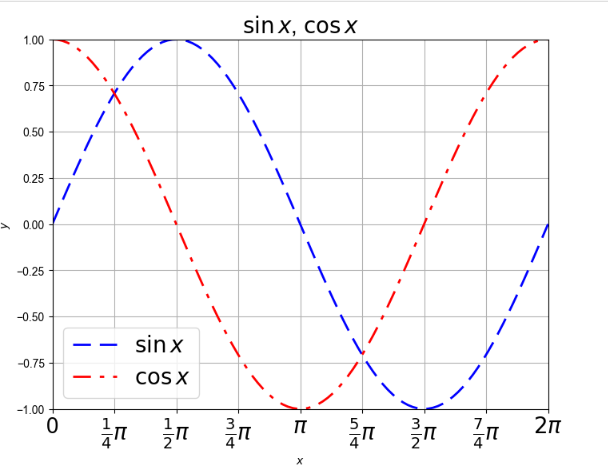
равна2,этикороткиештрихидлины2фактическивыглядяткакточки).

Можнозадатьподписиккривым(legend);гдеразместитьэтиподписитоже

можнорегулировать.

*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()

**Результат**:

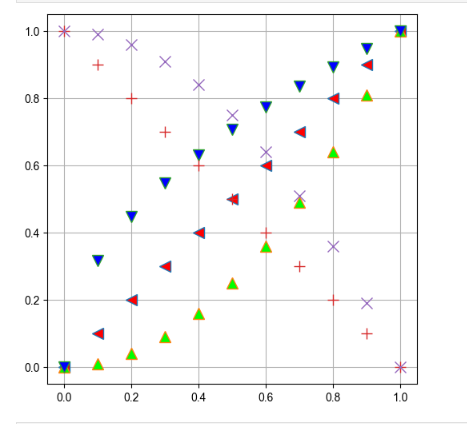


## Модифицированные маркеры

Еслиlinestyle='',тоточкинесоединяютсялиниями.Самиточкирисуютсямаркерамиразныхтипов.Типопределяетсястрокойизодногосимвола,которыйчем-топохожнанужныймаркер.Вдобавоккстандартныммаркерам,можноопределитьсамодельные.

*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()

Результат:



## Экспериментальные данные

Допустим,имеетсятеоретическаякривая(резонансбезфона).

xt=np.linspace(-4,4,101)

yt=1/(xt\*\*2+1)

Посколькуреальныхэкспериментальныхданныхподрукойнет,мыих

сгенерируем.Пустьонисогласуютсястеорией,ивсестатистическиеошибки

равны0.1.

xe=np.linspace(-3,3,21)

yerr=0.1\*np.ones(21)

ye=1/(xe\*\*2+1)+yerr\*np.random.normal(size=21)

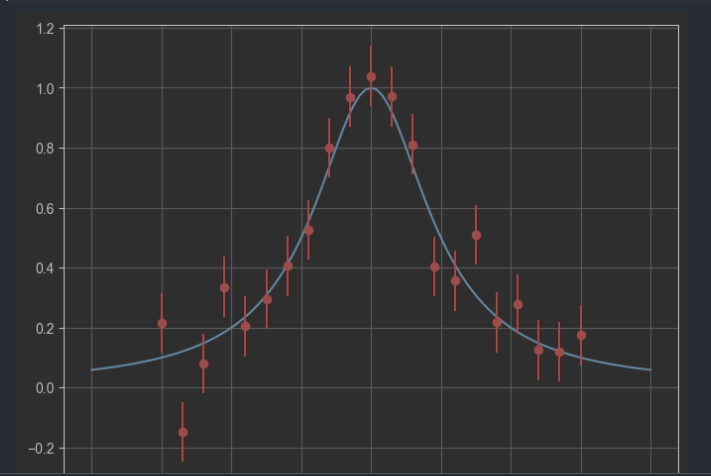
***Экспериментальные точки с усами***итеоретическаякриваянаодномграфике.

plt.plot(xt,yt)

**plt.errorbar(xe,ye,fmt='ro',yerr=yerr)**

*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()

**Результат**:



## Гистограмма

Сгенерируем*N*случайныхчиселснормальным(гауссовым)распределением(

среднее0,среднеквадратичноеотклонение1),ираскидаемихпо20битамот−3до3(точкизапределамиэтогоинтервалаотбрасываются).Длясравнения,вместес

