Бесплатная электронная книга

УЧУСЬ

matplotlib

Free unaffiliated eBook created from
Stack Overflow contributors.

#matplotlib

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: matplotlib

It is an unofficial and free matplotlib ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official matplotlib.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с matplotlib

замечания

обзор

matplotlib - это графическая библиотека для Python. Он предоставляет объектно-ориентированные API для встраивания графиков в приложения. Он похож на MATLAB в качестве и синтаксисе.

Он был первоначально написан JDHunter и активно развивается. Он распространяется под лицензией BSD-Style.

Версии

Версия	Поддерживаемые версии Python	замечания	Дата выхода
1.3.1	2,6, 2,7, 3,8	Старая стабильная версия	2013-10-10
1.4.3	2,6, 2,7, 3,8	Предыдущая стабильная версия	2015-07-14
1.5.3	2.7, 3.x	Текущая стабильная версия	2016-01-11
2.x	2.7, 3.x	Последняя версия разработки	2016-07-25

Examples

Установка и настройка

Существует несколько способов установки matplotlib, некоторые из которых будут зависеть от используемой вами системы. Если вам повезет, вы сможете использовать диспетчер пакетов, чтобы легко установить модуль matplotlib и его зависимости.

Windows

На компьютерах Windows вы можете попытаться использовать диспетчер пакетов pip для

установки matplotlib. См. Здесь информацию о настройке пипса в среде Windows.

OS X

Рекомендуется использовать пип менеджер пакетов для установки Matplotlib. Если вам нужно установить некоторые из не-Python-библиотек в вашей системе (например, `libfreetype`), рассмотрите возможность использования доморощенного .

Если вы не можете использовать pip по какой-либо причине, попробуйте установить его из источника .

Linux

В идеале, системный диспетчер пакетов или пип должен использоваться для установки matplotlib, либо путем установки пакета `python-matplotlib` , либо путем запуска `pip install matplotlib` .

Если это невозможно (например, у вас нет привилегий sudo на компьютере, который вы используете), вы можете установить его из источника с `--user` опции `--user` : `python setup.py install --user` . Как правило, это устанавливает matplotlib в `~/.local` .

Debian / Ubuntu

```
sudo apt-get install python-matplotlib
```

Fedora / Red Hat

```
sudo yum install python-matplotlib
```

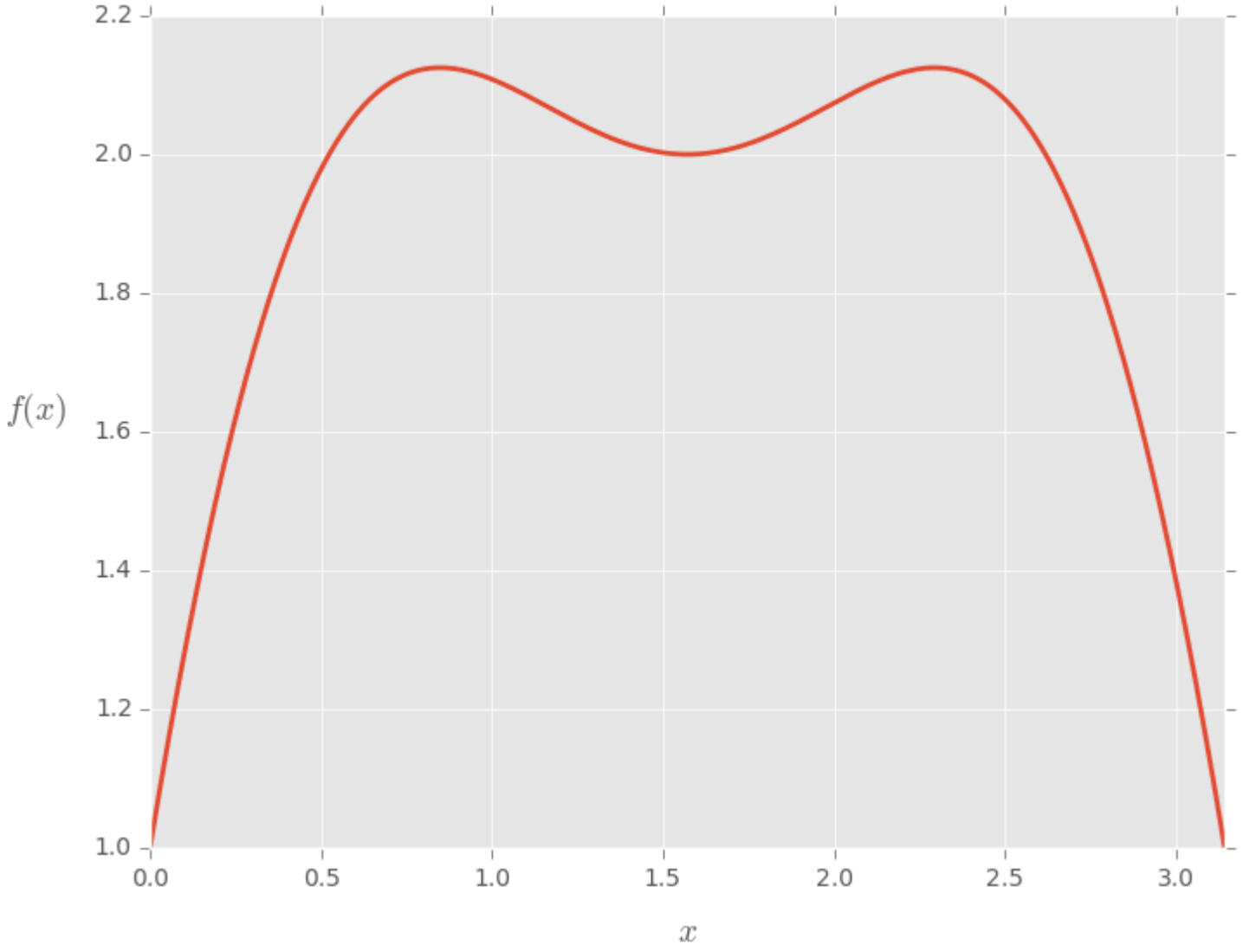
Поиск проблемы

См. Веб-сайт matplotlib, чтобы узнать, как исправить сломанный matplotlib.

Настройка графика matplotlib

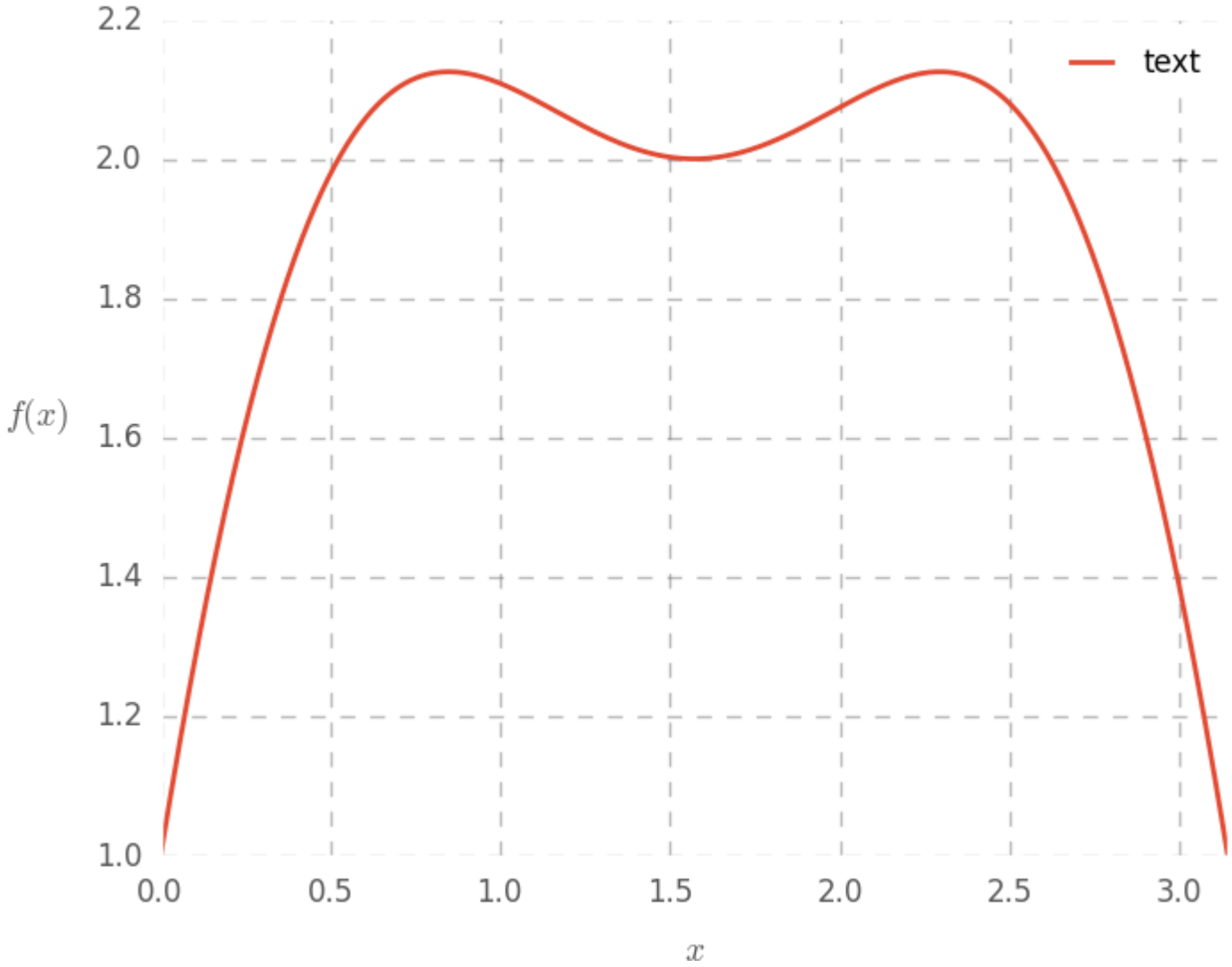
```
import pylab as plt
import numpy as np

plt.style.use('ggplot')

fig = plt.figure(1)
ax = plt.gca()

# make some testing data
```

```
x = np.linspace( 0, np.pi, 1000 )
test_f = lambda x: np.sin(x)*3 + np.cos(2*x)

# plot the test data
ax.plot( x, test_f(x) , lw = 2)

# set the axis labels
ax.set_xlabel(r'$x$', fontsize=14, labelpad=10)
ax.set_ylabel(r'$f(x)$', fontsize=14, labelpad=25, rotation=0)

# set axis limits
ax.set_xlim(0,np.pi)

plt.draw()
```




```
# Customize the plot
ax.grid(1, ls='--', color='#777777', alpha=0.5, lw=1)
ax.tick_params(labelsize=12, length=0)
ax.set_axis_bgcolor('w')
# add a legend
leg = plt.legend( ['text'], loc=1 )
fr = leg.get_frame()
```

```
fr.set_facecolor('w')
fr.set_alpha(.7)
plt.draw()
```



Императивный и объектно-ориентированный синтаксис

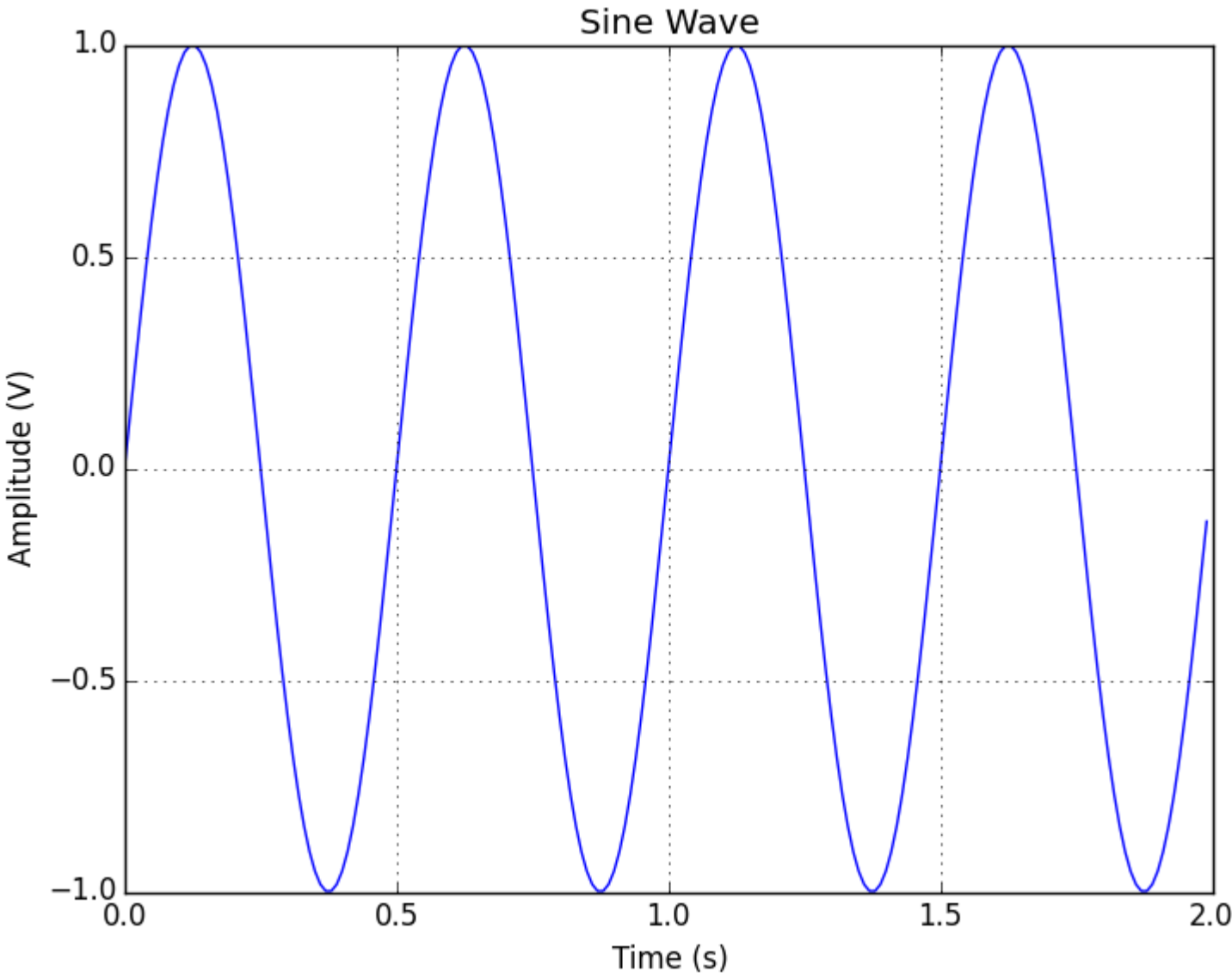
Matplotlib поддерживает как объектно-ориентированный, так и императивный синтаксис для построения графика. Настоящий синтаксис преднамеренно разработан, чтобы быть очень близким к синтаксису Matlab.

Обязательный синтаксис (иногда называемый синтаксисом state-machine) выдает строку команд, все из которых действуют на самую последнюю фигуру или ось (например, Matlab). С другой стороны, объектно-ориентированный синтаксис явно действует на объекты (фигура, ось и т. Д.), Представляющие интерес. Ключевой момент в zen Python гласит, что явный лучше, чем неявный, поэтому объектно-ориентированный синтаксис более питоничен. Тем не менее, императивный синтаксис удобен для новых конвертиров из Matlab и для написания небольших сценариев «отбрасывания». Ниже приведен пример

двух разных стилей.

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

t = np.arange(0, 2, 0.01)
y = np.sin(4 * np.pi * t)

# Imperative syntax
plt.figure(1)
plt.clf()
plt.plot(t, y)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (V)')
plt.title('Sine Wave')
plt.grid(True)

# Object oriented syntax
fig = plt.figure(2)
fig.clf()
ax = fig.add_subplot(1,1,1)
ax.plot(t, y)
ax.set_xlabel('Time (s)')
ax.set_ylabel('Amplitude (V)')
ax.set_title('Sine Wave')
ax.grid(True)
```

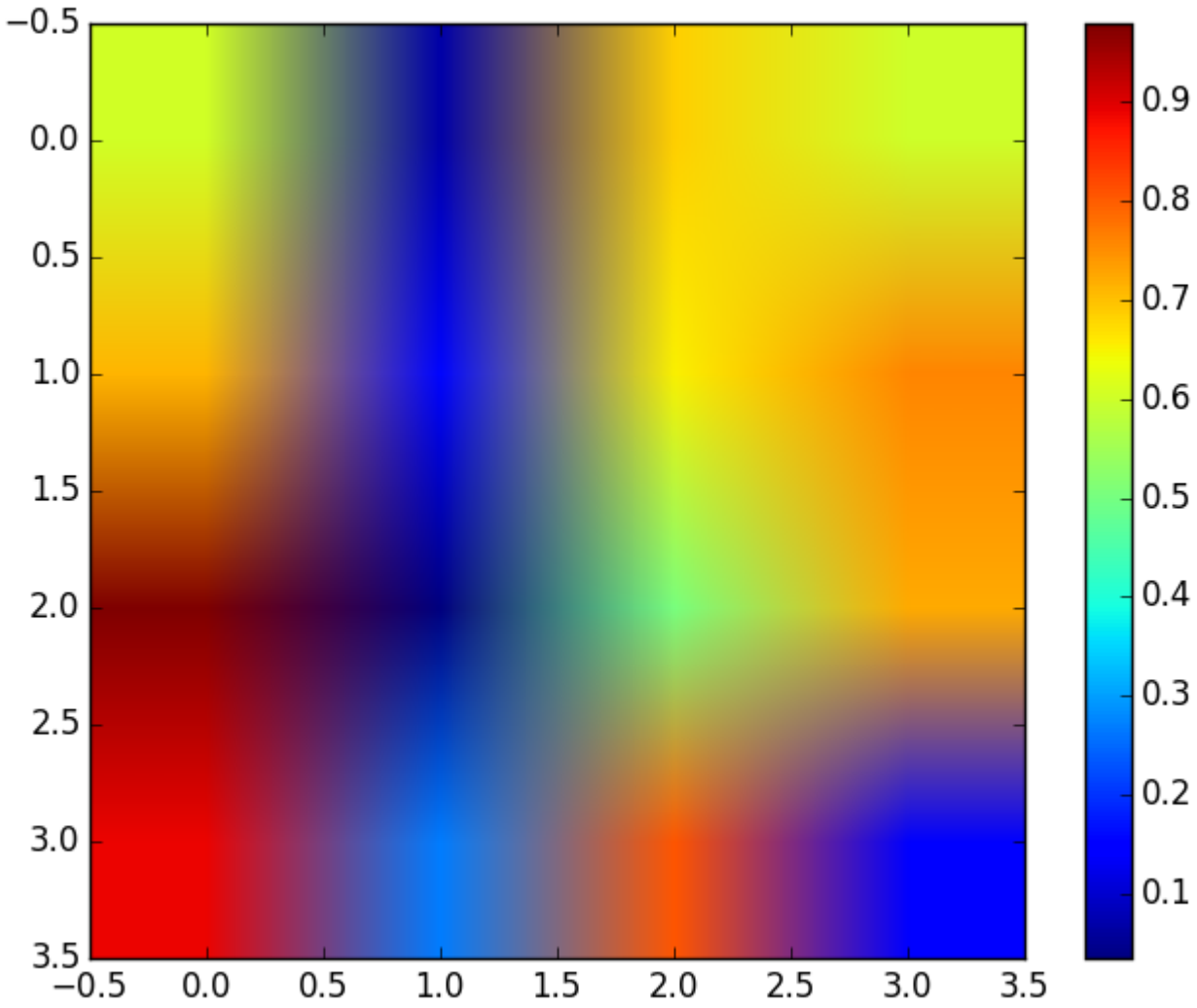
Оба примера дают тот же график, который показан ниже.



Двумерные (2D) массивы

Отображение двухмерного (2D) массива на осях.

```
import numpy as np
from matplotlib.pyplot import imshow, show, colorbar

image = np.random.rand(4,4)
imshow(image)
colorbar()
show()
```



Прочитайте Начало работы с matplotlib онлайн: https://riptutorial.com/ru/matplotlib/topic/881/начало-работы-с-matplotlib

глава 2: Анимация и интерактивный график

Вступление

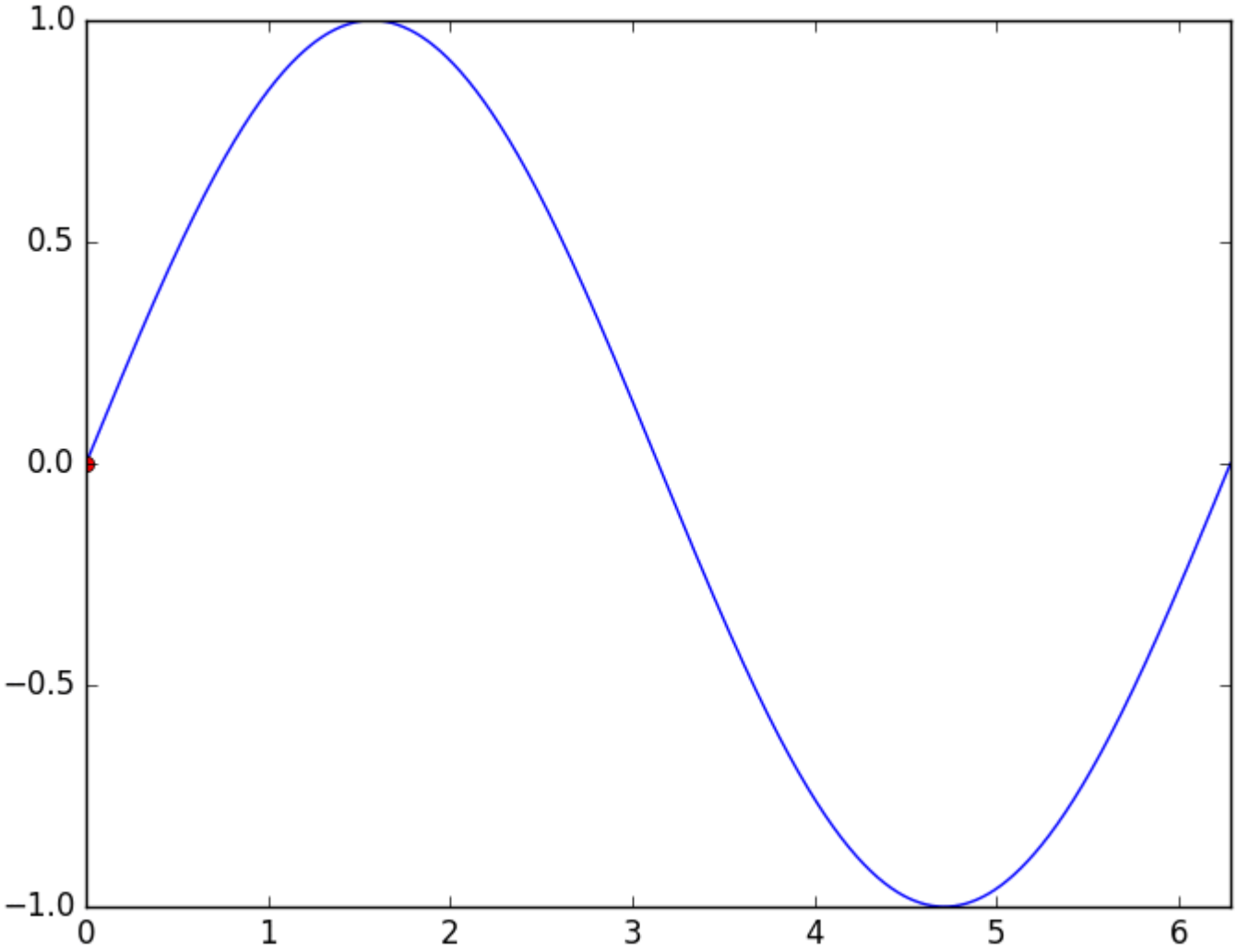
С помощью python matplotlib вы можете правильно создавать анимированные графики.

Examples

Базовая анимация с FuncAnimation

Пакет matplotlib.animation предлагает несколько классов для создания анимаций. `FuncAnimation` создает анимации, повторно вызывая функцию. Здесь мы используем функцию `animate()` которая меняет координаты точки на графике синусоидальной функции.

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

TWOPI = 2*np.pi

fig, ax = plt.subplots()

t = np.arange(0.0, TWOPI, 0.001)
s = np.sin(t)
l = plt.plot(t, s)

ax = plt.axis([0,TWOPI,-1,1])

redDot, = plt.plot([0], [np.sin(0)], 'ro')

def animate(i):
    redDot.set_data(i, np.sin(i))
    return redDot,

# create animation using the animate() function
myAnimation = animation.FuncAnimation(fig, animate, frames=np.arange(0.0, TWOPI, 0.1), \
                                      interval=10, blit=True, repeat=True)

plt.show()
```



Сохранить анимацию в gif

В этом примере мы используем `save` метод для сохранения `Animation` объекта с помощью ImageMagick.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib import rcParams

# make sure the full paths for ImageMagick and ffmpeg are configured
rcParams['animation.convert_path'] = r'C:\Program Files\ImageMagick\convert'
rcParams['animation.ffmpeg_path'] = r'C:\Program Files\ffmpeg\bin\ffmpeg.exe'

TWOPI = 2*np.pi

fig, ax = plt.subplots()

t = np.arange(0.0, TWOPI, 0.001)
s = np.sin(t)
l = plt.plot(t, s)
```

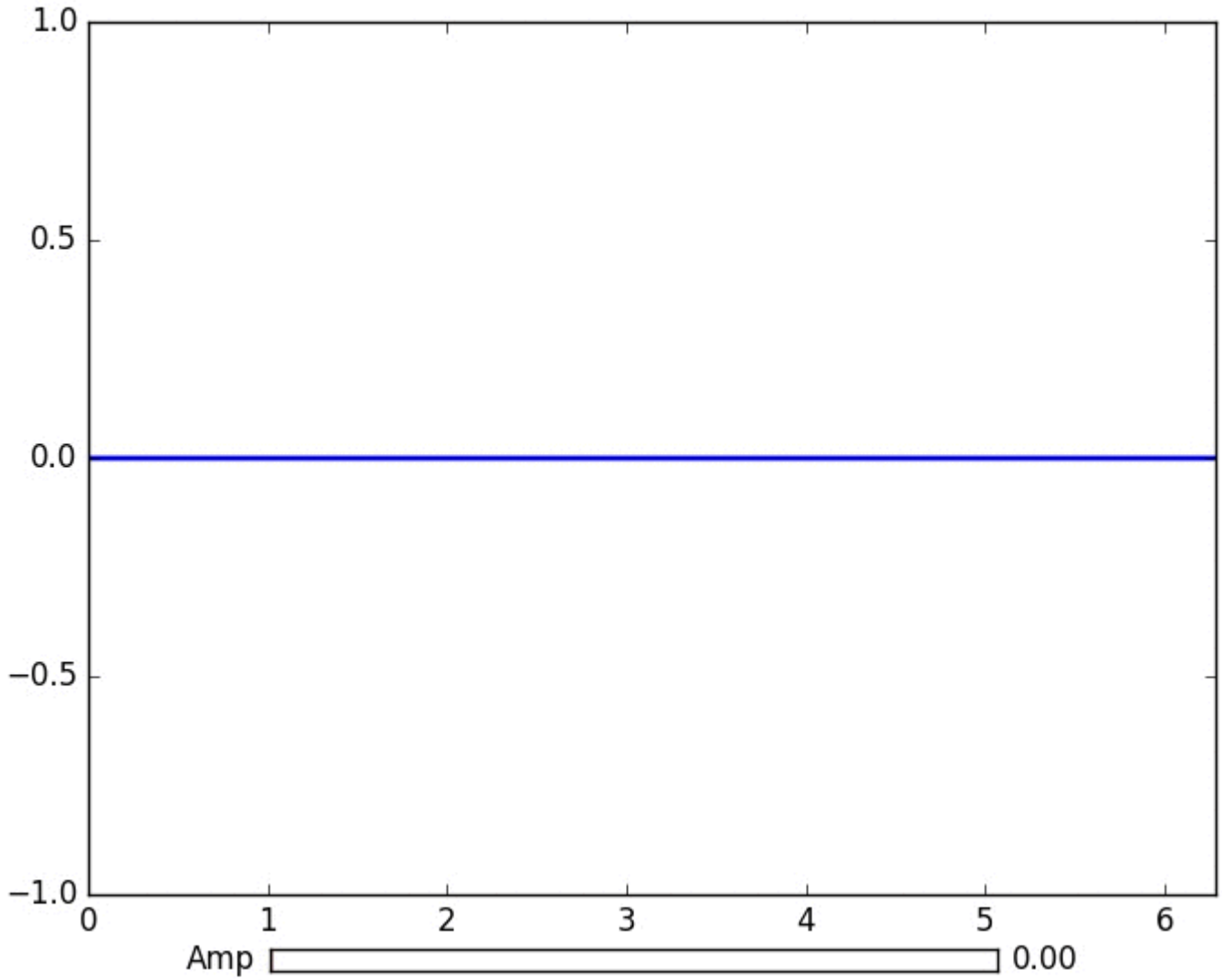
```
ax = plt.axis([0,TWOPI,-1,1])

redDot, = plt.plot([0], [np.sin(0)], 'ro')

def animate(i):
    redDot.set_data(i, np.sin(i))
    return redDot,

# create animation using the animate() function with no repeat
myAnimation = animation.FuncAnimation(fig, animate, frames=np.arange(0.0, TWOPI, 0.1), \
                                      interval=10, blit=True, repeat=False)

# save animation at 30 frames per second
myAnimation.save('myAnimation.gif', writer='imagemagick', fps=30)
```

Интерактивные элементы управления с помощью matplotlib.widgets

Для взаимодействия с сюжетами Matplotlib предлагает GUI нейтральных виджеты . Виджеты требуют объекта `matplotlib.axes.Axes` .

Вот демоверсия слайдера, которая ù обновляет амплитуду кривой синуса. Функция обновления активируется событием `on_changed()` слайдера.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.widgets import Slider

TWOPI = 2*np.pi

fig, ax = plt.subplots()

t = np.arange(0.0, TWOPI, 0.001)
initial_amp = .5
s = initial_amp*np.sin(t)
l, = plt.plot(t, s, lw=2)

ax = plt.axis([0,TWOPI,-1,1])

axamp = plt.axes([0.25, .03, 0.50, 0.02])
# Slider
samp = Slider(axamp, 'Amp', 0, 1, valinit=initial_amp)

def update(val):
    # amp is the current value of the slider
    amp = samp.val
    # update curve
    l.set_ydata(amp*np.sin(t))
    # redraw canvas while idle
    fig.canvas.draw_idle()

# call update function on slider value change
samp.on_changed(update)

plt.show()
```



Другие доступные виджеты:

- AxesWidget
- кнопка
- CheckButtons
- Курсор
- EllipseSelector
- Ласso
- LassoSelector
- LockDraw
- MultiCursor
- Радио-кнопки
- RectangleSelector
- SpanSelector
- SubplotTool
- ToolHandles

Строить живые данные из трубы с помощью matplotlib

Это может быть полезно, когда вы хотите визуализировать входящие данные в режиме реального времени. Эти данные могут, например, поступать от микроконтроллера, который непрерывно сэмплирует аналоговый сигнал.

В этом примере мы получим наши данные из именованного канала (также известного как fifo). В этом примере данные в трубе должны быть числами, разделенными символами новой строки, но вы можете адаптировать их по своему вкусу.

Пример данных:

```
100
123.5
1589
```

Дополнительная информация о названных каналах

Мы также будем использовать datatype deque из стандартных коллекций библиотек. Объект deque очень похож на список. Но с объектом deque довольно легко добавить что-то к нему, сохраняя при этом объект deque фиксированной длины. Это позволяет нам удерживать ось x на фиксированной длине, а не всегда увеличивать и сжимать график вместе. Дополнительная информация о объектах deque

Выбор правильного бэкэнда имеет жизненно важное значение для производительности. Проверьте, какие серверы работают в вашей операционной системе, и выберите быстрый. Для меня работали только qt4agg и бэкэнд по умолчанию, но по умолчанию он был слишком медленным. Дополнительная информация о бэкэндах в matplotlib

Этот пример основан на примере matplotlib для построения случайных данных .

Ни один из символов этого кода не должен удаляться.

```
import matplotlib
import collections
#selecting the right backend, change qt4agg to your desired backend
matplotlib.use('qt4agg')
import matplotlib.pyplot as plt
import matplotlib.animation as animation

#command to open the pipe
datapipe = open('path to your pipe','r')

#amount of data to be displayed at once, this is the size of the x axis
#increasing this amount also makes plotting slightly slower
data_amount = 1000

#set the size of the deque object
datalist = collections.deque([0]*data_amount,data_amount)

#configure the graph itself
```

```
fig, ax = plt.subplots()
line, = ax.plot([0,]*data_amount)

#size of the y axis is set here
ax.set_ylim(0,256)

def update(data):
        line.set_ydata(data)
        return line,

def data_gen():
    while True:
        """
        We read two data points in at once, to improve speed
        You can read more at once to increase speed
        Or you can read just one at a time for improved animation smoothness
        data from the pipe comes in as a string,
        and is seperated with a newline character,
        which is why we use respectively eval and rstrip.
        """
        datalist.append(eval((datapipe.readline()).rstrip('\n')))
        datalist.append(eval((datapipe.readline()).rstrip('\n')))
        yield datalist

ani = animation.FuncAnimation(fig,update,data_gen,interval=0, blit=True)
plt.show()
```

Если ваш участок начинает задерживаться через некоторое время, попробуйте добавить больше данных datalist.append, чтобы больше строк считывалось каждый кадр. Или выберите более быстрый бэкэнд, если сможете.

Это работало с данными 150 Гц с трубки на моем 1,7 Гц i3 4005u.

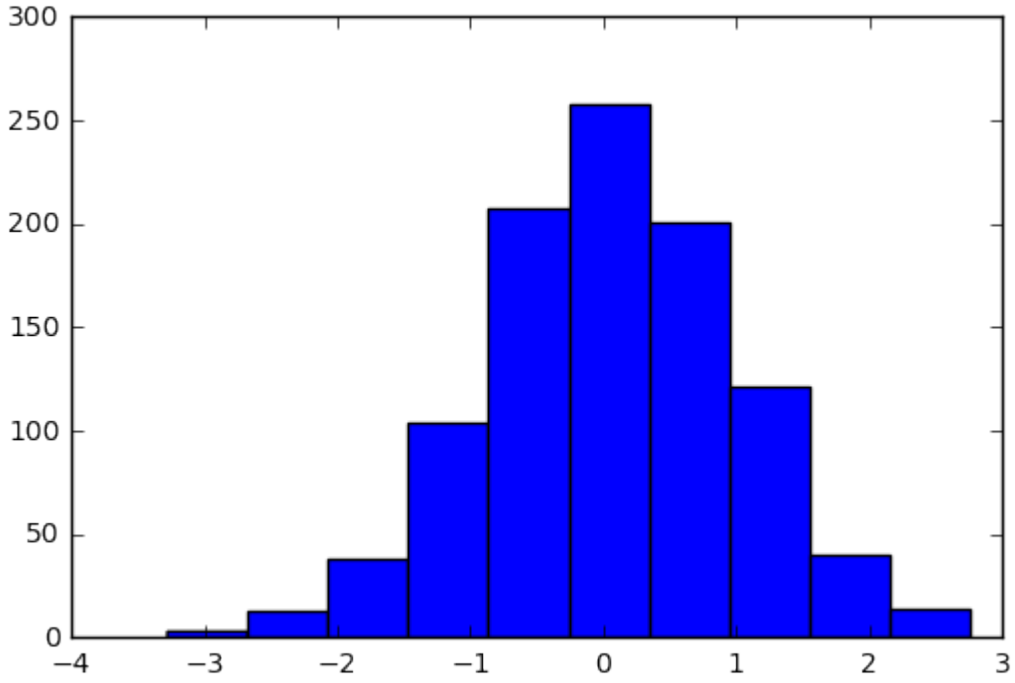
Прочитайте Анимация и интерактивный график онлайн:
https://riptutorial.com/ru/matplotlib/topic/6983/анимация-и-интерактивный-график

глава 3: Гистограмма

Examples

Простая гистограмма

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

# generate 1000 data points with normal distribution
data = np.random.randn(1000)

plt.hist(data)

plt.show()
```



Прочитайте Гистограмма онлайн: https://riptutorial.com/ru/matplotlib/topic/7329/гистограмма

глава 4: Графики LogLog

Вступление

Графическое отображение LogLog - это возможность проиллюстрировать экспоненциальную функцию линейным способом.

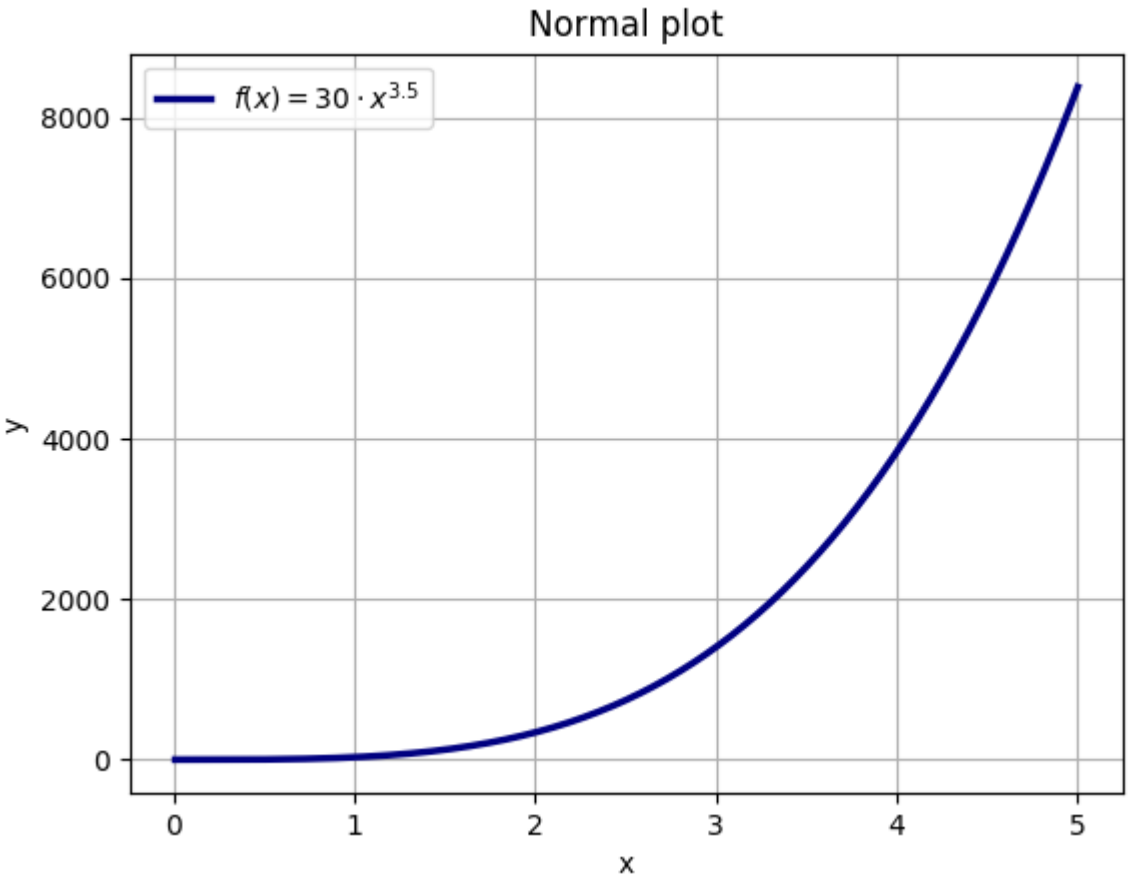
Examples

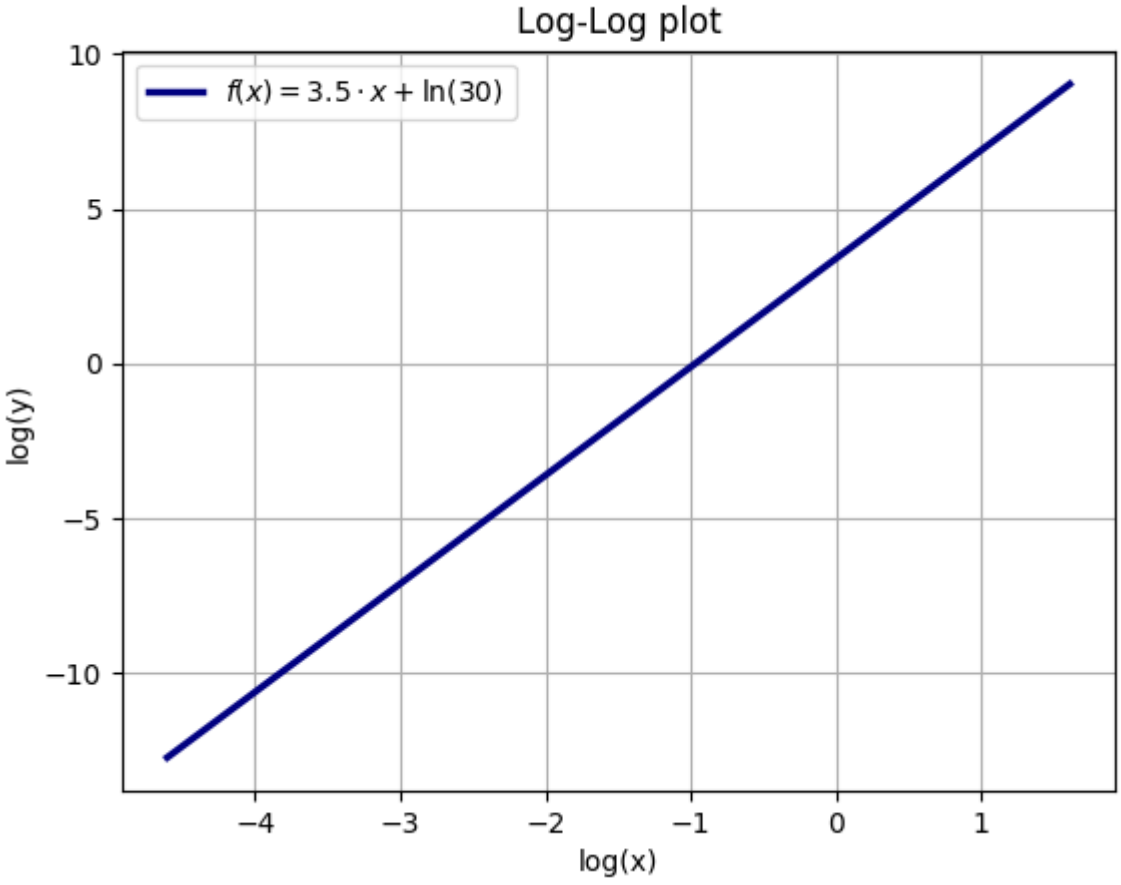
График LogLog

Пусть y (x) = A * x ^ a, например A = 30 и a = 3.5. Взяв натуральный логарифм (ln) обеих сторон, получим (используя общие правила для логарифмов): ln (y) = ln (A * x ^ a) = ln (A) + ln (x ^ a) = ln (A) + a * ln (x). Таким образом, график с логарифмическими осями как для x, так и для y будет линейной кривой. Наклон этой кривой является показателем a of y (x), а y-перехват y (0) является естественным логарифмом A, ln (A) = ln (30) = 3,401.

Следующий пример иллюстрирует связь между экспоненциальной функцией и линейным логарифмическим графиком (функция y = A * x ^ a с A = 30 и a = 3,5):

```
import numpy as np
import matplotlib.pyplot as plt
A = 30
a = 3.5
x = np.linspace(0.01, 5, 10000)
y = A * x**a

ax = plt.gca()
plt.plot(x, y, linewidth=2.5, color='navy', label=r'$f(x) = 30 \cdot x^{3.5}$')
plt.legend(loc='upper left')
plt.xlabel(r'x')
plt.ylabel(r'y')
ax.grid(True)
plt.title(r'Normal plot')
plt.show()
plt.clf()

xlog = np.log(x)
ylog = np.log(y)
ax = plt.gca()
plt.plot(xlog, ylog, linewidth=2.5, color='navy', label=r'$f(x) = 3.5\cdot x + \ln(30)$')
plt.legend(loc='best')
plt.xlabel(r'log(x)')
plt.ylabel(r'log(y)')
ax.grid(True)
plt.title(r'Log-Log plot')
plt.show()
plt.clf()
```


Normal plot
$f(x) = 30 \cdot x^{3.5}$
8000
6000
4000
2000
0
y
0
1
2
3
4
5
x


Log-Log plot
$f(x) = 3.5 \cdot x + \ln(30)$
10
5
0
−5
−10
log(y)
−4
−3
−2
−1
0
1
log(x)

Прочитайте Графики LogLog онлайн: https://riptutorial.com/ru/matplotlib/topic/10145/графики-loglog

глава 5: Закрытие окна фигуры

Синтаксис

- plt.close () # закрывает текущий активный показатель
- plt.close (fig) # закрывает фигуру с помощью ручки 'fig'
- plt.close (num) # закрывает цифру 'num'
- plt.close (name) # закрывает фигуру с меткой 'name'
- plt.close ('all') # закрывает все цифры

Examples

Закрытие текущей активной фигуры с помощью пистолета

Интерфейс pyplot для `matplotlib` может быть самым простым способом закрыть фигуру.

```
import matplotlib.pyplot as plt
plt.plot([0, 1], [0, 1])
plt.close()
```

Закрытие определенной фигуры с использованием plt.close ()

Конкретную цифру можно закрыть, удерживая ее ручку

```
import matplotlib.pyplot as plt

fig1 = plt.figure() # create first figure
plt.plot([0, 1], [0, 1])

fig2 = plt.figure() # create second figure
plt.plot([0, 1], [0, 1])

plt.close(fig1) # close first figure although second one is active
```

Прочитайте Закрытие окна фигуры онлайн: https://riptutorial.com/ru/matplotlib/topic/6628/закрытие-окна-фигуры

глава 6: Интеграция с TeX / LaTeX

замечания

- Для поддержки MatTotlib LaTeX требуется работающая установка LaTeX, dvipng (которая может быть включена в вашу установку LaTeX), и Ghostscript (рекомендуется GPL Ghostscript 8.60 или более поздняя версия).
- Поддержка pgf Matplotlib требует установки LaTeX, которая включает в себя пакеты TikZ / PGF (например, TeXLive), предпочтительно с установленными XeLaTeX или LuaLaTeX.

Examples

Вставка формул TeX в графики

Формулы TeX могут быть вставлены в график с использованием функции `rc`

```
import matplotlib.pyplot as plt
plt.rc(usetex = True)
```

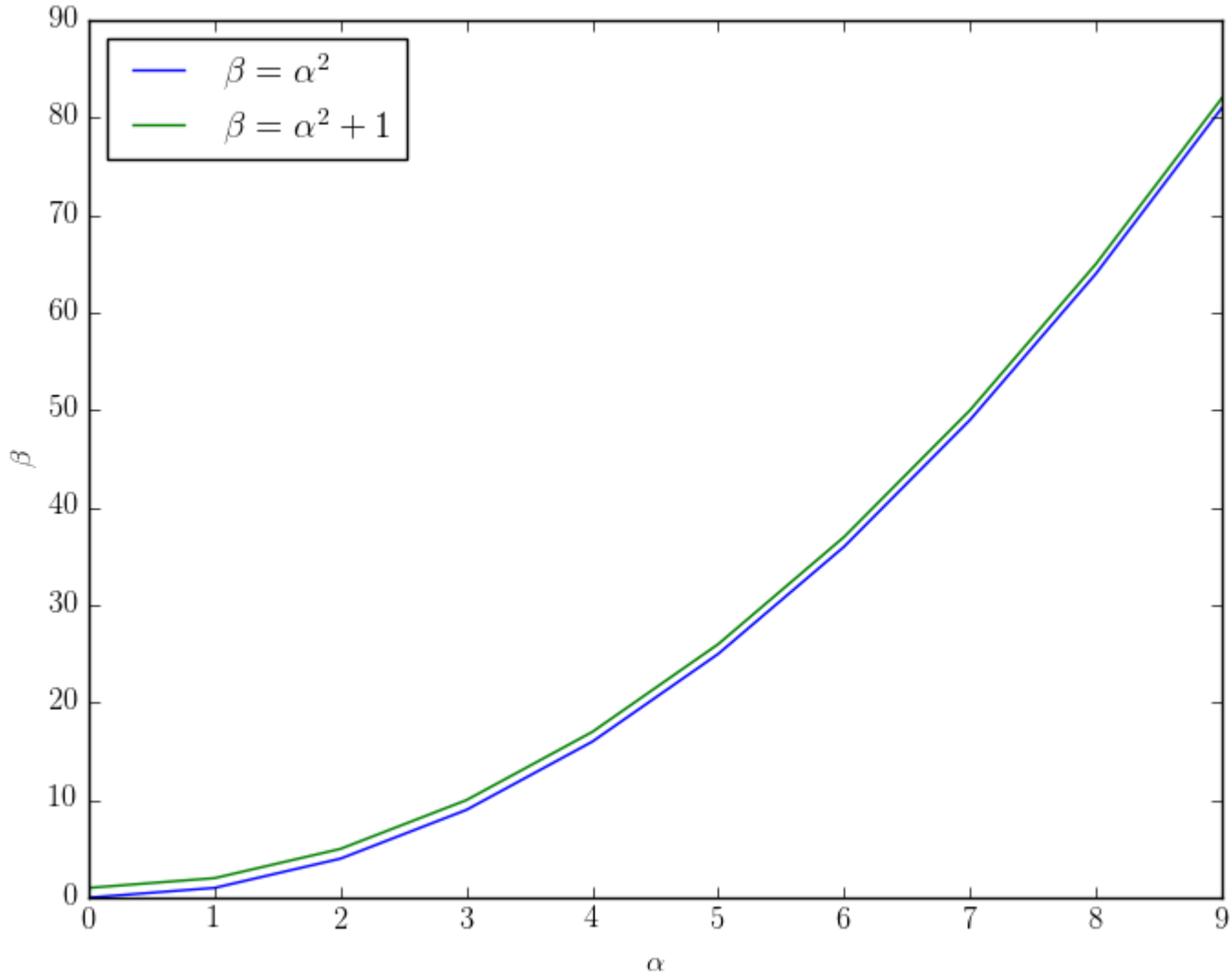
или доступ к `rcParams` :

```
import matplotlib.pyplot as plt
params = {'tex.usetex': True}
plt.rcParams.update(params)
```

TeX использует обратную косую черту `\` для команд и символов, которые могут конфликтовать со специальными символами в строках Python. Чтобы использовать литеральные обратные косые черты в строке Python, они должны быть либо экранированы, либо включены в необработанную строку:

```
plt.xlabel('\\alpha')
plt.xlabel(r'\alpha')
```

Следующий участок



может быть произведен кодом

```
import matplotlib.pyplot as plt
plt.rc(usetex = True)
x = range(0,10)
y = [t**2 for t in x]
z = [t**2+1 for t in x]
plt.plot(x, y, label = r'$\beta=\alpha^2$')
plt.plot(x, z, label = r'$\beta=\alpha^2+1$')
plt.xlabel(r'$\alpha$')
plt.ylabel(r'$\beta$')
plt.legend(loc=0)
plt.show()
```

Отображаемые уравнения (такие как `$$...$$` или `\begin{equation}...\end{equation}`) не поддерживаются. Тем не менее, отображаемый математический стиль возможен с `\displaystyle` .

Чтобы загрузить пакеты латекса, используйте аргумент `tex.latex.preamble` :

```
params = {'text.latex.preamble' : [r'\usepackage{siunitx}', r'\usepackage{amsmath}']}
plt.rcParams.update(params)
```

Обратите внимание, однако, предупреждение в примере matplotlibrc файла :

```
#text.latex.preamble : # IMPROPER USE OF THIS FEATURE WILL LEAD TO LATEX FAILURES
                       # AND IS THEREFORE UNSUPPORTED. PLEASE DO NOT ASK FOR HELP
                       # IF THIS FEATURE DOES NOT DO WHAT YOU EXPECT IT TO.
                       # preamble is a comma separated list of LaTeX statements
                       # that are included in the LaTeX document preamble.
                       # An example:
                       # text.latex.preamble : \usepackage{bm},\usepackage{euler}
                       # The following packages are always loaded with usetex, so
                       # beware of package collisions: color, geometry, graphicx,
                       # type1cm, textcomp. Adobe Postscript (PSSNFS) font packages
                       # may also be loaded, depending on your font settings
```

Сохранение и экспорт участков, использующих TeX

Чтобы включить графики, созданные с помощью matplotlib в документах TeX, они должны быть сохранены в виде файлов `pdf` или `eps` . Таким образом, любой текст в сюжете (включая формулы TeX) отображается как текст в конечном документе.

```
import matplotlib.pyplot as plt
plt.rc(usetex=True)
x = range(0, 10)
y = [t**2 for t in x]
z = [t**2+1 for t in x]
plt.plot(x, y, label=r'$\beta=\alpha^2$')
plt.plot(x, z, label=r'$\beta=\alpha^2+1$')
plt.xlabel(r'$\alpha$')
plt.ylabel(r'$\beta$')
plt.legend(loc=0)
plt.savefig('my_pdf_plot.pdf')  # Saving plot to pdf file
plt.savefig('my_eps_plot.eps')  # Saving plot to eps file
```

Сюжеты в matplotlib могут быть экспортированы в TeX-код, используя макрос `pgf` для отображения графики.