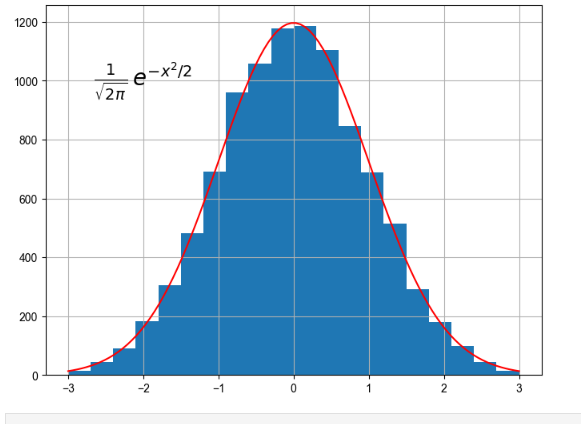
гистограммойнарисуемГауссовукривуювтомжемасштабе.Идаженапишем

формулуГаусса.

**Пример 1**

*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()

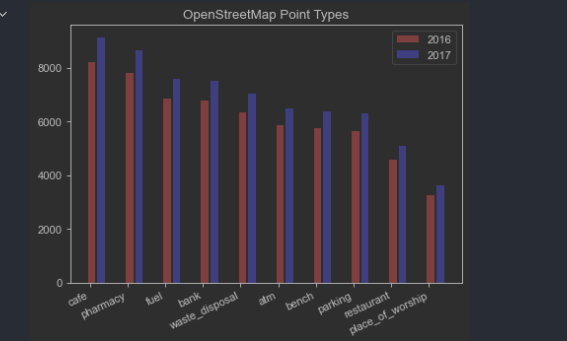
Результат:



**Пример 2 Столбиковая диаграмма**

*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()

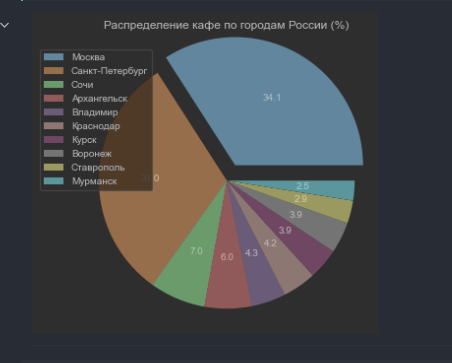
**Результат**:



## Круговая диаграмма

*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()

**Результат**:

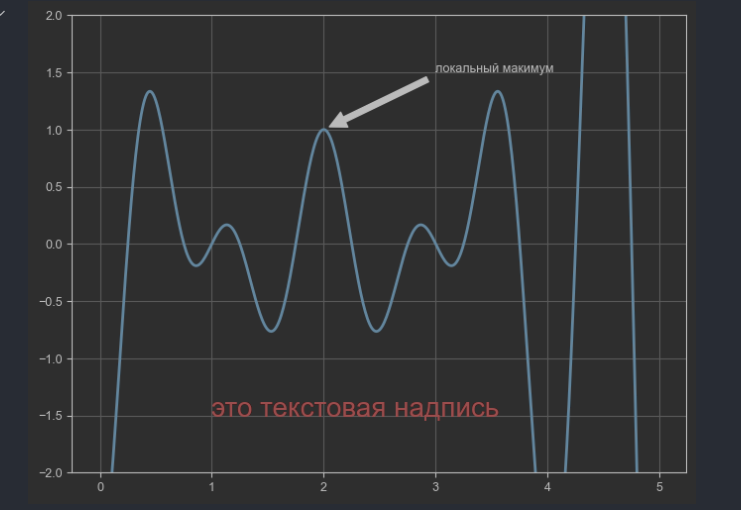


## Текст и надписи

Текстидополнительныенадписи(аннотации)размещаютсянадиаграмме:

*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()

**Результат**:



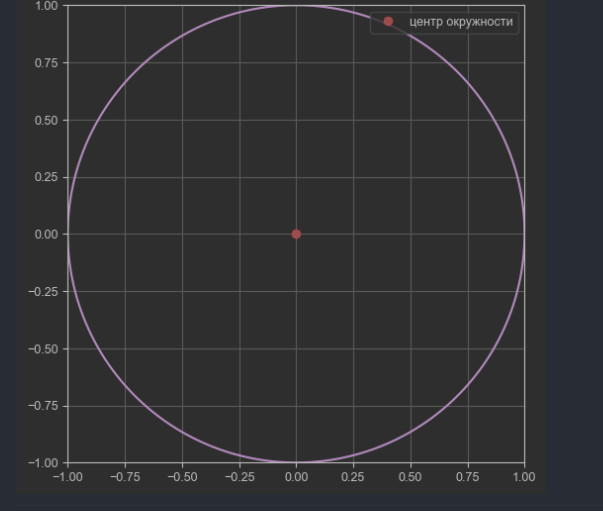
## Контурные графики

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

**Пример 1**

*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()