```
import matplotlib.pyplot as plt
plt.rc(usetex=True)
x = range(0, 10)
y = [t**2 for t in x]
z = [t**2+1 for t in x]
plt.plot(x, y, label=r'$\beta=\alpha^2$')
plt.plot(x, z, label=r'$\beta=\alpha^2+1$')
plt.xlabel(r'$\alpha$')
plt.ylabel(r'$\beta$')
plt.legend(loc=0)
plt.savefig('my_pgf_plot.pgf')
```

Используйте команду `rc` для изменения используемого движка TeX.

```
plt.rc('pgf', texsystem='pdflatex')  # or luatex, xelatex...
```

Чтобы включить цифру `.pgf` , напишите в своем документе LaTeX

```
\usepackage{pgf}
\input{my_pgf_plot.pgf}
```

Прочитайте Интеграция с TeX / LaTeX онлайн: https://riptutorial.com/ru/matplotlib/topic/2962/интеграция-с-tex---latex

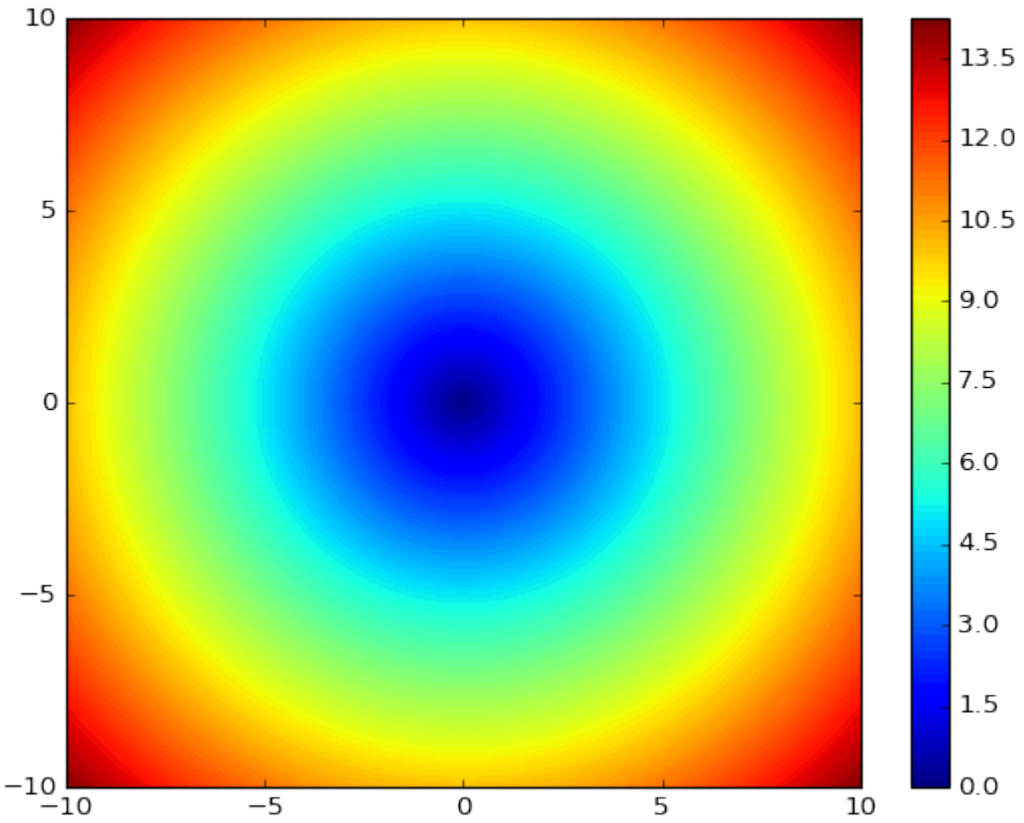
глава 7: Контурные карты

Examples

Простой заполненный контурный график

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

# generate 101 x and y values between -10 and 10
x = np.linspace(-10, 10, 101)
y = np.linspace(-10, 10, 101)

# make X and Y matrices representing x and y values of 2d plane
X, Y = np.meshgrid(x, y)

# compute z value of a point as a function of x and y (z = l2 distance form 0,0)
Z = np.sqrt(X ** 2 + Y ** 2)

# plot filled contour map with 100 levels
cs = plt.contourf(X, Y, Z, 100)

# add default colorbar for the map
plt.colorbar(cs)
```

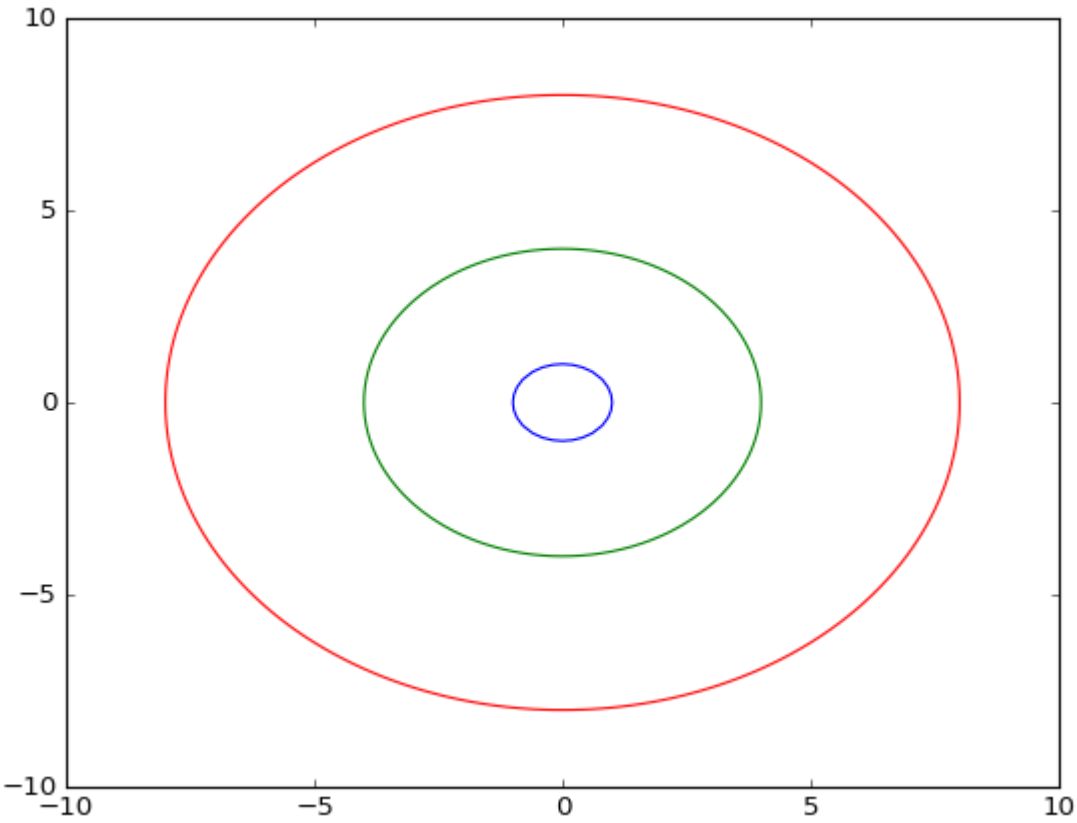
Результат:



Простая контурная печать

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

# generate 101 x and y values between -10 and 10
x = np.linspace(-10, 10, 101)
y = np.linspace(-10, 10, 101)

# make X and Y matrices representing x and y values of 2d plane
X, Y = np.meshgrid(x, y)

# compute z value of a point as a function of x and y (z = l2 distance form 0,0)
Z = np.sqrt(X ** 2 + Y ** 2)

# plot contour map with 3 levels
# colors: up to 1 - blue,  from 1 to 4 - green, from 4 to 8 - red
plt.contour(X, Y, Z, [1, 4, 8], colors=['b', 'g', 'r'])
```

Результат:



Прочитайте Контурные карты онлайн: https://riptutorial.com/ru/matplotlib/topic/8644/контурные-карты

глава 8: Легенды

Examples

Простая легенда

Предположим, у вас есть несколько строк на одном и том же сюжете, каждый из другого цвета, и вы хотите сделать легенду, чтобы сообщить, что представляет каждая строка. Вы можете сделать это, передав метку каждой строке при вызове `plot()` , например, следующая строка будет помечена как *«Моя линия 1»* .

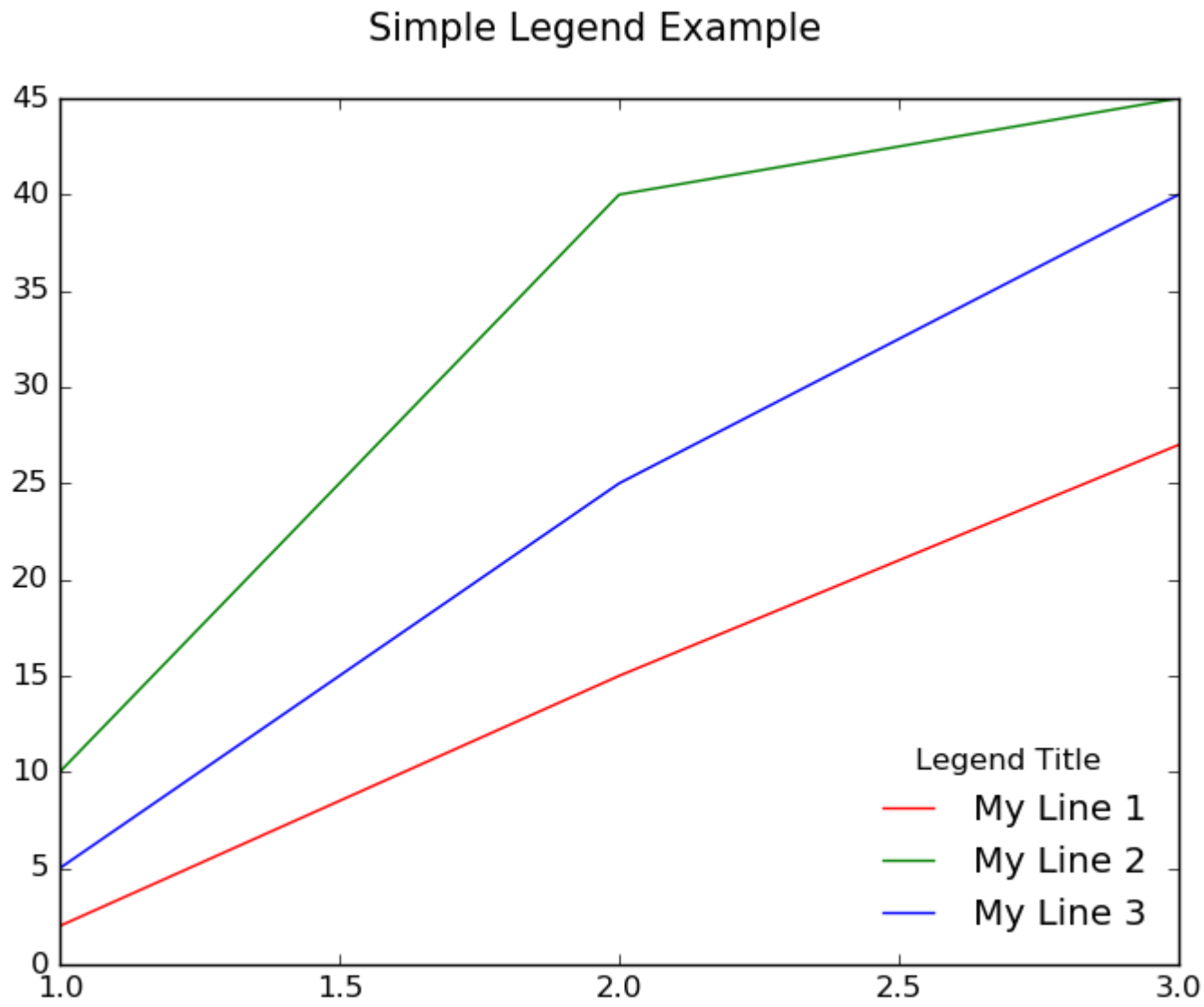
```
ax.plot(x, y1, color="red", label="My Line 1")
```

Это указывает текст, который будет отображаться в легенде для этой строки. Теперь, чтобы сделать реальную легенду видимой, мы можем вызвать `ax.legend()`

По умолчанию он создает легенду внутри поля в правом верхнем углу графика. Вы можете передать аргументы в `legend()` чтобы настроить его. Например, мы можем поместить его в нижнем правом углу, без рамки рамки вокруг него и создания заголовка для легенды, вызвав следующее:

```
ax.legend(loc="lower right", title="Legend Title", frameon=False)
```

Ниже приведен пример:

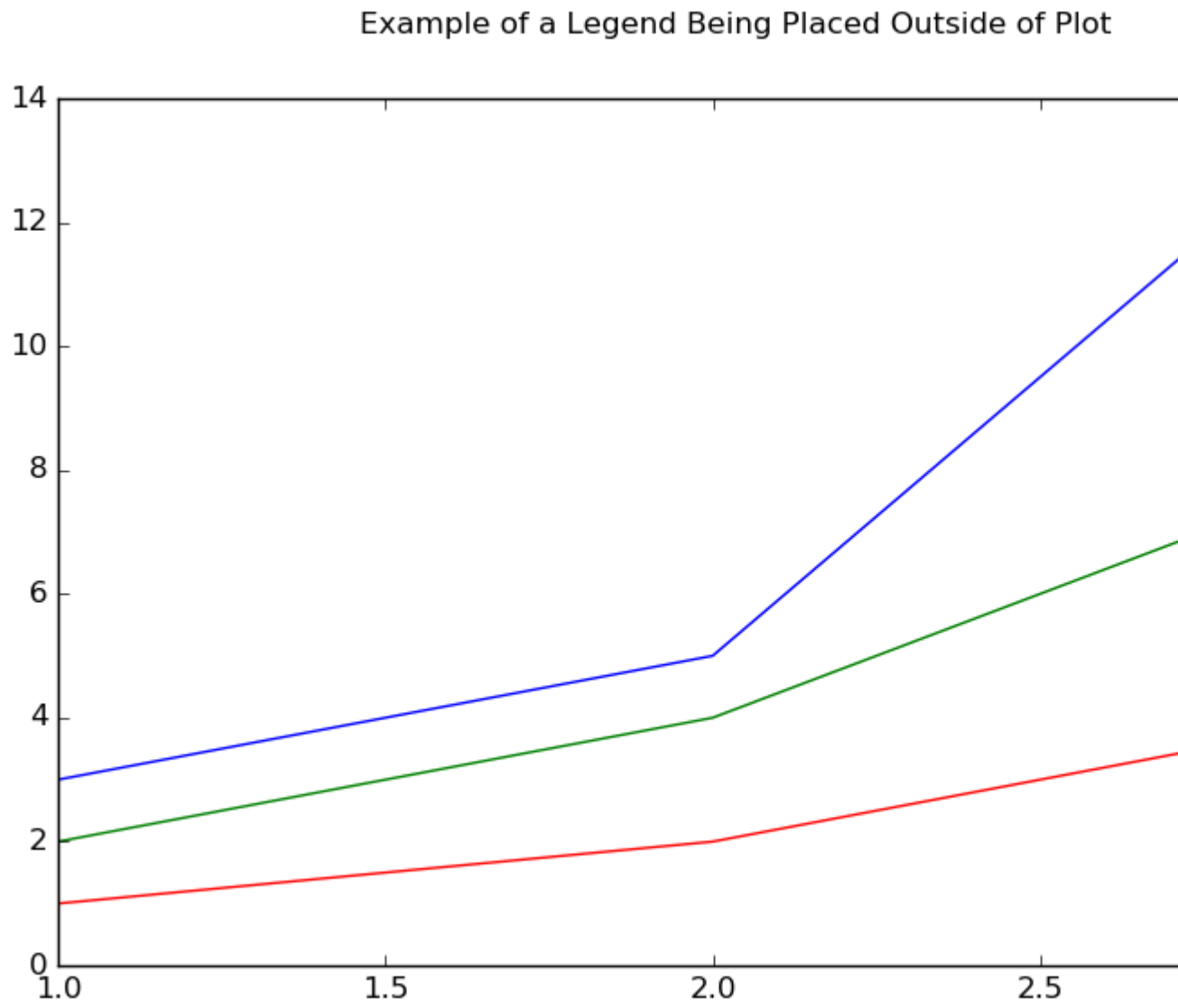



```
import matplotlib.pyplot as plt

# The data
x =  [1, 2, 3]
y1 = [2,  15, 27]
y2 = [10, 40, 45]
y3 = [5,  25, 40]

# Initialize the figure and axes
fig, ax = plt.subplots(1, figsize=(8, 6))

# Set the title for the figure
fig.suptitle('Simple Legend Example ', fontsize=15)

# Draw all the lines in the same plot, assigning a label for each one to be
# shown in the legend
ax.plot(x, y1, color="red", label="My Line 1")
ax.plot(x, y2, color="green", label="My Line 2")
ax.plot(x, y3, color="blue", label="My Line 3")

# Add a legend with title, position it on the lower right (loc) with no box framing (frameon)
ax.legend(loc="lower right", title="Legend Title", frameon=False)
```

```
# Show the plot
plt.show()
```

Легенда, размещенная за пределами участка

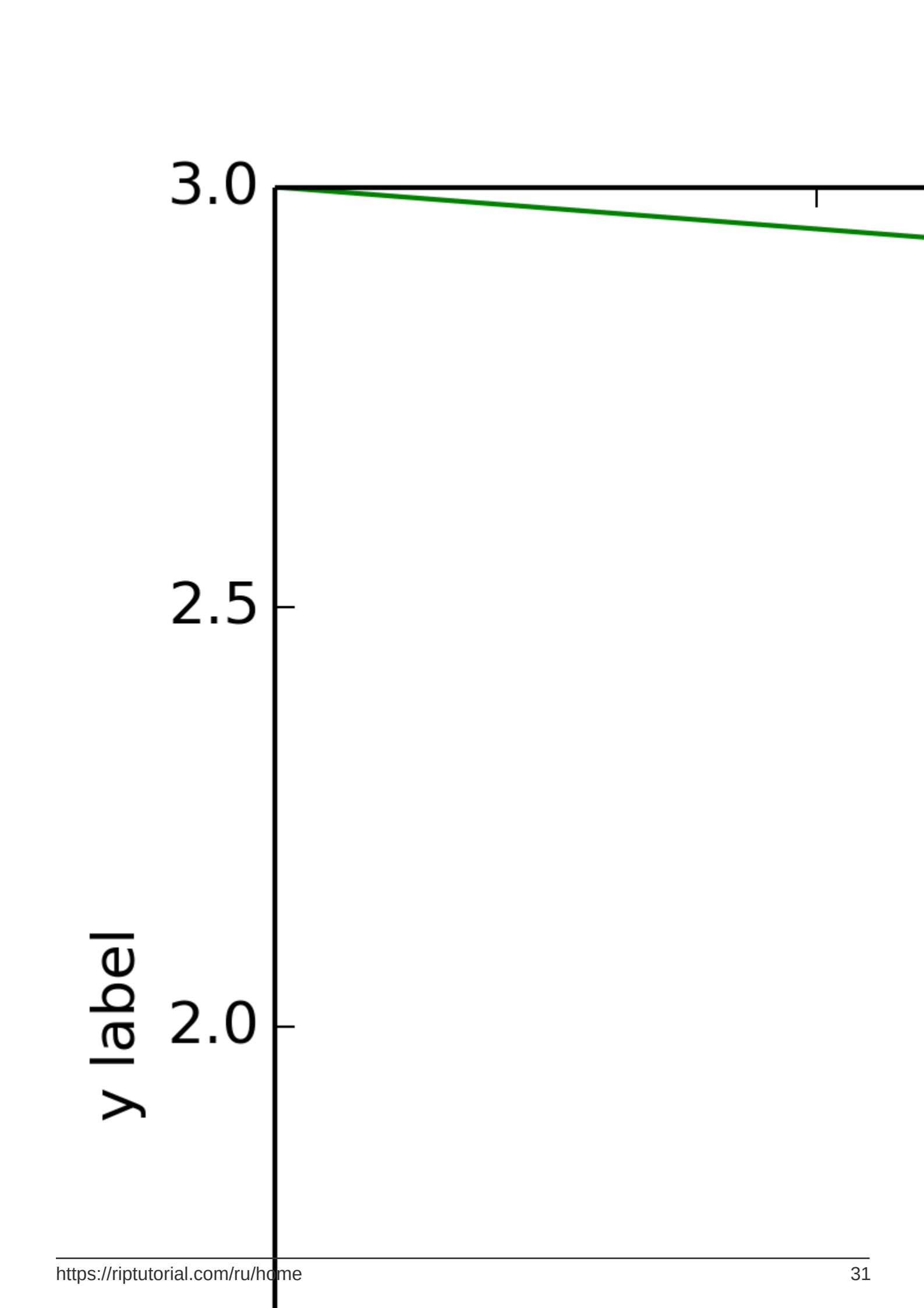
Иногда необходимо или желательно разместить легенду вне сюжета. Следующий код показывает, как это сделать.




```
import matplotlib.pylab as plt
fig, ax = plt.subplots(1, 1, figsize=(10,6)) # make the figure with the size 10 x 6 inches
fig.suptitle('Example of a Legend Being Placed Outside of Plot')

# The data
x =  [1, 2, 3]
y1 = [1, 2, 4]
y2 = [2, 4, 8]
y3 = [3, 5, 14]

# Labels to use for each line
line_labels = ["Item A", "Item B", "Item C"]
```

```
# Create the lines, assigning different colors for each one.
# Also store the created line objects
l1 = ax.plot(x, y1, color="red")[0]
l2 = ax.plot(x, y2, color="green")[0]
l3 = ax.plot(x, y3, color="blue")[0]

fig.legend([l1, l2, l3],              # List of the line objects
           labels= line_labels,       # The labels for each line
           loc="center right",        # Position of the legend
           borderaxespad=0.1,         # Add little spacing around the legend box
           title="Legend Title")      # Title for the legend

# Adjust the scaling factor to fit your legend text completely outside the plot
# (smaller value results in more space being made for the legend)
plt.subplots_adjust(right=0.85)

plt.show()
```

Другой способ разместить легенду вне сюжета - использовать `bbox_to_anchor` + `bbox_extra_artists` + `bbox_inches='tight'` , как показано в примере ниже:


3.0
2.5
2.0
y label

вместо создания легенды на уровне *осей* (которая создаст отдельную легенду для каждого подзаголовка). Это достигается путем вызова `fig.legend()` как это видно из кода для следующего кода.

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(10,4))
fig.suptitle('Example of a Single Legend Shared Across Multiple Subplots')

# The data
x =  [1, 2, 3]
y1 = [1, 2, 3]
y2 = [3, 1, 3]
y3 = [1, 3, 1]
y4 = [2, 2, 3]

# Labels to use in the legend for each line
line_labels = ["Line A", "Line B", "Line C", "Line D"]

# Create the sub-plots, assigning a different color for each line.
# Also store the line objects created
l1 = ax1.plot(x, y1, color="red")[0]
l2 = ax2.plot(x, y2, color="green")[0]
l3 = ax3.plot(x, y3, color="blue")[0]
l4 = ax3.plot(x, y4, color="orange")[0] # A second line in the third subplot

# Create the legend
fig.legend([l1, l2, l3, l4],     # The line objects
           labels=line_labels,   # The labels for each line
           loc="center right",   # Position of legend
           borderaxespad=0.1,    # Small spacing around legend box
           title="Legend Title"  # Title for the legend
           )

# Adjust the scaling factor to fit your legend text completely outside the plot
# (smaller value results in more space being made for the legend)
plt.subplots_adjust(right=0.85)

plt.show()
```

Что-то, что следует отметить в приведенном выше примере, следующее:

```
l1 = ax1.plot(x, y1, color="red")[0]
```

Когда вызывается `plot()` , он возвращает список объектов **line2D** . В этом случае он просто возвращает список с одним единственным объектом *line2D* , который извлекается с индексированием `[0]` и сохраняется в `l1` .

Список всех объектов *line2D,* которые мы заинтересованы в включении в легенду, должен быть передан в качестве первого аргумента `fig.legend()` . Также необходим второй аргумент `fig.legend()` . Он должен быть списком строк для использования в качестве меток для каждой строки в легенде.

Другие аргументы, переданные `fig.legend()` являются необязательными и просто помогают с тонкой настройкой эстетики легенды.

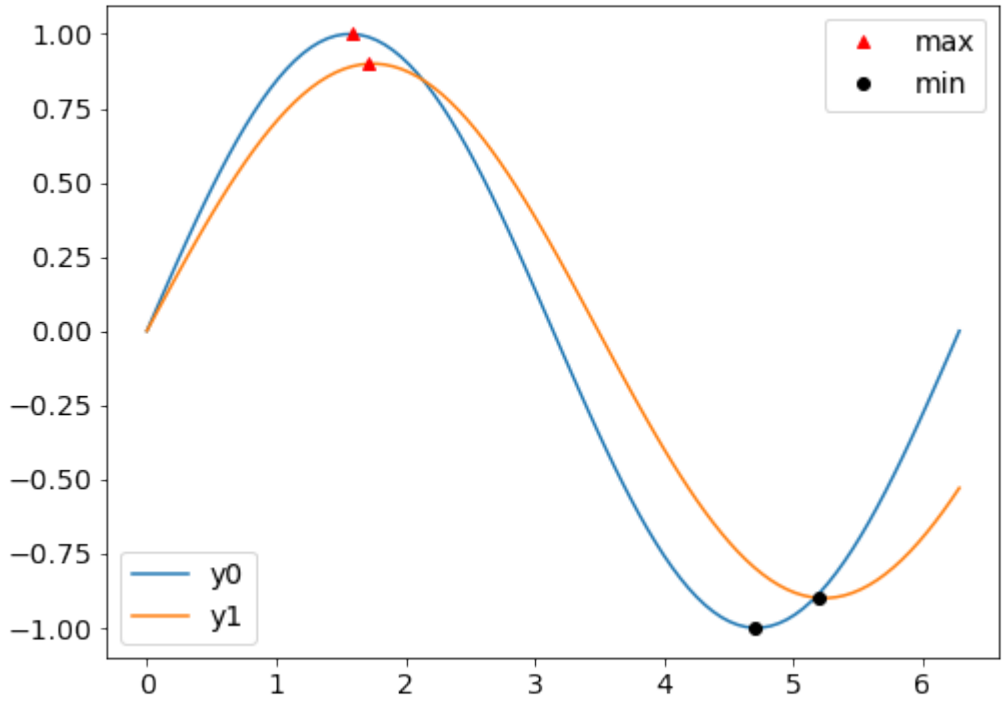
Несколько легенд о тех же топорах

Если вы вызываете `plt.legend()` или `ax.legend()` более одного раза, первая легенда удаляется и создается новый. Согласно официальной документации :

> Это было сделано так, чтобы можно было вызвать legend () несколько раз, чтобы обновить легенду до последних дескрипторов на оси

Не бойтесь, хотя: до сих пор довольно просто добавить вторую легенду (или третью, или четвертую ...) к осям. В примере здесь мы построим две строки, затем построим маркеры по их соответствующим максимумам и минимумам. Одна легенда относится к строкам, а другая - к маркерам.

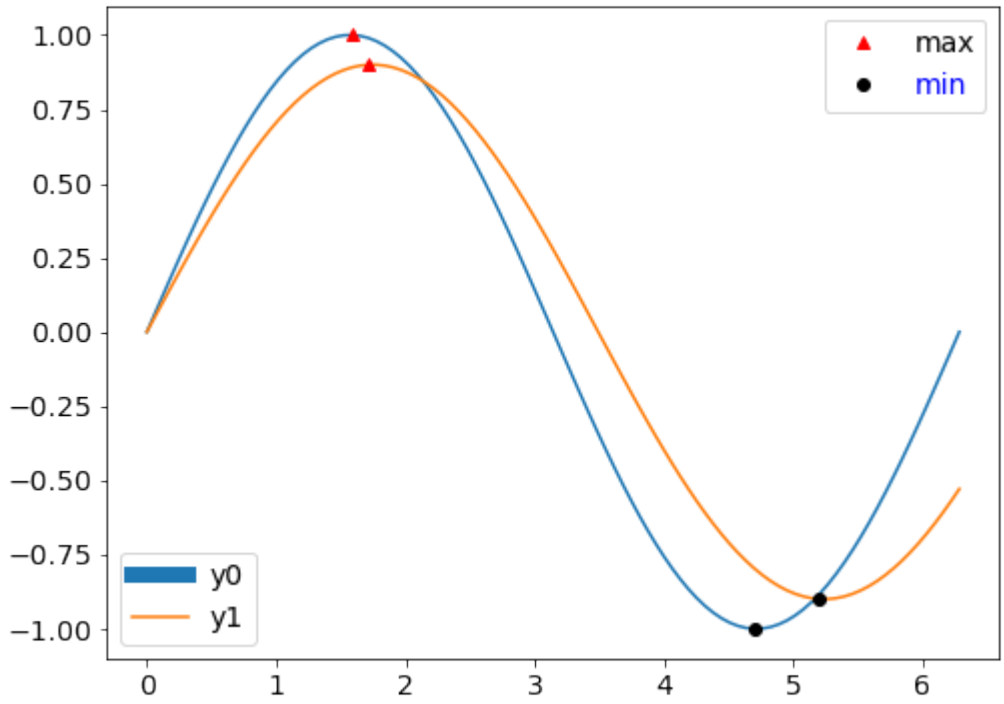
```
import matplotlib.pyplot as plt
import numpy as np

# Generate data for plotting:
x = np.linspace(0,2*np.pi,100)
y0 = np.sin(x)
y1 = .9*np.sin(.9*x)
# Find their maxima and minima and store
maxes = np.empty((2,2))
mins = np.empty((2,2))
for k,y in enumerate([y0,y1]):
    maxloc = y.argmax()
    maxes[k] = x[maxloc], y[maxloc]
    minloc = y.argmin()
    mins[k] = x[minloc], y[minloc]

# Instantiate figure and plot
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x,y0, label='y0')
ax.plot(x,y1, label='y1')
# Plot maxima and minima, and keep references to the lines
maxline, = ax.plot(maxes[:,0], maxes[:,1], 'r^')
minline, = ax.plot(mins[:,0], mins[:,1], 'ko')

# Add first legend:  only labeled data is included
leg1 = ax.legend(loc='lower left')
# Add second legend for the maxes and mins.
# leg1 will be removed from figure
leg2 = ax.legend([maxline,minline],['max','min'], loc='upper right')
# Manually add the first legend back
ax.add_artist(leg1)
```



Ключ должен убедиться, что у вас есть ссылки на объекты легенды. Первый экземпляр (`leg1`) удаляется из рисунка при добавлении второго, но объект `leg1` все еще существует и может быть добавлен обратно с помощью `ax.add_artist` .

Самое замечательное, что вы все еще можете манипулировать *обоими* легендами. Например, добавьте следующее в нижнюю часть приведенного выше кода:

```
leg1.get_lines()[0].set_lw(8)
leg2.get_texts()[1].set_color('b')
```



Наконец, стоит упомянуть, что в примере только строкам были нанесены метки при

построении графика, что означает, что `ax.legend()` добавляет только те строки в `leg1` . Поэтому легенда для маркеров (`leg2`) требовала, чтобы строки и метки были аргументами, когда они были созданы. Мы могли бы, в качестве альтернативы, дать метки маркерам, когда они были заложены. Но тогда *оба* вызова `ax.legend` потребовали бы дополнительных аргументов, чтобы каждая легенда содержала только те предметы, которые мы хотели.

Прочитайте Легенды онлайн: https://riptutorial.com/ru/matplotlib/topic/2840/легенды

глава 9: Манипуляция изображениями

Examples

Открытие изображений

Matplotlib включает `image` модуля для обработки изображений

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
```

Изображения читаются из файла (только `.png`) с функцией `imread` :

```
img = mpimg.imread('my_image.png')
```

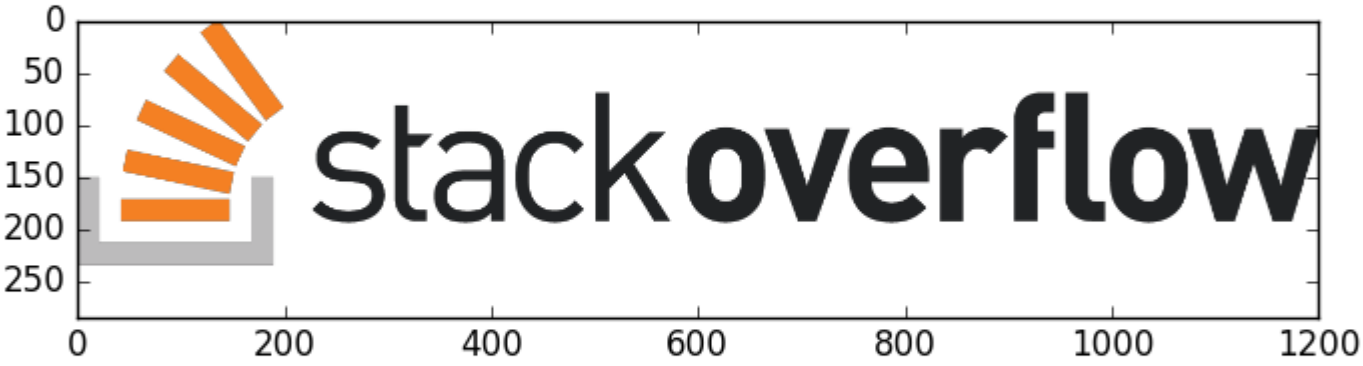
и они оказываются функцией `imshow` :

```
plt.imshow(img)
```

Давайте *построим* логотип переполнением стека :

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
img = mpimg.imread('so-logo.png')
plt.imshow(img)
plt.show()
```

Полученный график



Прочитайте Манипуляция изображениями онлайн:
https://riptutorial.com/ru/matplotlib/topic/4575/манипуляция-изображениями

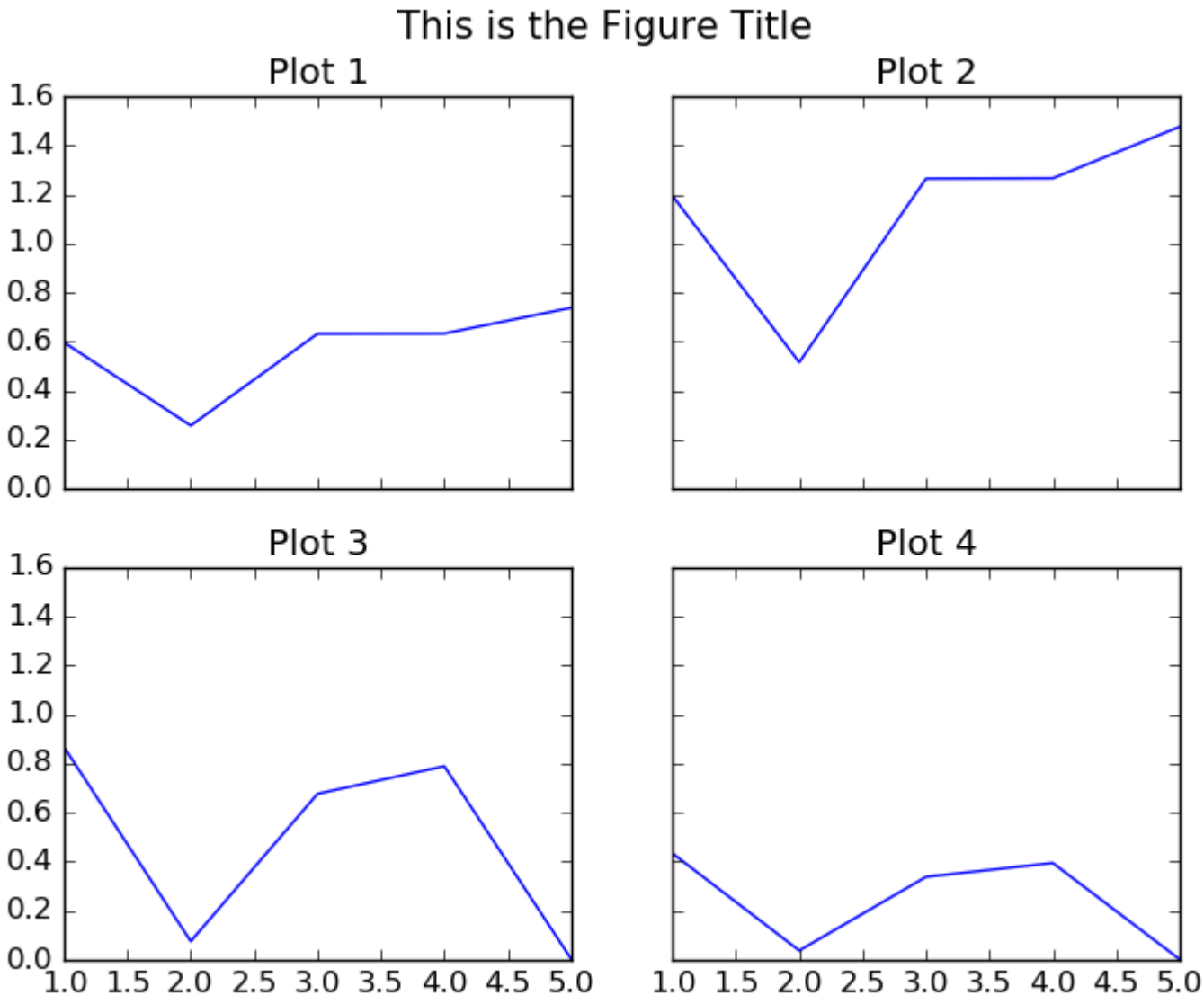
глава 10: Несколько участков

Синтаксис

- Элемент списка

Examples

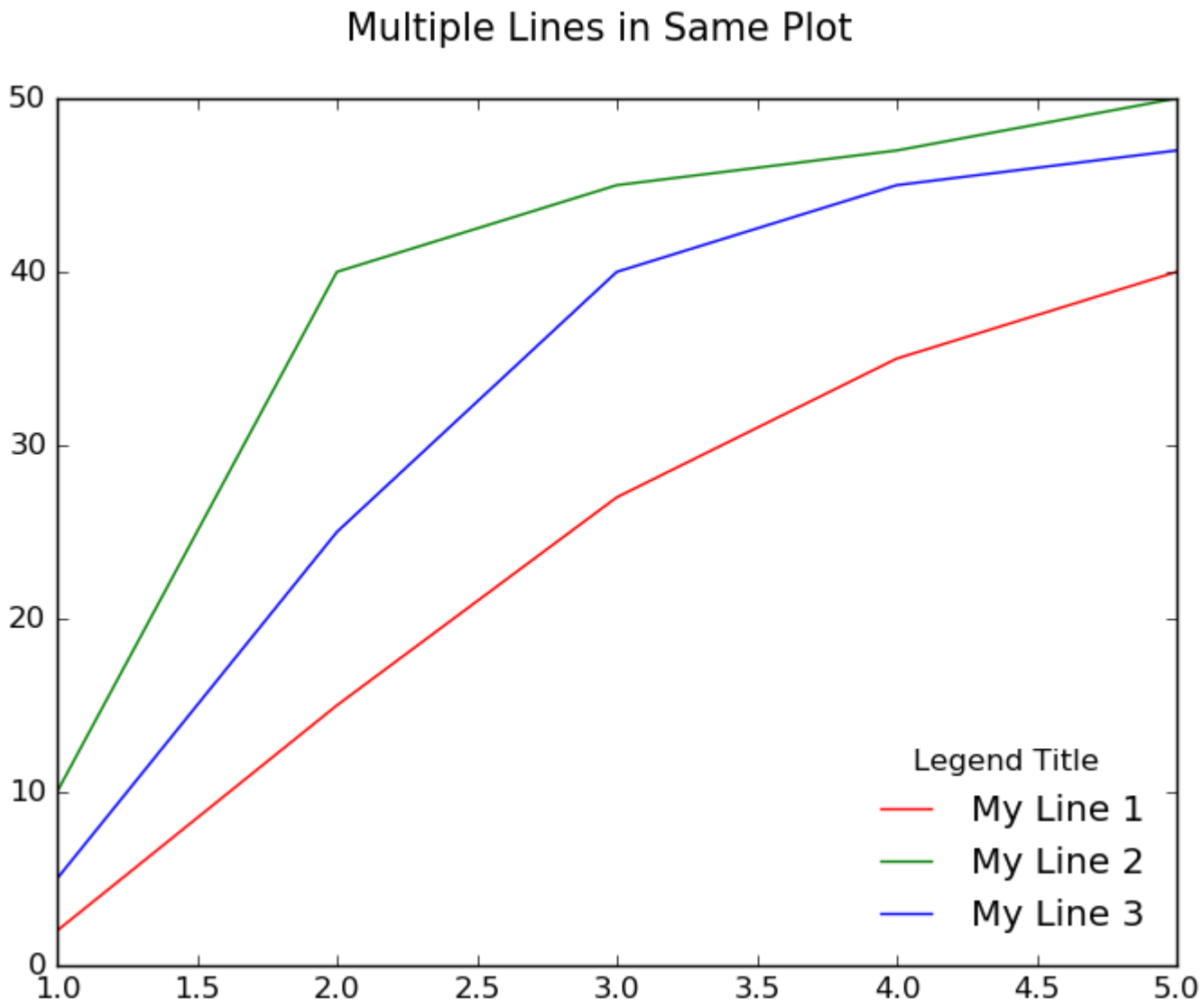
Сетка подзаголовков с использованием подзаголовка




```
"""
=============================================================================
CREATE A 2 BY 2 GRID OF SUB-PLOTS WITHIN THE SAME FIGURE.
=============================================================================
"""
import matplotlib.pyplot as plt
```

```
# The data
x = [1,2,3,4,5]
y1 = [0.59705847, 0.25786401, 0.63213726, 0.63287317, 0.73791151]
y2 = [1.19411694, 0.51572803, 1.26427451, 1.26574635, 1.47582302]
y3 = [0.86793828, 0.07563408, 0.67670068, 0.78932712, 0.0043694]
# 5 more random values
y4 = [0.43396914, 0.03781704, 0.33835034, 0.39466356, 0.0021847]

# Initialise the figure and a subplot axes. Each subplot sharing (showing) the
# same range of values for the x and y axis in the plots.
fig, axes = plt.subplots(2, 2, figsize=(8, 6), sharex=True, sharey=True)

# Set the title for the figure
fig.suptitle('This is the Figure Title', fontsize=15)

# Top Left Subplot
axes[0,0].plot(x, y1)
axes[0,0].set_title("Plot 1")

# Top Right Subplot
axes[0,1].plot(x, y2)
axes[0,1].set_title("Plot 2")

# Bottom Left Subplot
axes[1,0].plot(x, y3)
axes[1,0].set_title("Plot 3")

# Bottom Right Subplot
axes[1,1].plot(x, y4)
axes[1,1].set_title("Plot 4")

plt.show()
```

Несколько строк / кривых на одном и том же участке




```
"""
================================================================================
                    DRAW MULTIPLE LINES IN THE SAME PLOT
================================================================================
"""
import matplotlib.pyplot as plt

# The data
x =  [1, 2, 3, 4, 5]
y1 = [2,  15, 27, 35, 40]
y2 = [10, 40, 45, 47, 50]
y3 = [5,  25, 40, 45, 47]

# Initialise the figure and axes.
fig, ax = plt.subplots(1, figsize=(8, 6))

# Set the title for the figure
fig.suptitle('Multiple Lines in Same Plot', fontsize=15)

# Draw all the lines in the same plot, assigning a label for each one to be
# shown in the legend.
ax.plot(x, y1, color="red", label="My Line 1")
ax.plot(x, y2, color="green", label="My Line 2")
```

```
ax.plot(x, y3, color="blue", label="My Line 3")

# Add a legend, and position it on the lower right (with no box)
plt.legend(loc="lower right", title="Legend Title", frameon=False)

plt.show()
```

Несколько участков с сеткой

Пакет `gridspec` позволяет больше контролировать размещение подзаголовков. Это значительно упрощает управление границами участков и интервалом между отдельными подзаголовками. Кроме того, он позволяет использовать оси различного размера на одном и том же рисунке, определяя оси, которые занимают несколько мест сетки.

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

# Make some data
t = np.arange(0, 2, 0.01)
y1 = np.sin(2*np.pi * t)
y2 = np.cos(2*np.pi * t)
y3 = np.exp(t)
y4 = np.exp(-t)

# Initialize the grid with 3 rows and 3 columns
ncols = 3
nrows = 3
grid = GridSpec(nrows, ncols,
                left=0.1, bottom=0.15, right=0.94, top=0.94, wspace=0.3, hspace=0.3)

fig = plt.figure(0)
fig.clf()

# Add axes which can span multiple grid boxes
ax1 = fig.add_subplot(grid[0:2, 0:2])
ax2 = fig.add_subplot(grid[0:2, 2])
ax3 = fig.add_subplot(grid[2, 0:2])
ax4 = fig.add_subplot(grid[2, 2])

ax1.plot(t, y1, color='royalblue')
ax2.plot(t, y2, color='forestgreen')
ax3.plot(t, y3, color='darkorange')
ax4.plot(t, y4, color='darkmagenta')

# Add labels and titles
fig.suptitle('Figure with Subplots')
ax1.set_ylabel('Voltage (V)')
ax3.set_ylabel('Voltage (V)')
ax3.set_xlabel('Time (s)')
ax4.set_xlabel('Time (s)')
```

Этот код создает график, показанный ниже.

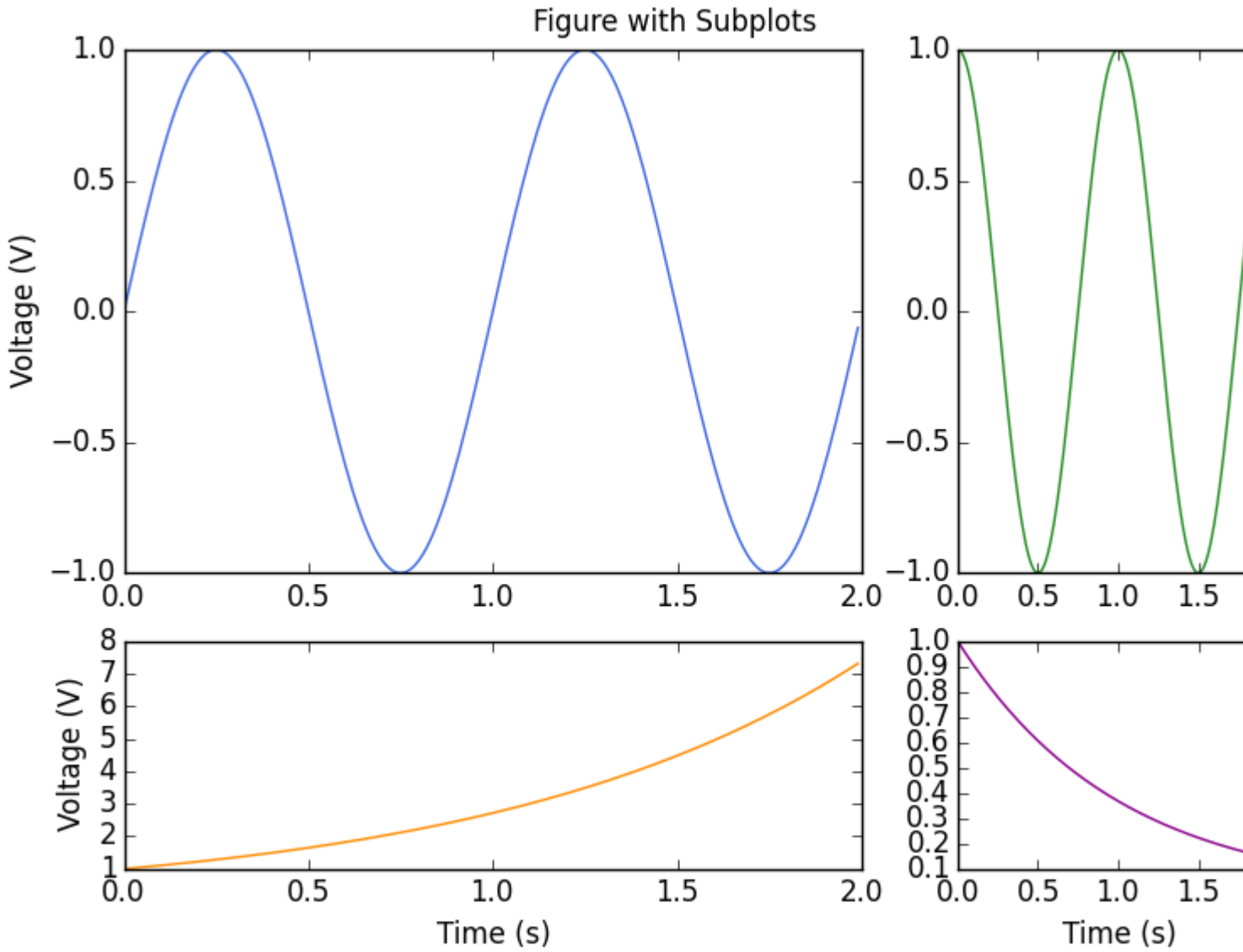
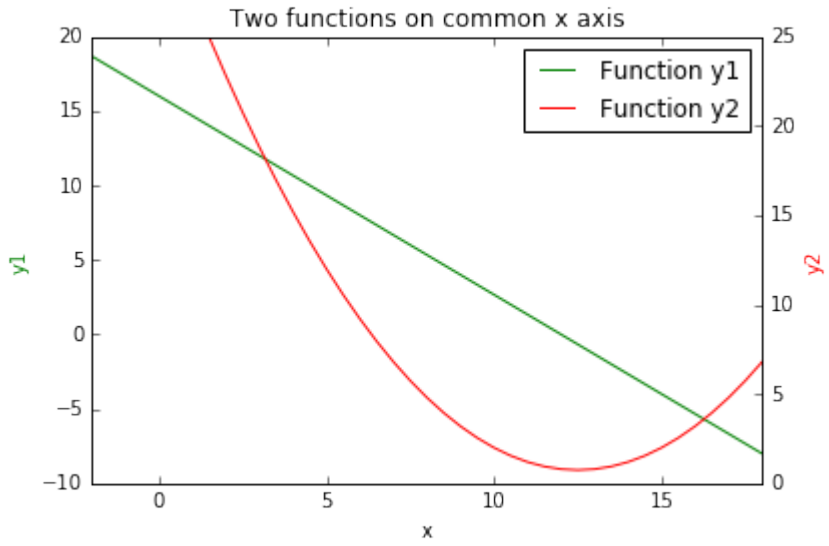


График 2 функций на общей оси x.

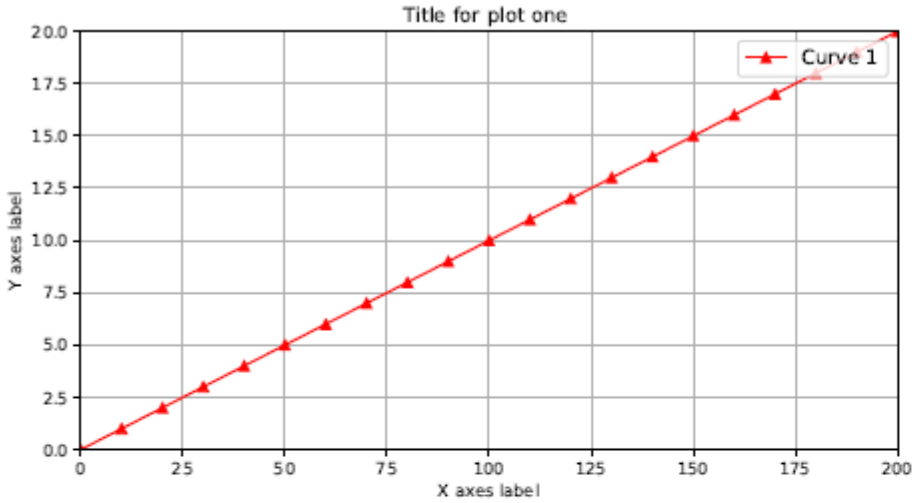
```
import numpy as np
import matplotlib.pyplot as plt

# create some data
x = np.arange(-2, 20, 0.5)                  # values of x
y1 = map(lambda x: -4.0/3.0*x + 16, x)      # values of y1(x)
y2 = map(lambda x: 0.2*x**2 -5*x + 32, x)   # svalues of y2(x)

fig = plt.figure()
ax1 = fig.add_subplot(111)

# create line plot of y1(x)
line1, = ax1.plot(x, y1, 'g', label="Function y1")
ax1.set_xlabel('x')
ax1.set_ylabel('y1', color='g')

# create shared axis for y2(x)
ax2 = ax1.twinx()
```

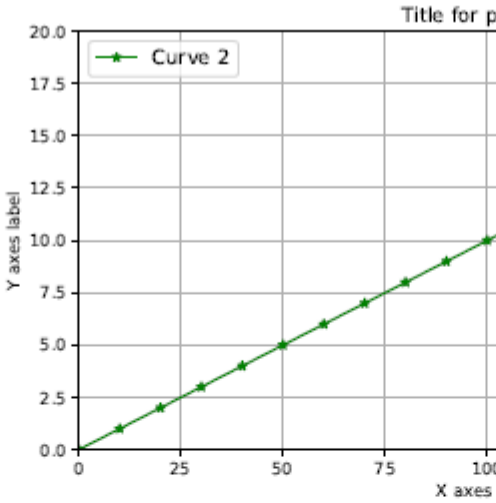
```
# create line plot of y2(x)
line2, = ax2.plot(x, y2, 'r', label="Function y2")
ax2.set_ylabel('y2', color='r')