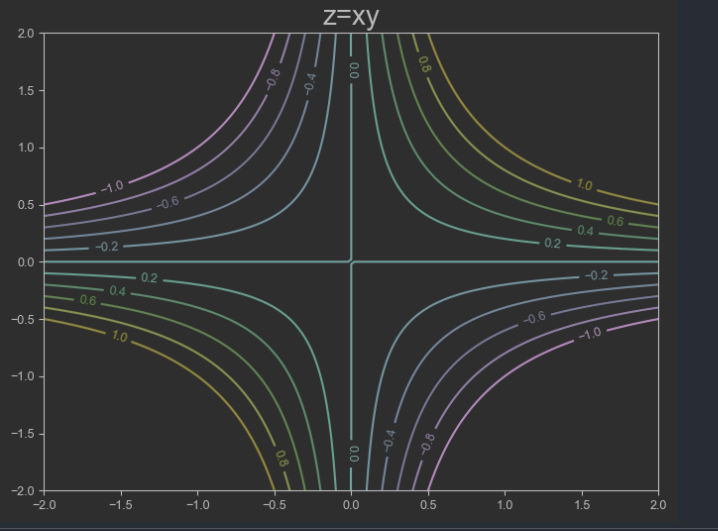
**Результат**:



**Пример 2**

*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()

**Результат**:



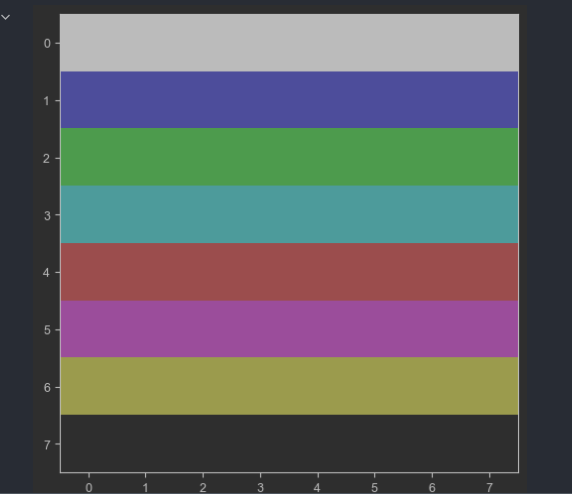
## Images (пиксельные картинки)

Картинказадаётсямассивомz:z[i,j]-этоцветпикселяi,j,массивиз3элементов

(rgb,числаот0до1).Длянаглядностивпримереформируются«натуральные»цветавсписокcol:

*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()

**Результат**:



## Построение 3D графиков. Работа с mplot3d Toolkit

До этого момента все графики, которые мы строили были двумерные, Matplotlib позволяет строить 3D графики. Импортируем необходимые модули для работы с 3D: **import matplotlib.pyplot as plt**

**from mpl\_toolkits.mplot3d import Axes3D**

Рассмотрим некоторые из инструментов для построения 3D графиков.

### Линейный график

Для построения линейного графика используется функция plot():

Axes3D.plot(self, xs, ys, \*args, zdir='z', \*\*kwargs)

**Параметры функции Axes3D.plot:**

• xs: 1D массив

◦ x координаты.

• ys: 1D массив

◦ y координаты.

• zs: скалярное значение или 1D массив

◦ z координаты. Если передан скаляр, то он будет присвоен всем точкам графика.

• zdir: {'x', 'y', 'z'}; значение по умолчанию: 'z'

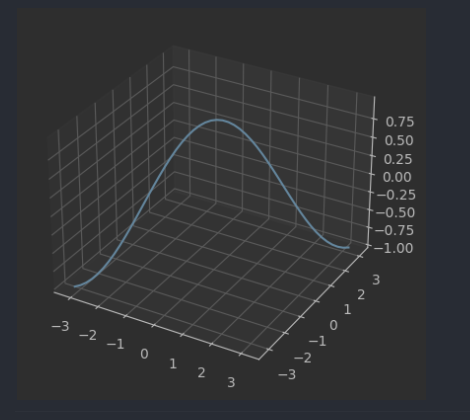
◦ Определяет ось, которая будет принята за z направление.

• \*\*kwargs

◦ Дополнительные аргументы, аналогичные тем, что используются в функции plot() для построения двумерных графиков.