# set title, plot limits, etc
plt.title('Two functions on common x axis')
plt.xlim(-2, 18)
plt.ylim(0, 25)

# add a legend, and position it on the upper right
plt.legend((line1, line2), ('Function y1', 'Function y2'))

plt.show()
```

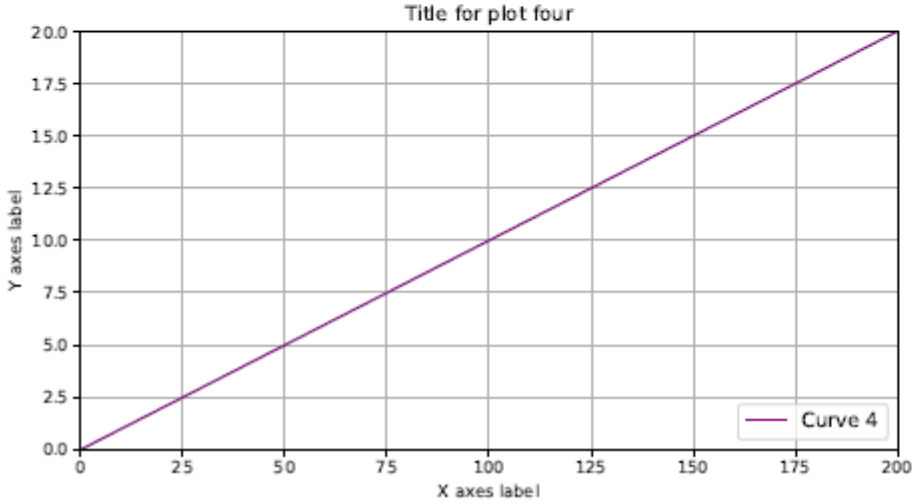
Этот код создает график, показанный ниже.

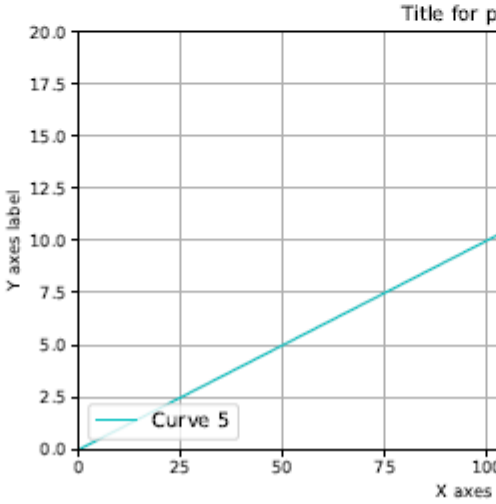


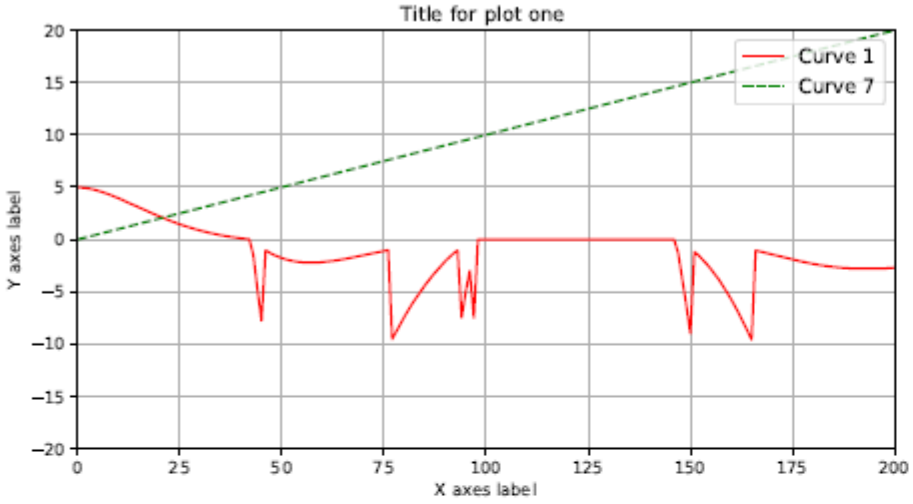
Множество сюжетов и многострочных объектов

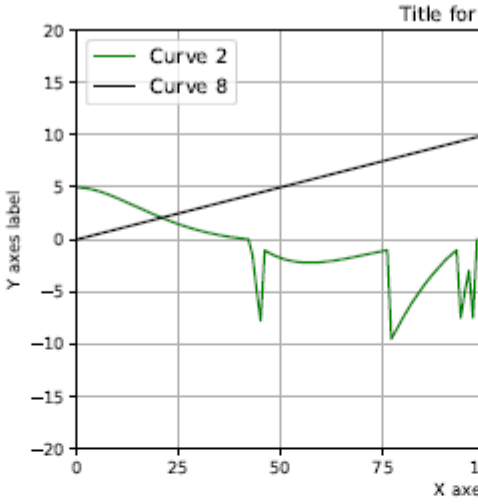

Title for plot one
Curve 1
Y axes label
X axes label
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
25
50
75
100
125
150
175
200

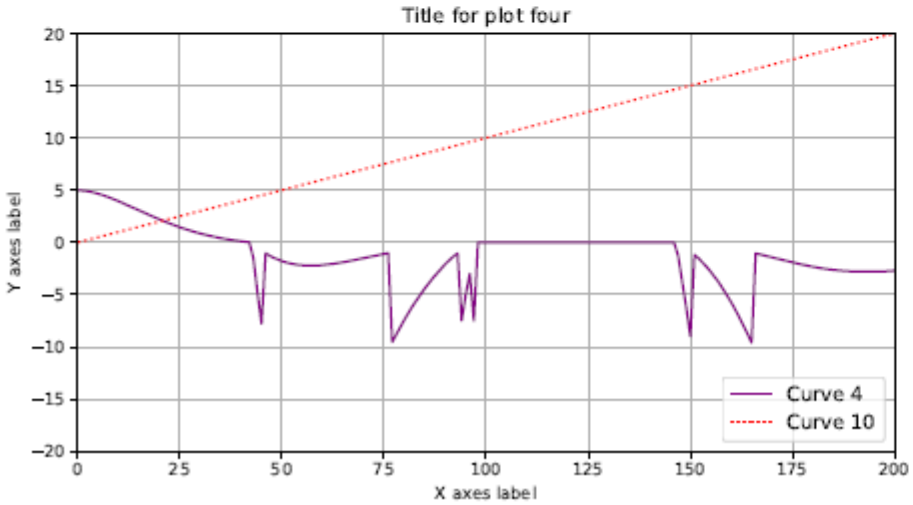

Curve 2
Y axes label
X axes

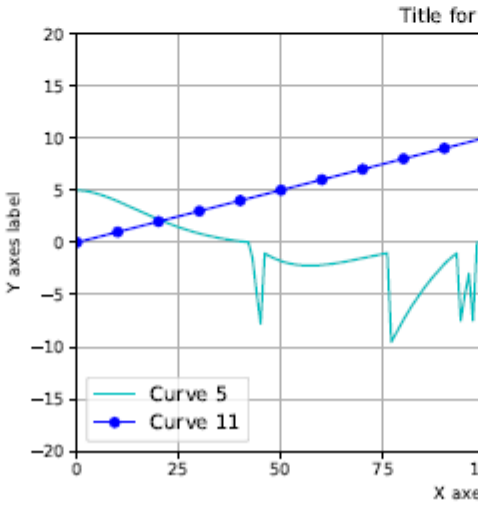

Title for plot four
Curve 4
Y axes label
X axes label
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
25
50
75
100
125
150
175
200


Curve 5
Y axes label
X axes









CAE.csv

```
TIME,Acceleration
0,4.992235
0.09952711,4.956489
0.1999273,4.915645
0.2994544,4.850395
0.3998545,4.763977
0.4993816,4.65888
0.5997818,4.537595
0.6993089,4.402862
0.799709,4.256423
0.8992361,4.100522
0.9996362,3.937148
1.099163,3.768047
1.199564,3.579082
```

```
import matplotlib
matplotlib.use("TKAgg")

# module to save pdf files
from matplotlib.backends.backend_pdf import PdfPages

import matplotlib.pyplot as plt  # module to plot

import pandas as pd  # module to read csv file
```

```
# module to allow user to select csv file
from tkinter.filedialog import askopenfilename

# module to allow user to select save directory
from tkinter.filedialog import askdirectory


#=============================================================================
#  User chosen Data for plots
#=============================================================================

# User choose csv file then read csv file
filename = askopenfilename() # user selected file
data = pd.read_csv(filename, delimiter=',')

# check to see if data is reading correctly
#print(data)

#=============================================================================
#  Plots on two different Figures and sets the size of the figures
#=============================================================================

# figure size = (width,height)
f1 = plt.figure(figsize=(30,10))
f2 = plt.figure(figsize=(30,10))

#-----------------------------------------------------------------------------
#  Figure 1 with 6 plots
#-----------------------------------------------------------------------------

# plot one
# Plot column labeled TIME from csv file and color it red
# subplot(2 Rows, 3 Columns, First subplot,)
ax1 = f1.add_subplot(2,3,1)
ax1.plot(data[["TIME"]], label = 'Curve 1', color = "r", marker = '^', markevery = 10)
# added line marker triangle


# plot two
# plot column labeled TIME from csv file and color it green
# subplot(2 Rows, 3 Columns, Second subplot)
ax2 = f1.add_subplot(2,3,2)
ax2.plot(data[["TIME"]], label = 'Curve 2', color = "g", marker = '*', markevery = 10)
# added line marker star


# plot three
# plot column labeled TIME from csv file and color it blue
# subplot(2 Rows, 3 Columns, Third subplot)
ax3 = f1.add_subplot(2,3,3)
ax3.plot(data[["TIME"]], label = 'Curve 3', color = "b", marker = 'D', markevery = 10)
# added line marker diamond


# plot four
# plot column labeled TIME from csv file and color it purple
# subplot(2 Rows, 3 Columns, Fourth subplot)
ax4 = f1.add_subplot(2,3,4)
ax4.plot(data[["TIME"]], label = 'Curve 4', color = "#800080")
```

```
# plot five
# plot column labeled TIME from csv file and color it cyan
# subplot(2 Rows, 3 Columns, Fifth subplot)
ax5 = f1.add_subplot(2,3,5)
ax5.plot(data[["TIME"]], label = 'Curve 5', color = "c")


# plot six
# plot column labeled TIME from csv file and color it black
# subplot(2 Rows, 3 Columns, Sixth subplot)
ax6 = f1.add_subplot(2,3,6)
ax6.plot(data[["TIME"]], label = 'Curve 6', color = "k")

#-------------------------------------------------------------------------------
# Figure 2 with 6 plots
#-------------------------------------------------------------------------------

# plot one
# Curve 1: plot column labeled Acceleration from csv file and color it red
# Curve 2: plot column labeled     TIME     from csv file and color it green
# subplot(2 Rows, 3 Columns, First subplot)
ax10 = f2.add_subplot(2,3,1)
ax10.plot(data[["Acceleration"]], label = 'Curve 1', color = "r")
ax10.plot(data[["TIME"]], label = 'Curve 7', color="g", linestyle ='--')
# dashed line


# plot two
# Curve 1: plot column labeled Acceleration from csv file and color it green
# Curve 2: plot column labeled     TIME     from csv file and color it black
# subplot(2 Rows, 3 Columns, Second subplot)
ax20 = f2.add_subplot(2,3,2)
ax20.plot(data[["Acceleration"]], label = 'Curve 2', color = "g")
ax20.plot(data[["TIME"]], label = 'Curve 8', color = "k", linestyle ='-')
# solid line (default)


# plot three
# Curve 1: plot column labeled Acceleration from csv file and color it blue
# Curve 2: plot column labeled     TIME     from csv file and color it purple
# subplot(2 Rows, 3 Columns, Third subplot)
ax30 = f2.add_subplot(2,3,3)
ax30.plot(data[["Acceleration"]], label = 'Curve 3', color = "b")
ax30.plot(data[["TIME"]], label = 'Curve 9', color = "#800080", linestyle ='-.')
# dash_dot line


# plot four
# Curve 1: plot column labeled Acceleration from csv file and color it purple
# Curve 2: plot column labeled     TIME     from csv file and color it red
# subplot(2 Rows, 3 Columns, Fourth subplot)
ax40 = f2.add_subplot(2,3,4)
ax40.plot(data[["Acceleration"]], label = 'Curve 4', color = "#800080")
ax40.plot(data[["TIME"]], label = 'Curve 10', color = "r", linestyle =':')
# dotted line


# plot five
# Curve 1: plot column labeled Acceleration from csv file and color it cyan
# Curve 2: plot column labeled     TIME     from csv file and color it blue
# subplot(2 Rows, 3 Columns, Fifth subplot)
```

```
ax50 = f2.add_subplot(2,3,5)
ax50.plot(data[["Acceleration"]], label = 'Curve 5', color = "c")
ax50.plot(data[["TIME"]], label = 'Curve 11', color = "b", marker = 'o', markevery = 10)
# added line marker circle


# plot six
# Curve 1: plot column labeled Acceleration from csv file and color it black
# Curve 2: plot column labeled     TIME     from csv file and color it cyan
# subplot(2 Rows, 3 Columns, Sixth subplot)
ax60 = f2.add_subplot(2,3,6)
ax60.plot(data[["Acceleration"]], label = 'Curve 6', color = "k")
ax60.plot(data[["TIME"]], label = 'Curve 12', color = "c", marker = 's', markevery = 10)
# added line marker square


#=============================================================================
#  Figure Plot options
#=============================================================================

#-----------------------------------------------------------------------------
#  Figure 1 options
#-----------------------------------------------------------------------------

#switch to figure one for editing
plt.figure(1)

# Plot one options
ax1.legend(loc='upper right', fontsize='large')
ax1.set_title('Title for plot one ')
ax1.set_xlabel('X axes label')
ax1.set_ylabel('Y axes label')
ax1.grid(True)
ax1.set_xlim([0,200])
ax1.set_ylim([0,20])

# Plot two options
ax2.legend(loc='upper left', fontsize='large')
ax2.set_title('Title for plot two ')
ax2.set_xlabel('X axes label')
ax2.set_ylabel('Y axes label')
ax2.grid(True)
ax2.set_xlim([0,200])
ax2.set_ylim([0,20])

# Plot three options
ax3.legend(loc='upper center', fontsize='large')
ax3.set_title('Title for plot three ')
ax3.set_xlabel('X axes label')
ax3.set_ylabel('Y axes label')
ax3.grid(True)
ax3.set_xlim([0,200])
ax3.set_ylim([0,20])

# Plot four options
ax4.legend(loc='lower right', fontsize='large')
ax4.set_title('Title for plot four')
ax4.set_xlabel('X axes label')
ax4.set_ylabel('Y axes label')
ax4.grid(True)
ax4.set_xlim([0,200])
```

```
ax4.set_ylim([0,20])

# Plot five options
ax5.legend(loc='lower left', fontsize='large')
ax5.set_title('Title for plot five ')
ax5.set_xlabel('X axes label')
ax5.set_ylabel('Y axes label')
ax5.grid(True)
ax5.set_xlim([0,200])
ax5.set_ylim([0,20])

# Plot six options
ax6.legend(loc='lower center', fontsize='large')
ax6.set_title('Title for plot six')
ax6.set_xlabel('X axes label')
ax6.set_ylabel('Y axes label')
ax6.grid(True)
ax6.set_xlim([0,200])
ax6.set_ylim([0,20])

#------------------------------------------------------------------------------
#  Figure 2 options
#------------------------------------------------------------------------------

#switch to figure two for editing
plt.figure(2)

# Plot one options
ax10.legend(loc='upper right', fontsize='large')
ax10.set_title('Title for plot one ')
ax10.set_xlabel('X axes label')
ax10.set_ylabel('Y axes label')
ax10.grid(True)
ax10.set_xlim([0,200])
ax10.set_ylim([-20,20])

# Plot two options
ax20.legend(loc='upper left', fontsize='large')
ax20.set_title('Title for plot two ')
ax20.set_xlabel('X axes label')
ax20.set_ylabel('Y axes label')
ax20.grid(True)
ax20.set_xlim([0,200])
ax20.set_ylim([-20,20])

# Plot three options
ax30.legend(loc='upper center', fontsize='large')
ax30.set_title('Title for plot three ')
ax30.set_xlabel('X axes label')
ax30.set_ylabel('Y axes label')
ax30.grid(True)
ax30.set_xlim([0,200])
ax30.set_ylim([-20,20])

# Plot four options
ax40.legend(loc='lower right', fontsize='large')
ax40.set_title('Title for plot four')
ax40.set_xlabel('X axes label')
ax40.set_ylabel('Y axes label')
ax40.grid(True)
ax40.set_xlim([0,200])
```

```
ax40.set_ylim([-20,20])

# Plot five options
ax50.legend(loc='lower left', fontsize='large')
ax50.set_title('Title for plot five ')
ax50.set_xlabel('X axes label')
ax50.set_ylabel('Y axes label')
ax50.grid(True)
ax50.set_xlim([0,200])
ax50.set_ylim([-20,20])

# Plot six options
ax60.legend(loc='lower center', fontsize='large')
ax60.set_title('Title for plot six')
ax60.set_xlabel('X axes label')
ax60.set_ylabel('Y axes label')
ax60.grid(True)
ax60.set_xlim([0,200])
ax60.set_ylim([-20,20])


#============================================================================
#  User chosen file location Save PDF
#============================================================================

savefilename = askdirectory()# user selected file path
pdf = PdfPages(f'{savefilename}/longplot.pdf')
# using formatted string literals ("f-strings")to place the variable into the string

# save both figures into one pdf file
pdf.savefig(1)
pdf.savefig(2)

pdf.close()


#============================================================================
#  Show plot
#============================================================================

# manually set the subplot spacing when there are multiple plots
#plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace =None, hspace=None )

# Automaticlly adds space between plots
plt.tight_layout()

plt.show()
```

Прочитайте Несколько участков онлайн: https://riptutorial.com/ru/matplotlib/topic/3279/несколько-участков

глава 11: Объекты фигур и осей

Examples

Создание фигуры

Фигура содержит все элементы сюжета. Основным способом создания фигуры в `matplotlib` является использование `pyplot` .

```
import matplotlib.pyplot as plt
fig = plt.figure()
```

Вы можете указать номер, который вы можете использовать для доступа к ранее созданной фигуре. Если номер не указан, идентификатор последней созданной фигуры будет увеличиваться и использоваться вместо этого; цифры индексируются начиная с 1, а не 0.

```
import matplotlib.pyplot as plt
fig = plt.figure()
fig == plt.figure(1)  # True
```

Вместо числа цифры могут также идентифицироваться строкой. Если вы используете интерактивный бэкэнд, это также задает заголовок окна.

```
import matplotlib.pyplot as plt
fig = plt.figure('image')
```

Чтобы выбрать использование фигуры

```
plt.figure(fig.number) # or
plt.figure(1)
```

Создание осей

Существует два основных способа создания осей в matplotlib: использование pyplot или использование объектно-ориентированного API.

Использование pyplot:

```
import matplotlib.pyplot as plt

ax = plt.subplot(3, 2, 1)  # 3 rows, 2 columns, the first subplot
```

Использование объектно-ориентированного API:

```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(3, 2, 1)
```

Функция удобства `plt.subplots()` может использоваться для создания фигуры и набора подзадач в одной команде:

```
import matplotlib.pyplot as plt

fig, (ax1, ax2) = plt.subplots(ncols=2, nrows=1)  # 1 row, 2 columns
```

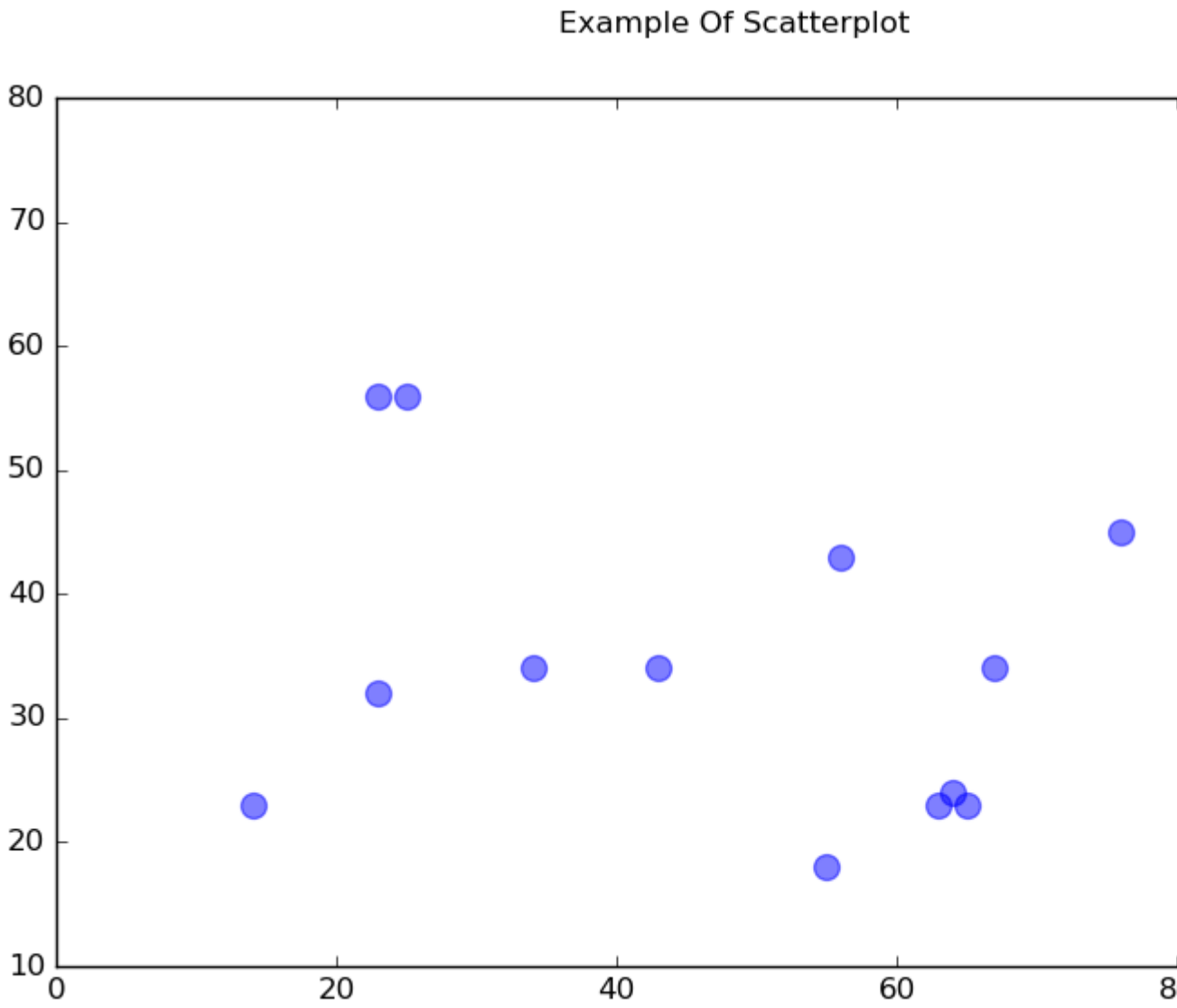
Прочитайте Объекты фигур и осей онлайн: https://riptutorial.com/ru/matplotlib/topic/2307/объекты-фигур-и-осей

глава 12: Основные сюжеты

Examples

Границы рассеяния

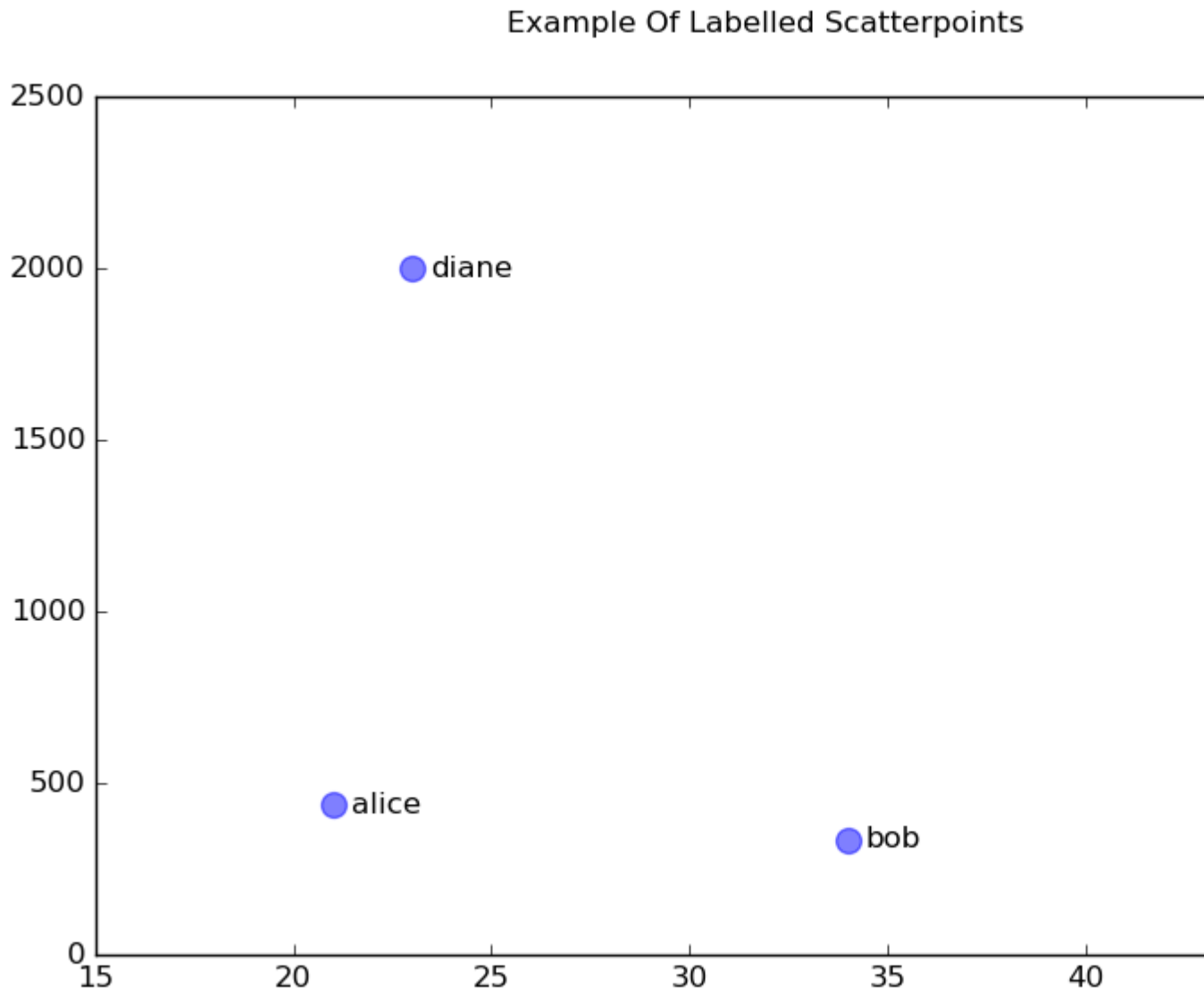
Простая диаграмма рассеяния




```
import matplotlib.pyplot as plt

# Data
x = [43,76,34,63,56,82,87,55,64,87,95,23,14,65,67,25,23,85]
y = [34,45,34,23,43,76,26,18,24,74,23,56,23,23,34,56,32,23]

fig, ax = plt.subplots(1, figsize=(10, 6))
fig.suptitle('Example Of Scatterplot')
```

```
# Create the Scatter Plot
ax.scatter(x, y,
            color="blue",    # Color of the dots
            s=100,           # Size of the dots
            alpha=0.5,       # Alpha/transparency of the dots (1 is opaque, 0 is transparent)
            linewidths=1)    # Size of edge around the dots

# Show the plot
plt.show()
```

Scatterplot с отмеченными точками

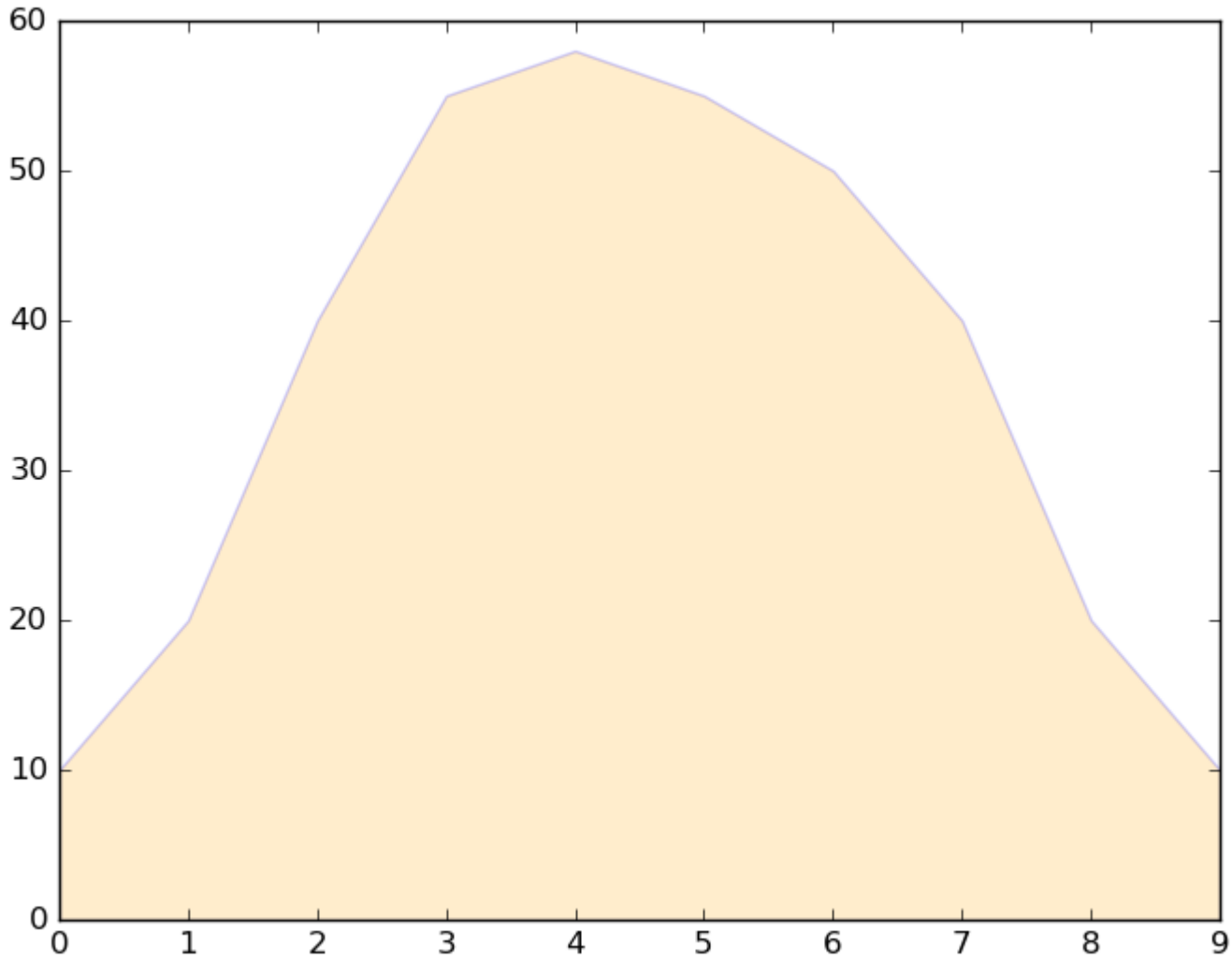



```
import matplotlib.pyplot as plt

# Data
x = [21, 34, 44, 23]
y = [435, 334, 656, 1999]
labels = ["alice", "bob", "charlie", "diane"]
```

```
# Create the figure and axes objects
fig, ax = plt.subplots(1, figsize=(10, 6))
fig.suptitle('Example Of Labelled Scatterpoints')

# Plot the scatter points
ax.scatter(x, y,
           color="blue",  # Color of the dots
           s=100,         # Size of the dots
           alpha=0.5,     # Alpha of the dots
           linewidths=1)  # Size of edge around the dots

# Add the participant names as text labels for each point
for x_pos, y_pos, label in zip(x, y, labels):
    ax.annotate(label,             # The label for this point
                xy=(x_pos, y_pos), # Position of the corresponding point
                xytext=(7, 0),     # Offset text by 7 points to the right
                textcoords='offset points', # tell it to use offset points
                ha='left',         # Horizontally aligned to the left
                va='center')       # Vertical alignment is centered

# Show the plot
plt.show()
```

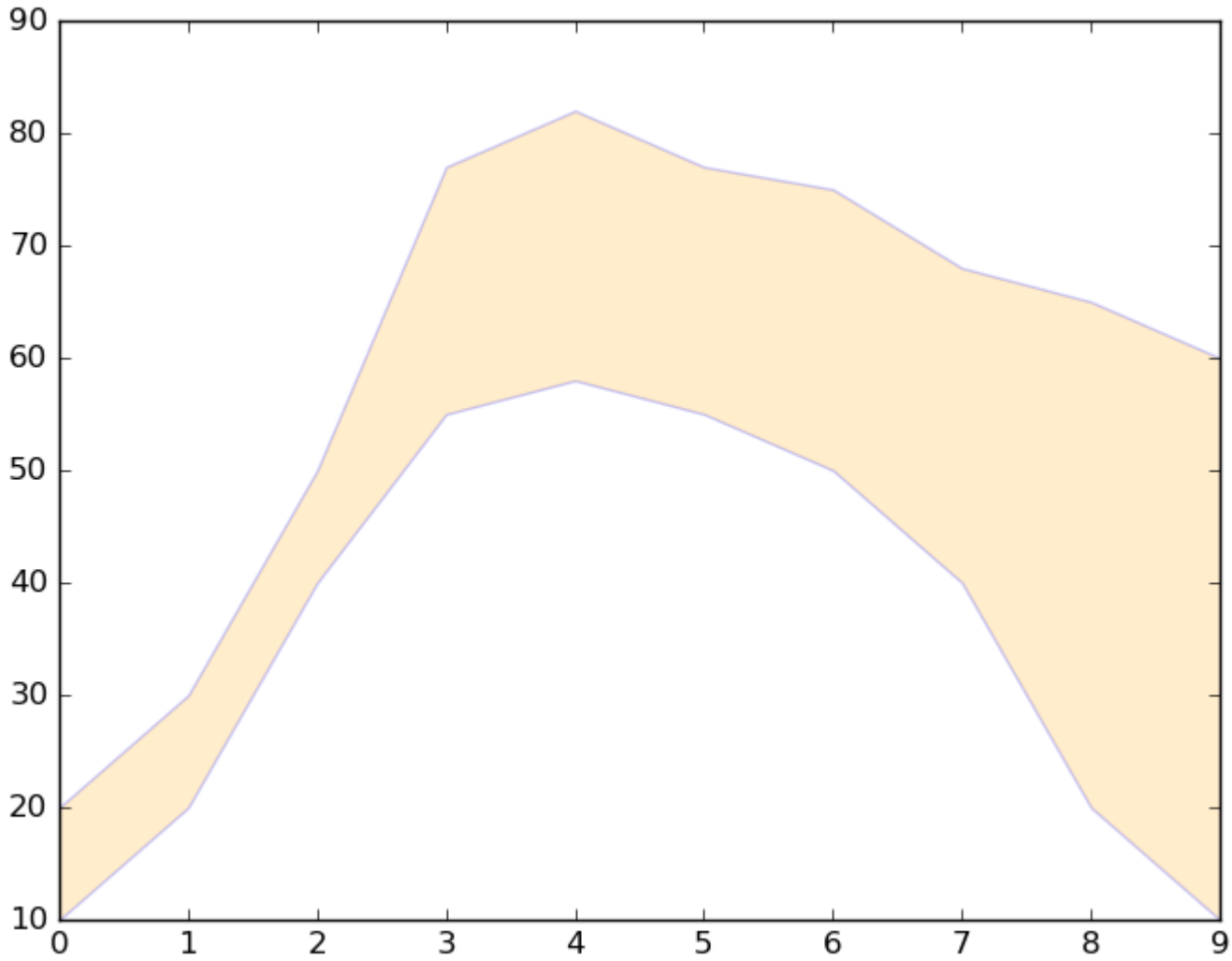
Затененные участки

Заштрихованная область ниже линии




```
import matplotlib.pyplot as plt

# Data
x =  [0,1,2,3,4,5,6,7,8,9]
y1 = [10,20,40,55,58,55,50,40,20,10]

# Shade the area between y1 and line y=0
plt.fill_between(x, y1, 0,
                 facecolor="orange", # The fill color
                 color='blue',       # The outline color
                 alpha=0.2)          # Transparency of the fill

# Show the plot
plt.show()
```

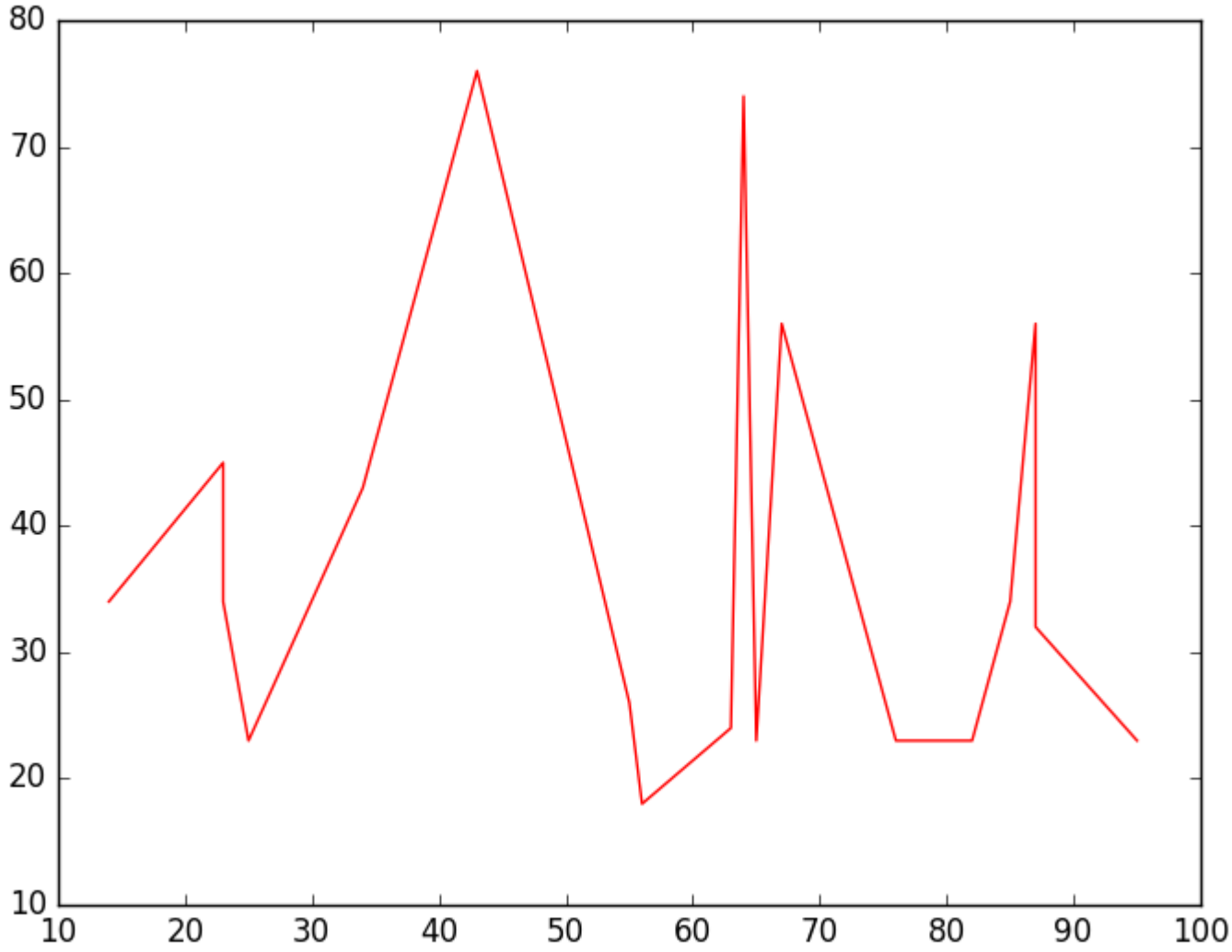
Заштрихованная область между двумя линиями




```
import matplotlib.pyplot as plt

# Data
x =  [0,1,2,3,4,5,6,7,8,9]
y1 = [10,20,40,55,58,55,50,40,20,10]
y2 = [20,30,50,77,82,77,75,68,65,60]

# Shade the area between y1 and y2
plt.fill_between(x, y1, y2,
                 facecolor="orange", # The fill color
                 color='blue',       # The outline color
                 alpha=0.2)          # Transparency of the fill

# Show the plot
plt.show()
```

Линейные графики

Простой график




```
import matplotlib.pyplot as plt

# Data
x = [14,23,23,25,34,43,55,56,63,64,65,67,76,82,85,87,87,95]
y = [34,45,34,23,43,76,26,18,24,74,23,56,23,23,34,56,32,23]

# Create the plot
plt.plot(x, y, 'r-')
# r- is a style code meaning red solid line

# Show the plot
plt.show()
```

Заметим, что в общем случае `y` не является функцией `x` а также что значения в `x` не нужно сортировать. Вот как выглядит линейный график с несортированными x-значениями:

```
# shuffle the elements in x
np.random.shuffle(x)
plt.plot(x, y, 'r-')
plt.show()
```

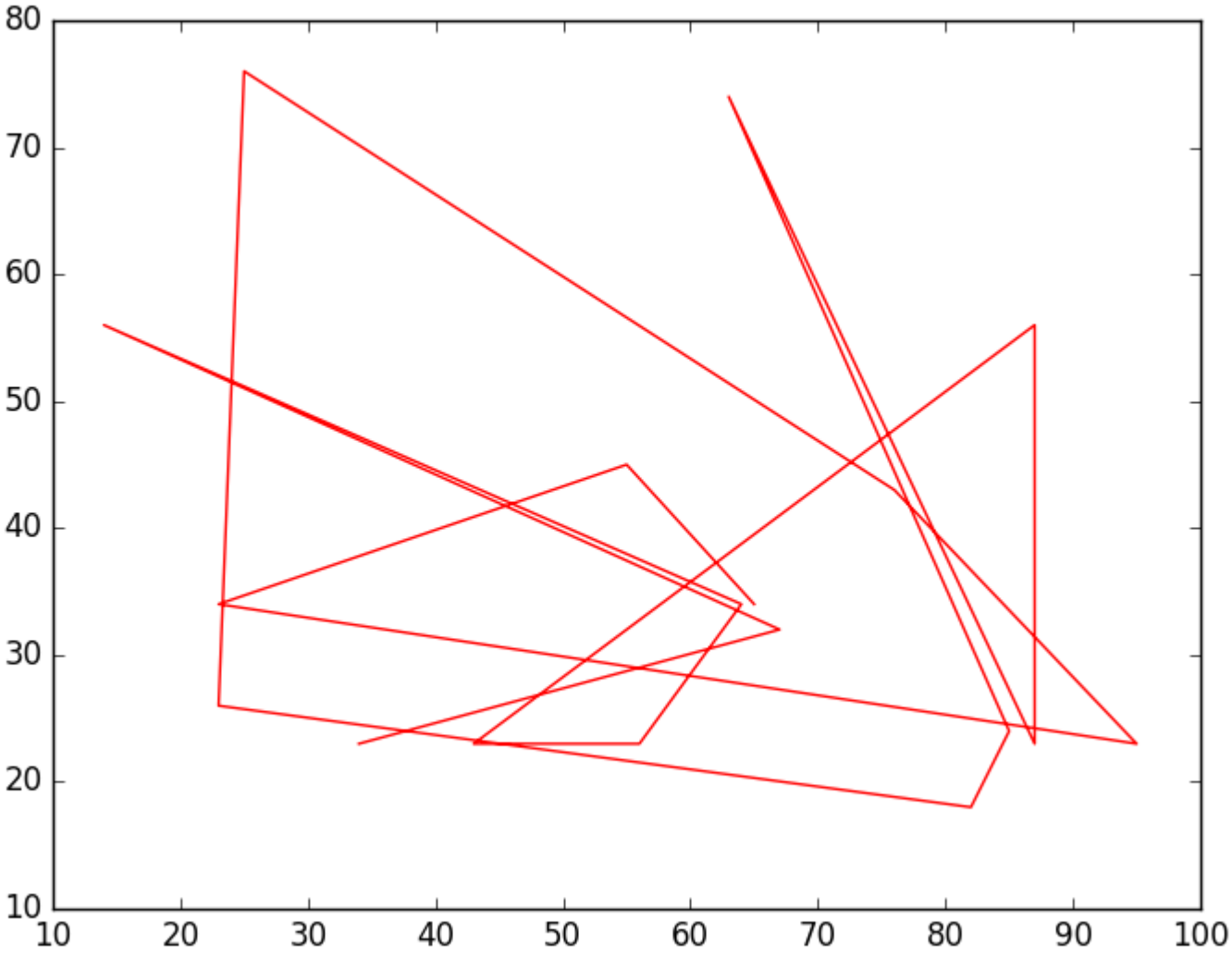
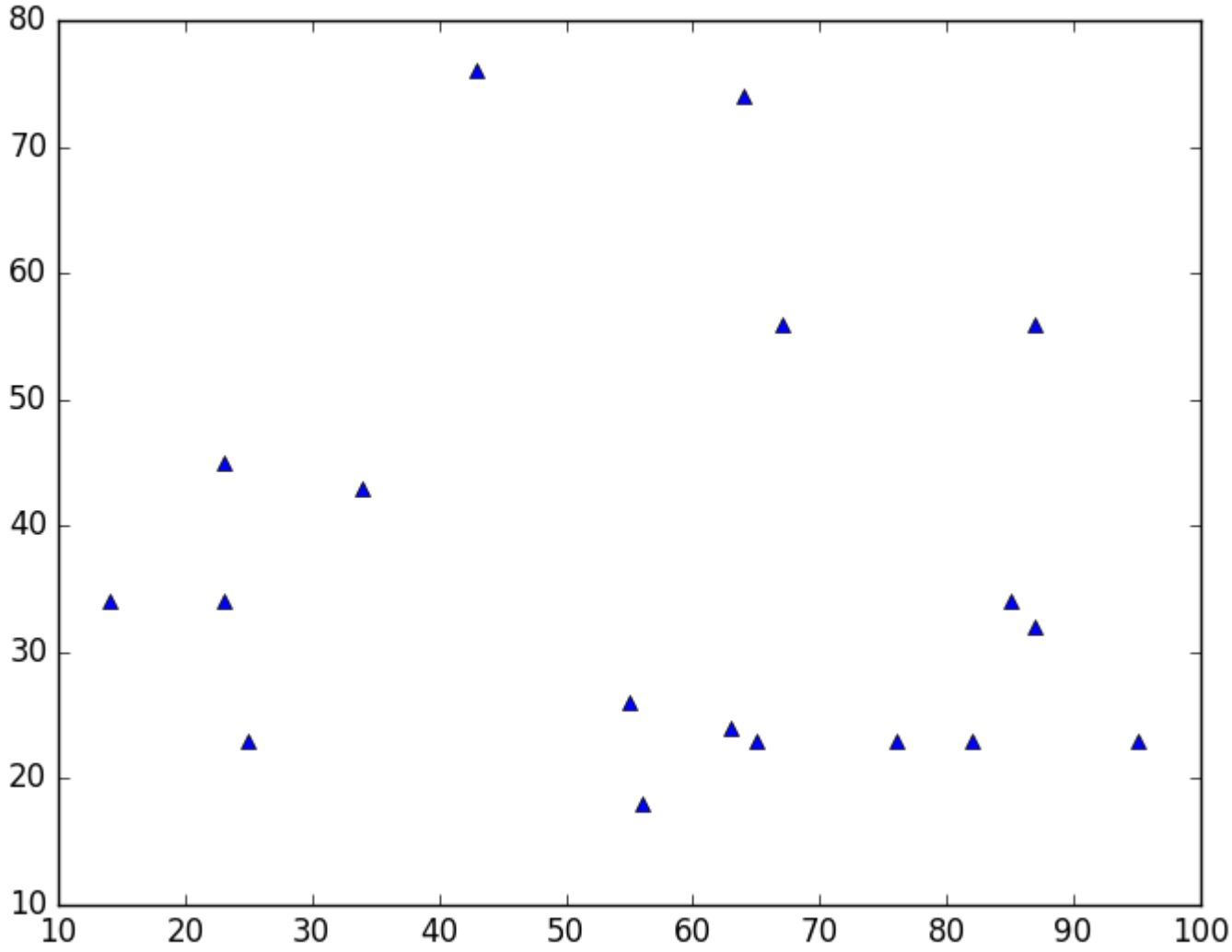


График данных

Это похоже на график рассеяния , но вместо этого использует функцию `plot()` .
Единственное различие в коде здесь - аргумент стиля.

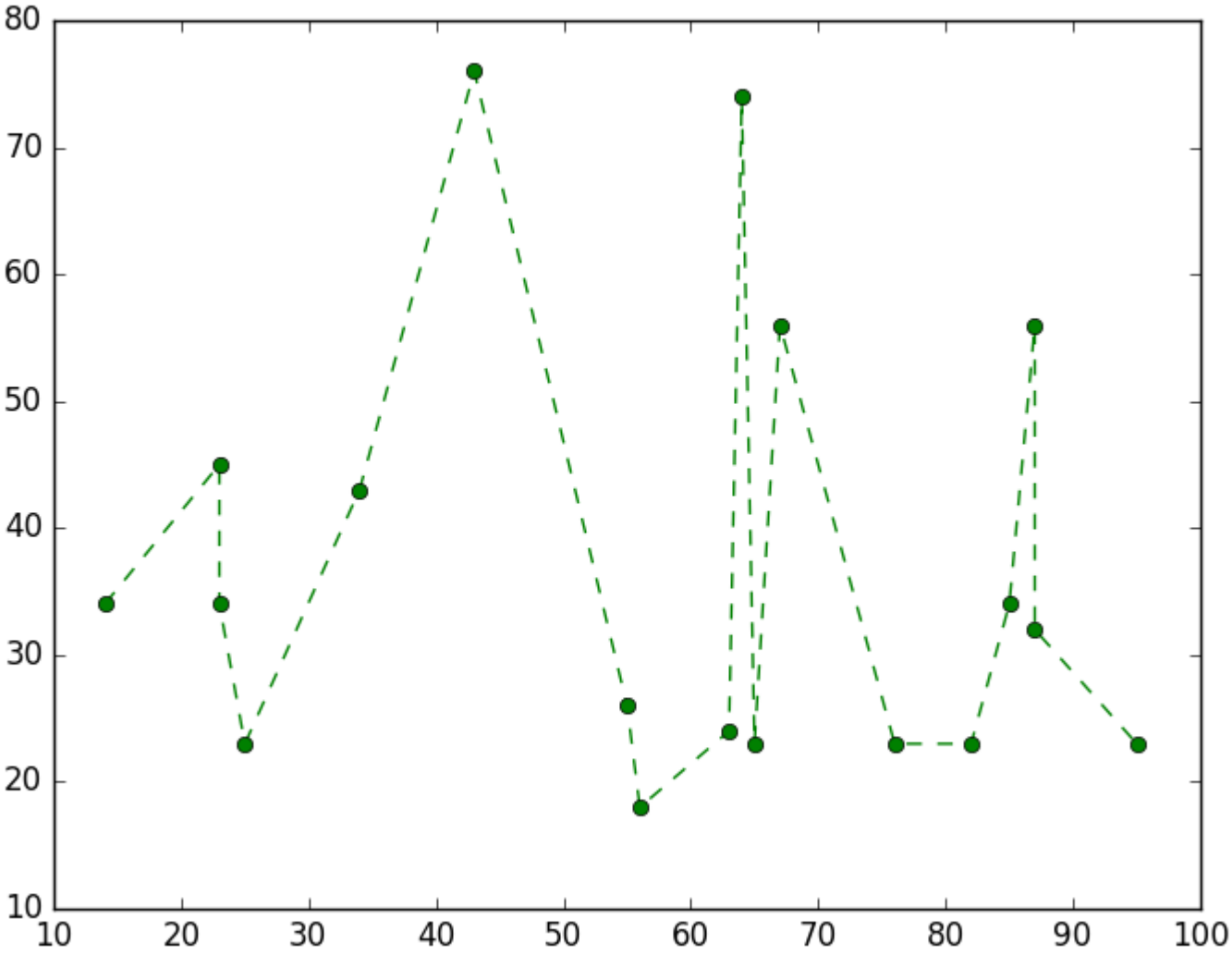
```
plt.plot(x, y, 'b^')
# Create blue up-facing triangles
```



Данные и линия

Аргумент стиля может принимать символы как для маркеров, так и для стиля линии:

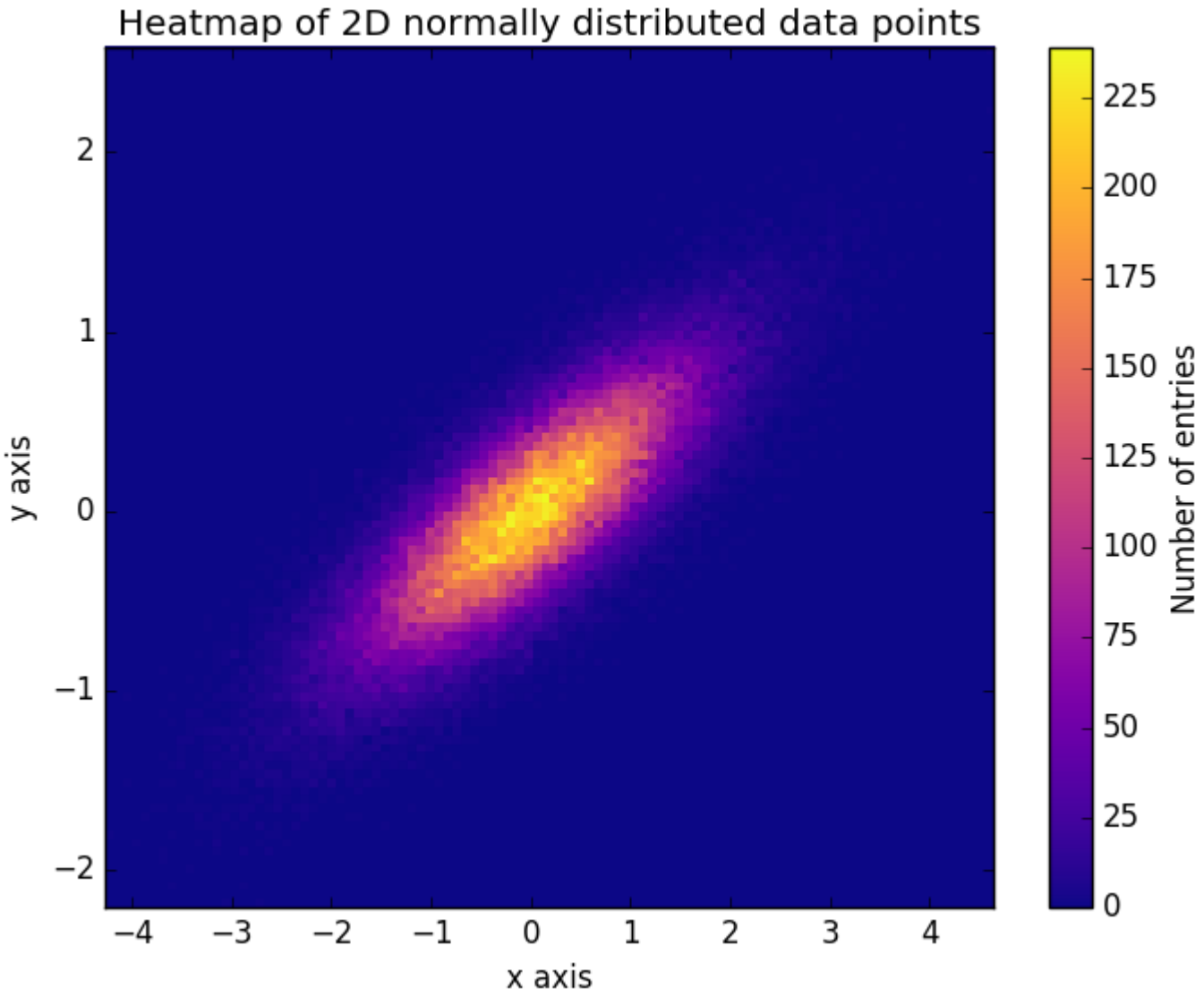
```
plt.plot(x, y, 'go--')
# green circles and dashed line
```



Тепловая карта

Тепловые карты полезны для визуализации скалярных функций двух переменных. Они обеспечивают «плоское» изображение двумерных гистограмм (представляющих, например, плотность определенной области).

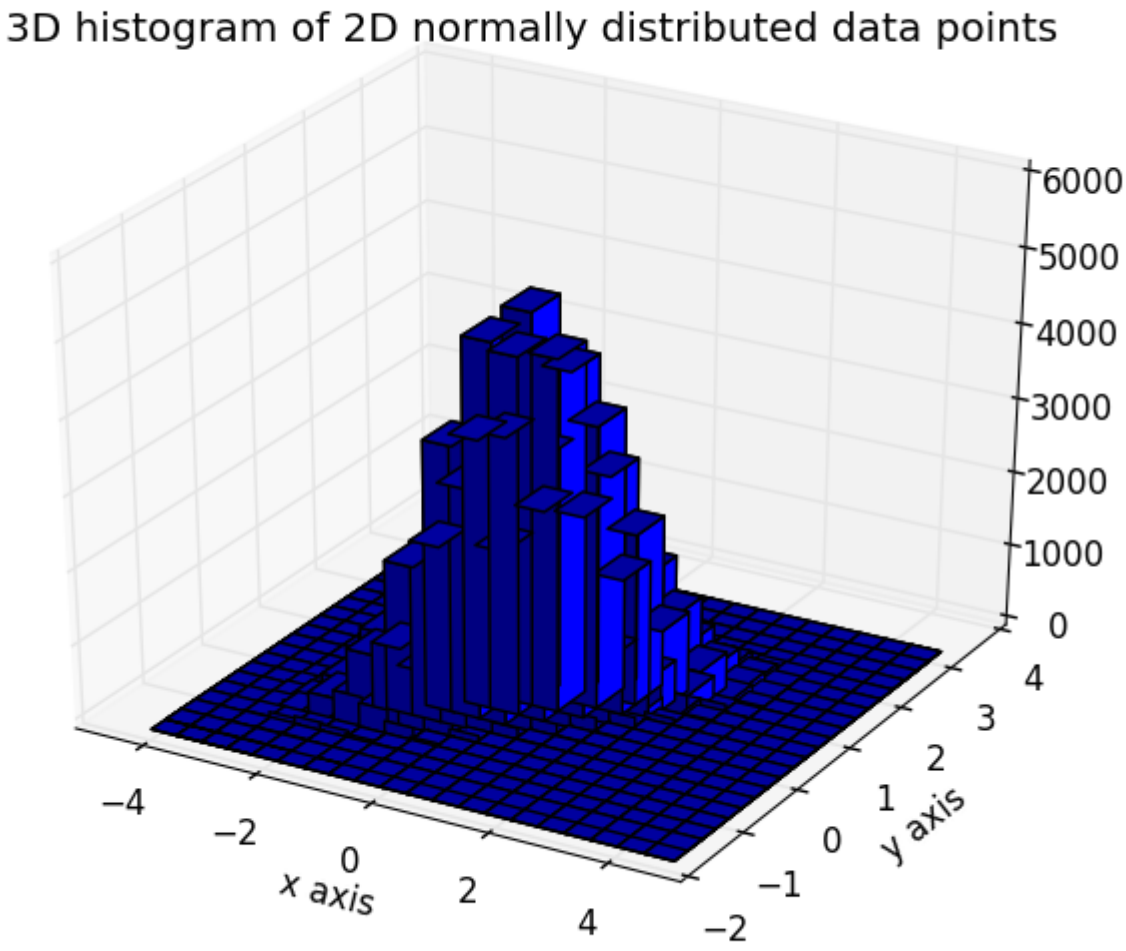
Следующий исходный код иллюстрирует тепломассы с использованием двумерных нормально распределенных чисел с центром в 0 в обоих направлениях (означает `[0.0, 0.0]`) и а с заданной ковариационной матрицей. Данные генерируются с помощью функции numpy numpy.random.multivariate_normal ; она затем подается в `hist2d` функции pyplot matplotlib.pyplot.hist2d .




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

# Define numbers of generated data points and bins per axis.
N_numbers = 100000
N_bins = 100

# set random seed
np.random.seed(0)

# Generate 2D normally distributed numbers.
x, y = np.random.multivariate_normal(
        mean=[0.0, 0.0],      # mean
        cov=[[1.0, 0.4],
             [0.4, 0.25]],    # covariance matrix
        size=N_numbers
        ).T                   # transpose to get columns


# Construct 2D histogram from data using the 'plasma' colormap
plt.hist2d(x, y, bins=N_bins, normed=False, cmap='plasma')
```

```
# Plot a colorbar with label.
cb = plt.colorbar()
cb.set_label('Number of entries')

# Add title and labels to plot.
plt.title('Heatmap of 2D normally distributed data points')
plt.xlabel('x axis')
plt.ylabel('y axis')

# Show the plot.
plt.show()
```