*import* numpy *as* np  
*import* matplotlib.pyplot *as* plt  
x = np.linspace(-np.pi, np.pi, 50)  
y = x  
z = np.cos(x)  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.plot(x, y, z, label='parametric curve')  
plt.show()

**Результат:**



### Каркасная поверхность

Для построения каркасной поверхности используется функция plot\_wireframe(): plot\_wireframe(self, X, Y, Z, \*args, \*\*kwargs)

**Параметры функции Axes3D.wireframe():**

• X, Y, Z: 2D массивы

◦ Данные для построения поверхности.

• rcount, ccount: int, значение по умолчанию: 50

◦ Максимальное количество элементов каркаса, которое будет использовано в каждом из направлении.

• rstride, cstride: int

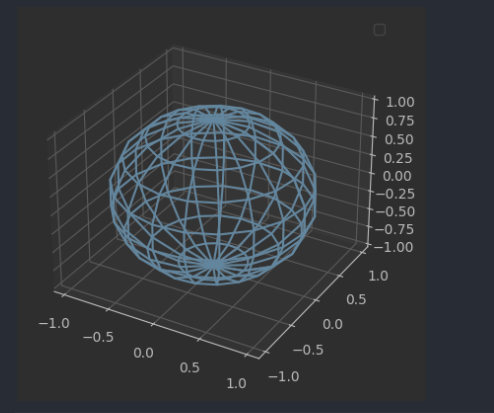
◦ Параметры, определяющие величину шага, с которым будут браться элементы строки / столбца из переданных массивов. Параметры rstride, cstride и rcount, ccount являются взаимоисключающими.

• \*\*kwargs

◦ Дополнительные аргументы, определяемые Line3DCollection.

u, v = np.mgrid[0:2\*np.pi:20j, 0:np.pi:10j]  
x = np.cos(u)\*np.sin(v)  
y = np.sin(u)\*np.sin(v)  
z = np.cos(v)  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.plot\_wireframe(x, y, z)  
ax.legend()  
plt.show()

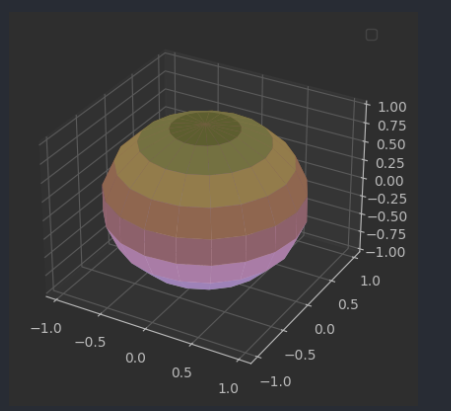
**Результат:**



### Поверхность, залитая цветом

u, v = np.mgrid[0:2\*np.pi:20j, 0:np.pi:10j]  
x = np.cos(u)\*np.sin(v)  
y = np.sin(u)\*np.sin(v)  
z = np.cos(v)  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.plot\_surface(x, y, z, cmap='inferno')  
ax.legend()  
plt.show()

**Результат:**



# Задание

1. Изучить теоретический материал и примеры, приведенные в описании
2. Построить график произвольной **квадратичной функции**, масштаб задать произвольно. Сделать настройки подписей осей, заголовка фигуры
3. Построить график произвольной **гармонической функции**, масштаб задать произвольно. Сделать настройки подписей осей, заголовка фигуры
4. Вывести графики п.2 и 3 в разные поля при помощи функции **subplot()**
5. Построить графики п.2 и 3 в одной координатной плоскости, сделать подписи графиков, указать легенду.
6. Построить график функции, заданной параметрически,
7. Построить столбиковую диаграмму по данным статистического комитета КР по прожиточному минимуму, представленному в таблице

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Наименование показателей** | **2012** | **2013** | **2014** | **2015** | **2016** | **2017** | **2018** | **2019** | **2020** | **2021** | **2022** |
| **Кыргызская Республика** | **4341.15** | **4599.21** | **4981.51** | **5182.99** | **4794.34** | **4900.79** | **4792.54** | **4806.32** | **5358.53** | **6268.31** | **7178.32** |

1. Построить круговой график по данным п. 7
2. Построить график произвольной каркасной поверхности.

### Источник:

С.Ю. Борю. Модули и пакеты в Python. Учебно-методическое пособие для студентов естественнонаучных специальностей. Запорожье, 2021

<http://kit.znu.edu.ua/kit/iLec/5sem/pack_python/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D0%B8_%D0%B8_%D0%BF%D0%B0%D0%BA%D0%B5%D1%82%D1%8B_%D0%B2_Python_21.htm>