Вот те же данные, что и 3D-гистограмма (здесь мы используем только 20 бит для повышения эффективности). Код основан на этой демо-версии matplotlib .




```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

# Define numbers of generated data points and bins per axis.
```

```
N_numbers = 100000
N_bins = 20

# set random seed
np.random.seed(0)

# Generate 2D normally distributed numbers.
x, y = np.random.multivariate_normal(
        mean=[0.0, 0.0],      # mean
        cov=[[1.0, 0.4],
             [0.4, 0.25]],    # covariance matrix
        size=N_numbers
        ).T                   # transpose to get columns

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
hist, xedges, yedges = np.histogram2d(x, y, bins=N_bins)

# Add title and labels to plot.
plt.title('3D histogram of 2D normally distributed data points')
plt.xlabel('x axis')
plt.ylabel('y axis')

# Construct arrays for the anchor positions of the bars.
# Note: np.meshgrid gives arrays in (ny, nx) so we use 'F' to flatten xpos,
# ypos in column-major order. For numpy >= 1.7, we could instead call meshgrid
# with indexing='ij'.
xpos, ypos = np.meshgrid(xedges[:-1] + 0.25, yedges[:-1] + 0.25)
xpos = xpos.flatten('F')
ypos = ypos.flatten('F')
zpos = np.zeros_like(xpos)

# Construct arrays with the dimensions for the 16 bars.
dx = 0.5 * np.ones_like(zpos)
dy = dx.copy()
dz = hist.flatten()

ax.bar3d(xpos, ypos, zpos, dx, dy, dz, color='b', zsort='average')

# Show the plot.
plt.show()
```

Прочитайте Основные сюжеты онлайн: https://riptutorial.com/ru/matplotlib/topic/3266/основные-сюжеты

глава 13: присущи рефлективный, вербальный

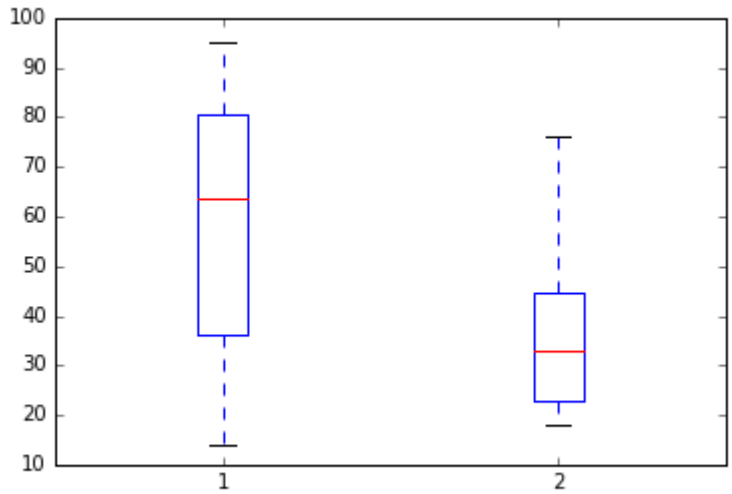
Examples

Основные боксы

Boxplots - это описательные диаграммы, которые помогают сравнивать распределение разных рядов данных. Они являются *описательными,* потому что они показывают меры (например, *медианные*), которые не предполагают базового распределения вероятности.

Самый простой пример boxplot в matplotlib можно достичь, просто передав данные в виде списка списков:

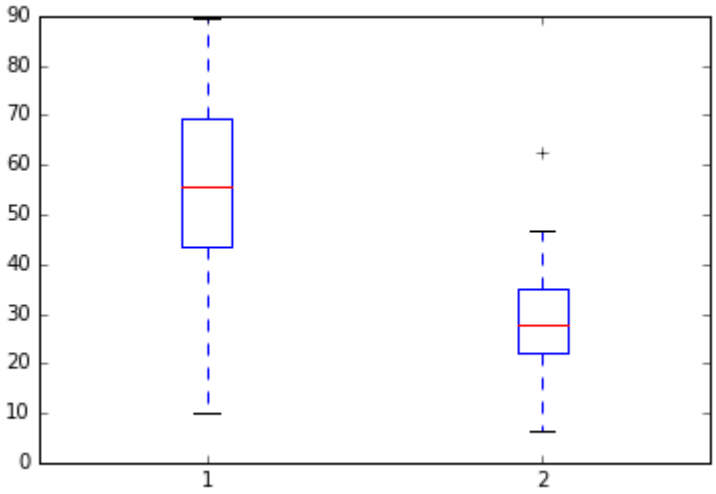
```
import matplotlib as plt

dataline1 = [43,76,34,63,56,82,87,55,64,87,95,23,14,65,67,25,23,85]
dataline2 = [34,45,34,23,43,76,26,18,24,74,23,56,23,23,34,56,32,23]
data = [ dataline1, dataline2 ]

plt.boxplot( data )
```



Однако обычной практикой является использование массивов `numpy` качестве параметров для графиков, поскольку они часто являются результатом предыдущих вычислений. Это можно сделать следующим образом:

```
import numpy as np
import matplotlib as plt

np.random.seed(123)
```

```
dataline1 = np.random.normal( loc=50, scale=20, size=18 )
dataline2 = np.random.normal( loc=30, scale=10, size=18 )
data = np.stack( [ dataline1, dataline2 ], axis=1 )

plt.boxplot( data )
```



Прочитайте присущи рефлективный, вербальный онлайн:
https://riptutorial.com/ru/matplotlib/topic/6086/присущи-рефлективный--вербальный

глава 14: присущи рефлективный, вербальный

Examples

Функция Boxplot

Matplotlib имеет собственную реализацию boxplot . Соответствующие аспекты этой функции заключаются в том, что по умолчанию в квадратике показана медиана (процентиль 50%) с красной линией. Коробка представляет Q1 и Q3 (процентили 25 и 75), а усы дают представление о диапазоне данных (возможно, при Q1 - 1,5 *IQR; Q3 + 1,5* IQR, IQR - межквартильный диапазон, но в этом нет подтверждения). Также обратите внимание, что образцы, выходящие за этот диапазон, показаны как маркеры (они называются летчиками).

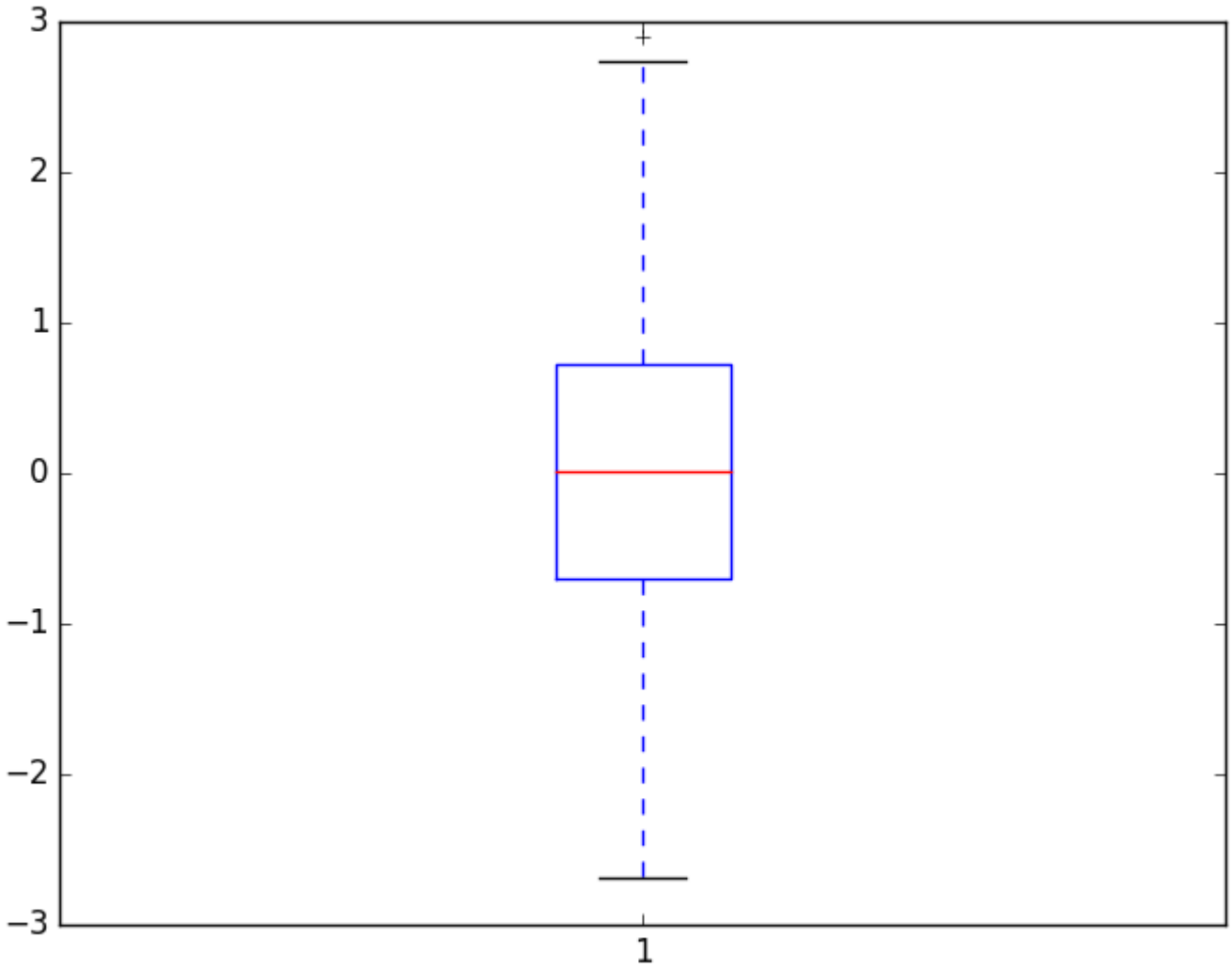
> **ПРИМЕЧАНИЕ.** Не все реализации *boxplot* соответствуют тем же правилам. Возможно, наиболее распространенная диаграмма boxplot использует бакенбарды для представления минимума и максимума (что делает листовки несуществующими). Также обратите внимание , что этот участок иногда называют *коробчатыми и усами-графиком* и *схемой коробчатых и-усов.*

Следующий рецепт показывает некоторые из вещей, которые вы можете сделать с текущей реализацией matplotlib boxplot:

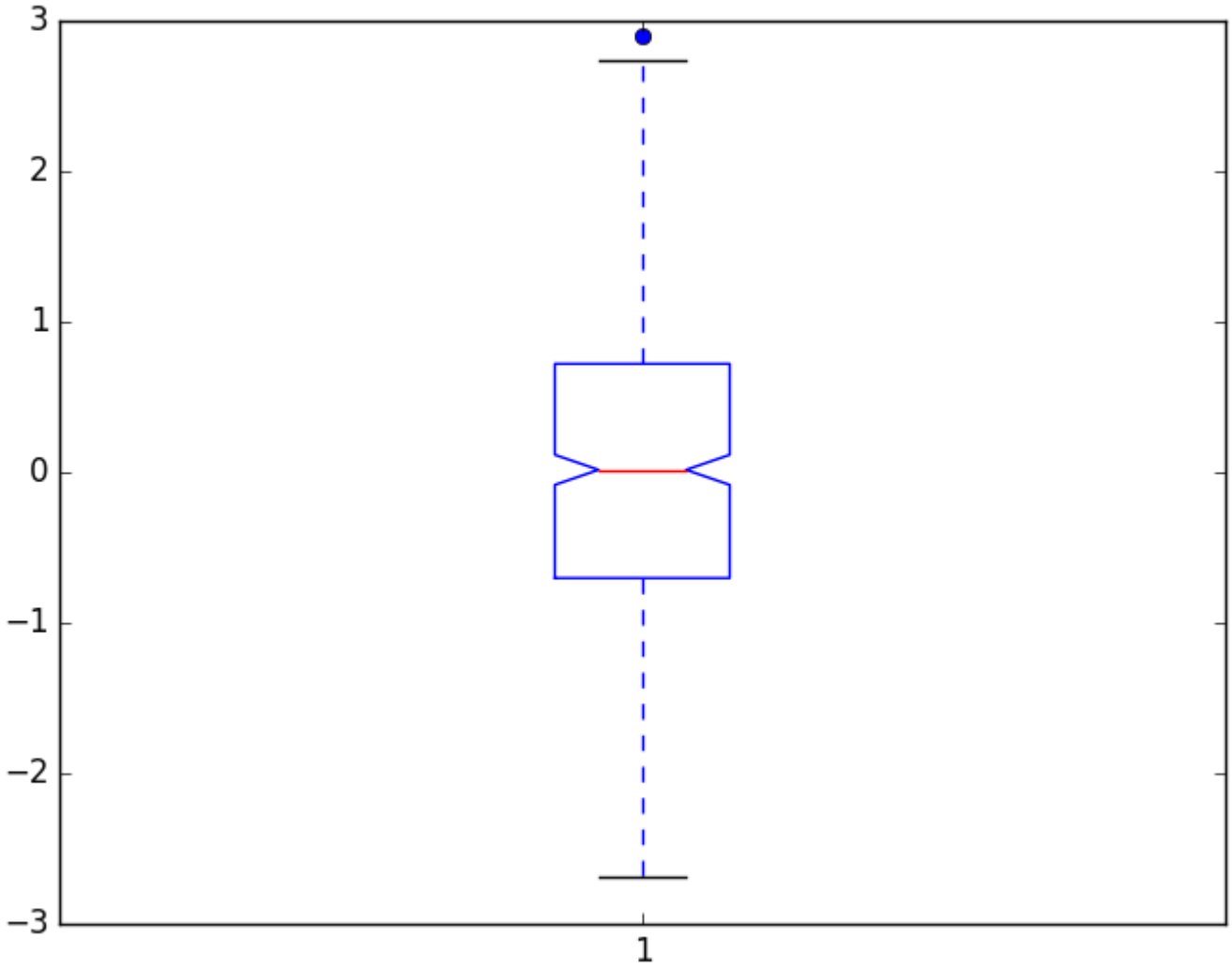
```
import matplotlib.pyplot as plt
import numpy as np

X1 = np.random.normal(0, 1, 500)
X2 = np.random.normal(0.3, 1, 500)

# The most simple boxplot
plt.boxplot(X1)
plt.show()

# Changing some of its features
plt.boxplot(X1, notch=True, sym="o") # Use sym="" to shown no fliers; also showfliers=False
plt.show()

# Showing multiple boxplots on the same window
plt.boxplot((X1, X2), notch=True, sym="o", labels=["Set 1", "Set 2"])
plt.show()

# Hidding features of the boxplot
plt.boxplot(X2, notch=False, showfliers=False, showbox=False, showcaps=False, positions=[4],
labels=["Set 2"])
plt.show()

# Advanced customization of the boxplot
line_props = dict(color="r", alpha=0.3)
bbox_props = dict(color="g", alpha=0.9, linestyle="dashdot")
```

```
flier_props = dict(marker="o", markersize=17)
plt.boxplot(X1, notch=True, whiskerprops=line_props, boxprops=bbox_props,
flierprops=flier_props)
plt.show()
```

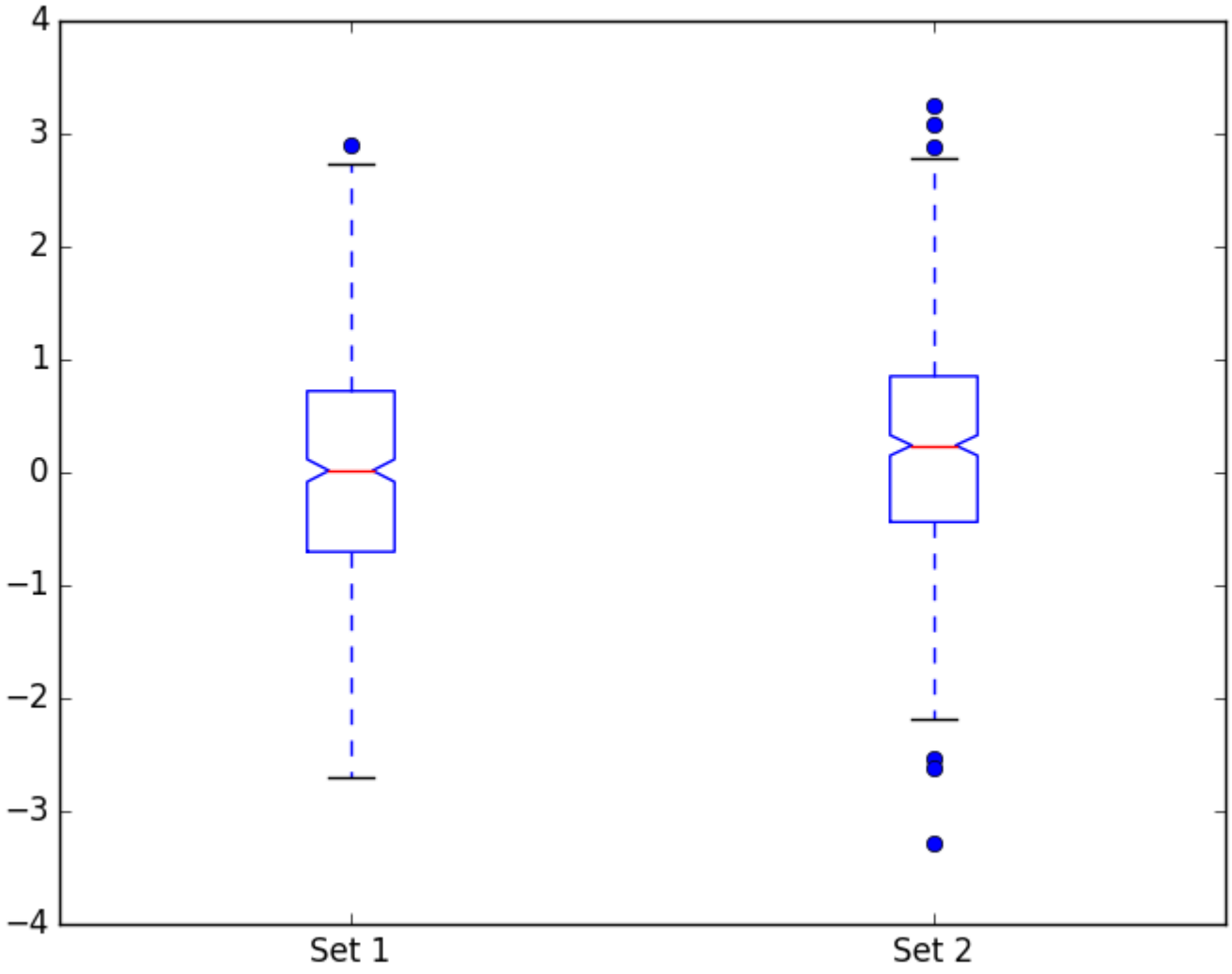
Это приводит к следующим графикам:



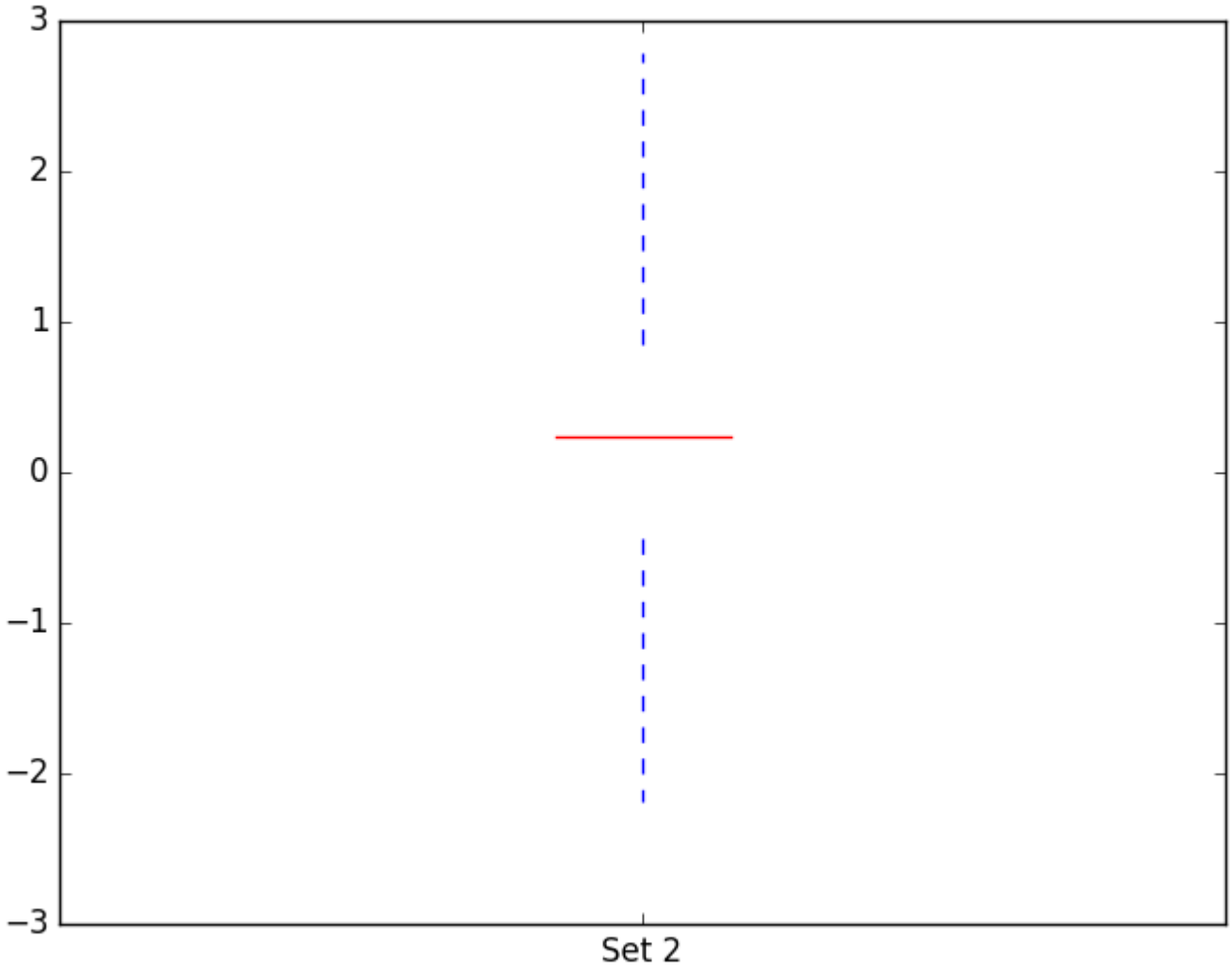
1. *По умолчанию matplotlib boxplot*



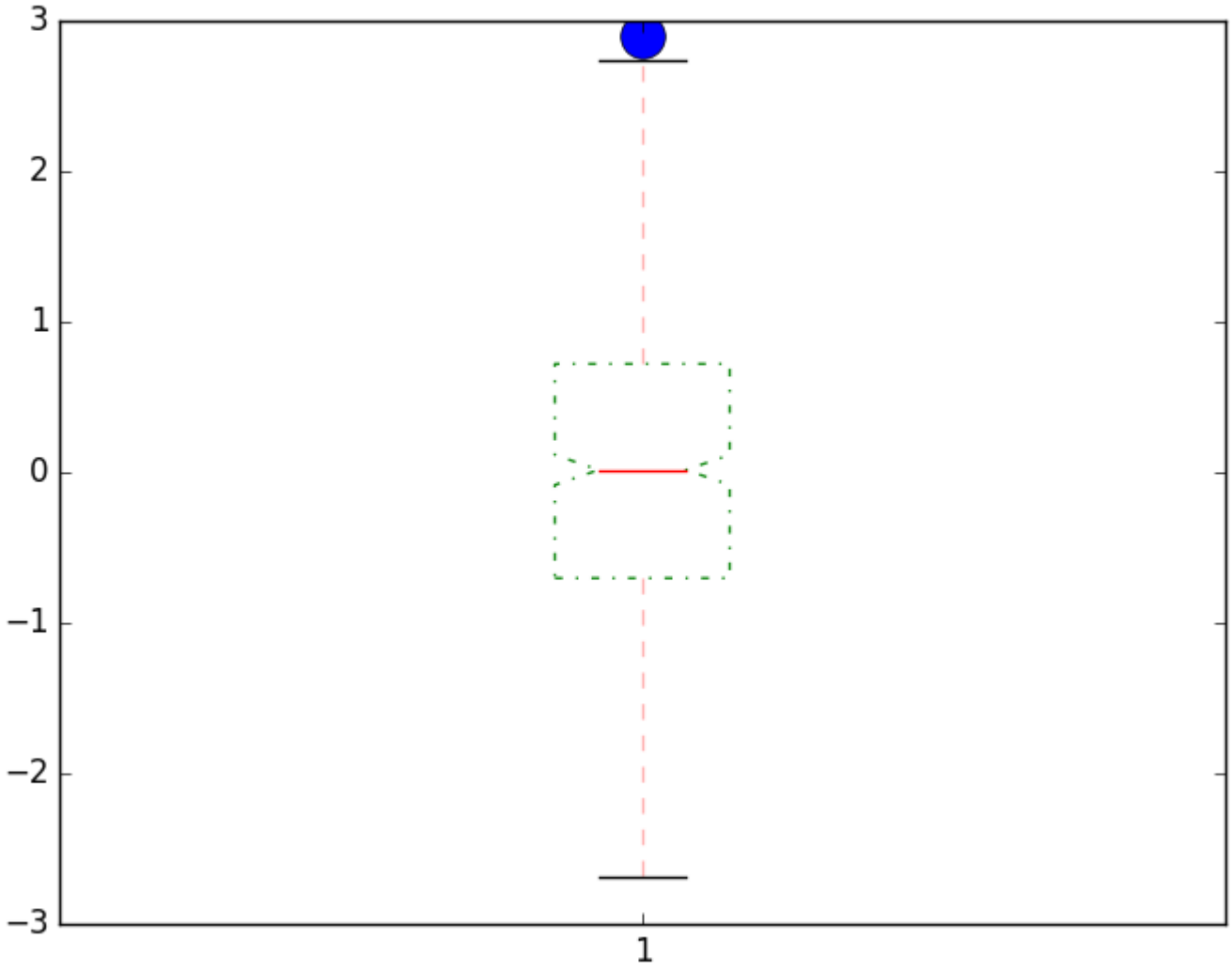
2. *Изменение некоторых функций boxplot с использованием аргументов функции*



3. *Несколько ящиков в одном окне графика*



4. *Хранение некоторых функций boxplot*



5. *Расширенная настройка ящика с помощью реквизита*

Если вы намереваетесь выполнить некоторую расширенную настройку вашего boxplot, вы должны знать, что создаваемые вами реквизиты (например):

```
line_props = dict(color="r", alpha=0.3)
bbox_props = dict(color="g", alpha=0.9, linestyle="dashdot")
flier_props = dict(marker="o", markersize=17)
plt.boxplot(X1, notch=True, whiskerprops=line_props, boxprops=bbox_props,
flierprops=flier_props)
plt.show()
```

... ссылаются главным образом (если не все) на объекты Line2D . Это означает, что доступны только аргументы, доступные в этом классе. Вы заметите наличие таких ключевых слов, как `whiskerprops` , `boxprops` , `flierprops` и `capprops` . Это те элементы, которые вам нужны для создания реквизита для дальнейшей настройки.

ПРИМЕЧАНИЕ. Дальнейшая настройка boxplot с использованием этой

реализации может оказаться сложной задачей. В некоторых случаях использование других элементов matplotlib, таких как патчи для создания собственной коробки, может быть выгодным (например, значительные изменения в элементе box).

Прочитайте присущи рефлективный, вербальный онлайн:
https://riptutorial.com/ru/matplotlib/topic/6368/присущи-рефлективный--вербальный

глава 15: Системы координат

замечания

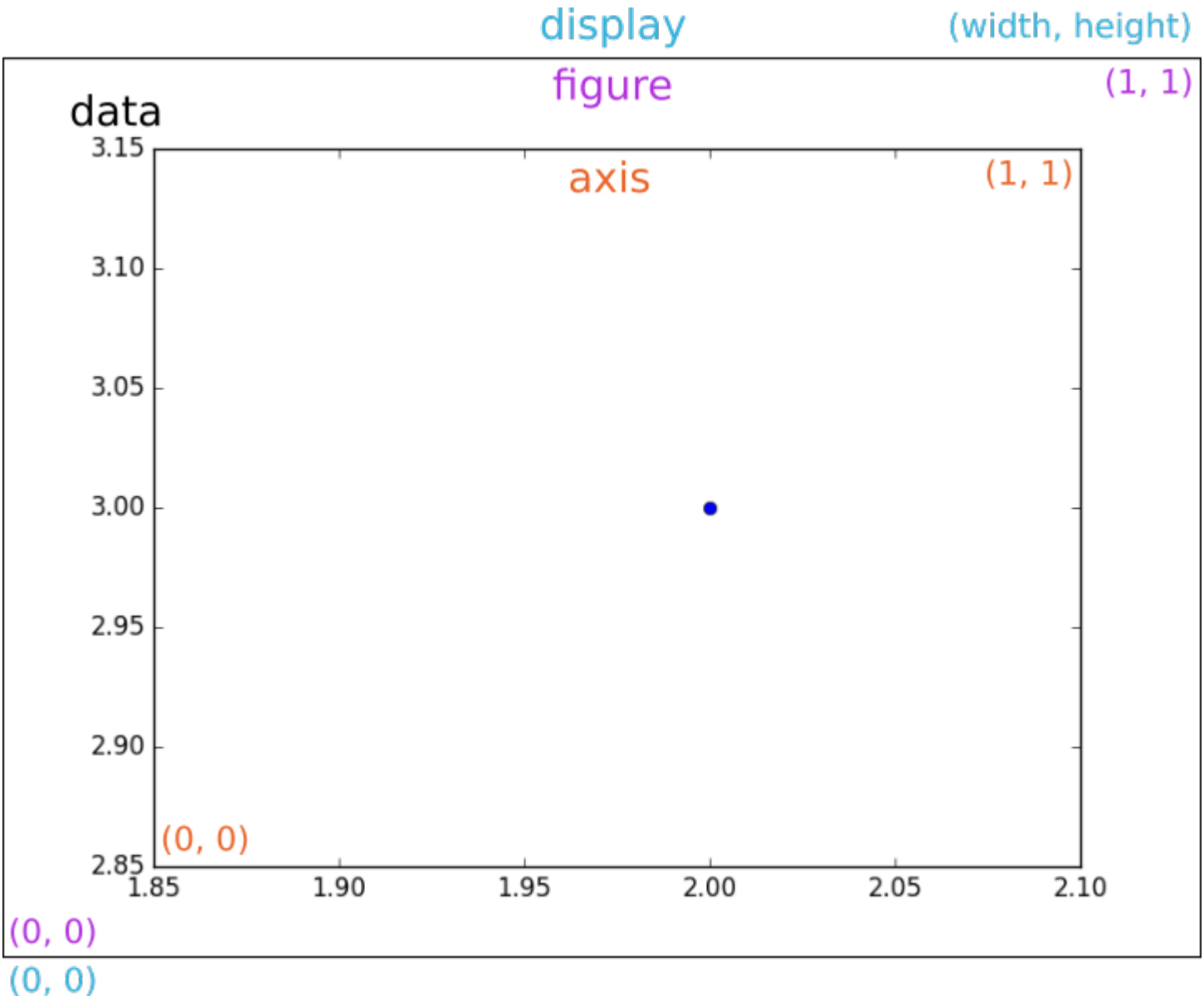
Matplotlib имеет четыре различные системы координат, которые могут быть использованы для облегчения позиционирования другого объекта, например текста. Каждая система имеет соответствующий объект преобразования, который преобразует координаты из этой системы в так называемую систему координат дисплея.

Система координат данных - это система, определяемая данными о соответствующих осях. Это полезно при попытке позиционировать некоторый объект относительно данных, нанесенных на график. Диапазон задается свойствами `xlim` и `ylim` `Axes` . Его соответствующим объектом преобразования является `ax.transData` .

Система координат осей - это система, привязанная к объекту `Axes` . Точки (0, 0) и (1, 1) определяют нижние левые и верхние правые углы осей. По существу, это полезно при позиционировании относительно осей, например, в верхней части графика. Соответствующим объектом преобразования является `ax.transAxes` .

Система координат рисунка аналогична системе координат осей, за исключением того, что она привязана к `Figure` . Точки (0, 0) и (1, 1) представляют нижние и верхние правые углы фигуры. Это полезно при попытке позиционировать что-то относительно всего изображения. Соответствующим объектом преобразования является `fig.transFigure` .

Отображение системы координат - это система изображения, заданного в пикселях. Точки (0, 0) и (ширина, высота) - это нижние и верхние правые пиксели изображения или дисплея. Его можно использовать для позиционирования абсолютно. Поскольку объекты преобразования преобразуют координаты в эту систему координат, система отображения не имеет связанного с ней объекта преобразования. Однако при необходимости можно использовать `None` или `matplotlib.transforms.IdentityTransform()` .



Более подробная информация доступна здесь .

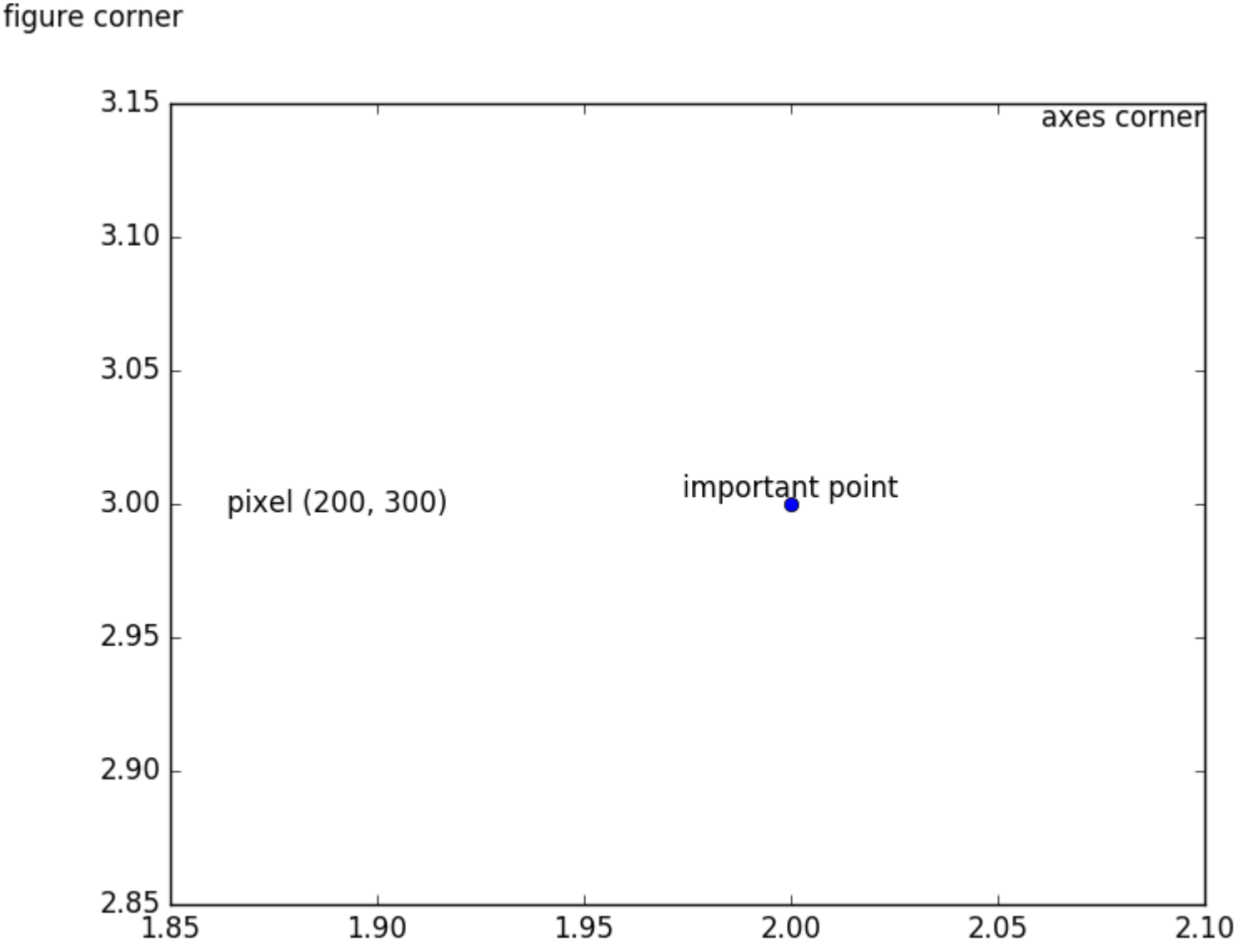
Examples

Системы координат и текст

Системы координат Matplotlib очень удобны, когда вы пытаетесь комментировать детали, которые вы делаете. Иногда вы хотели бы разместить текст относительно ваших данных, например, при попытке маркировать определенную точку. Иногда вам, возможно, хотелось бы добавить текст поверх фигуры. Этого можно легко добиться, выбрав соответствующую систему координат, передав объект `transform` параметр `transform` в вызове `text()` .

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

ax.plot([2.], [3.], 'bo')
```

```
plt.text(  # position text relative to data
    2., 3., 'important point',  # x, y, text,
    ha='center', va='bottom',   # text alignment,
    transform=ax.transData      # coordinate system transformation
)
plt.text(  # position text relative to Axes
    1.0, 1.0, 'axes corner',
    ha='right', va='top',
    transform=ax.transAxes
)
plt.text(  # position text relative to Figure
    0.0, 1.0, 'figure corner',
    ha='left', va='top',
    transform=fig.transFigure
)
plt.text(  # position text absolutely at specific pixel on image
    200, 300, 'pixel (200, 300)',
    ha='center', va='center',
    transform=None
)

plt.show()
```



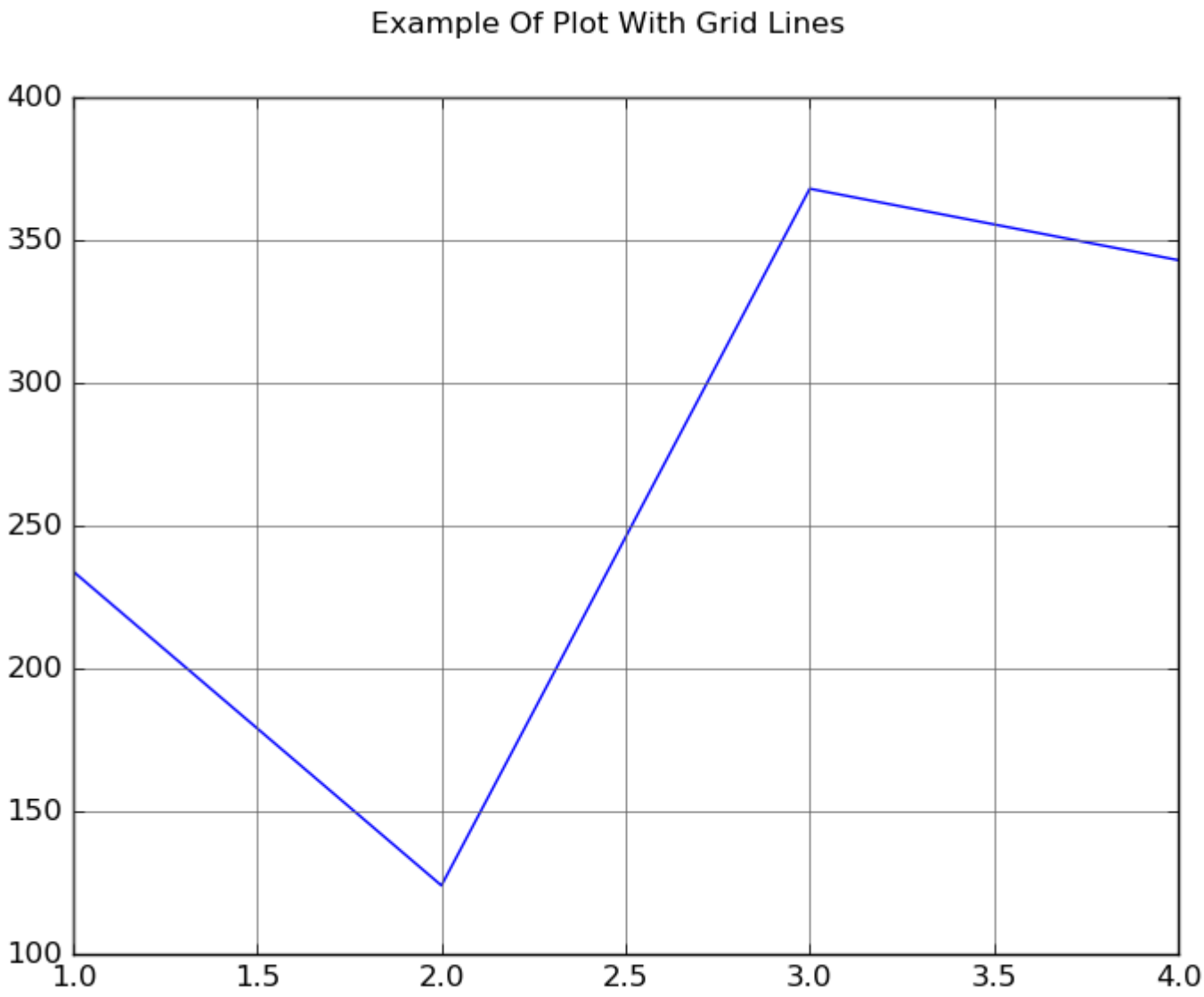
Прочитайте Системы координат онлайн: https://riptutorial.com/ru/matplotlib/topic/4566/системы-координат

глава 16: Строки сетки и метки

Examples

Участок с сеткой

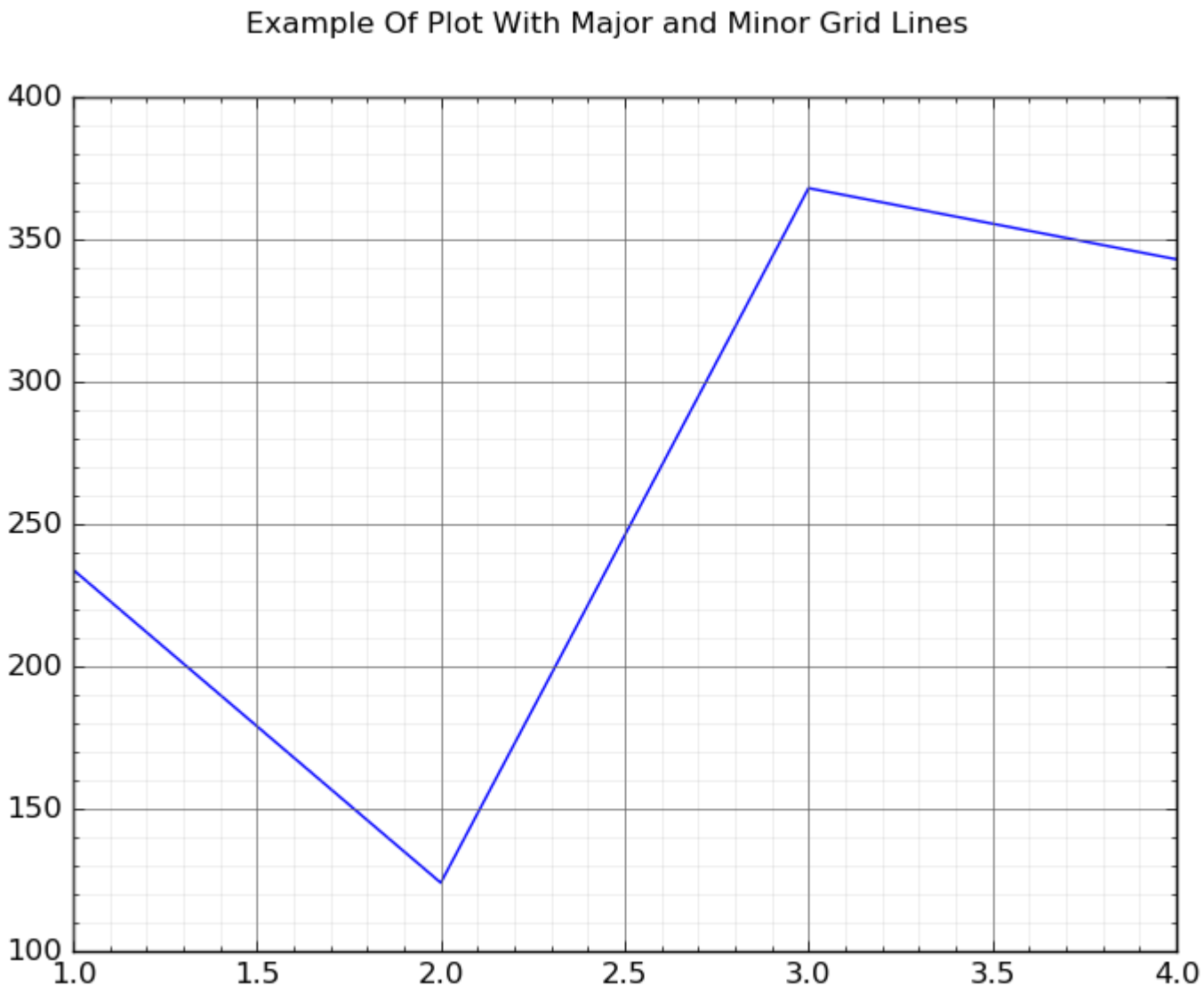
Участок с линиями сетки




```
import matplotlib.pyplot as plt

# The Data
x = [1, 2, 3, 4]
y = [234, 124,368, 343]

# Create the figure and axes objects
```

```
fig, ax = plt.subplots(1, figsize=(8, 6))
fig.suptitle('Example Of Plot With Grid Lines')

# Plot the data
ax.plot(x,y)

# Show the grid lines as dark grey lines
plt.grid(b=True, which='major', color='#666666', linestyle='-')

plt.show()
```

Участок с основными и малыми сетками




```
import matplotlib.pyplot as plt

# The Data
x = [1, 2, 3, 4]
y = [234, 124,368, 343]
```

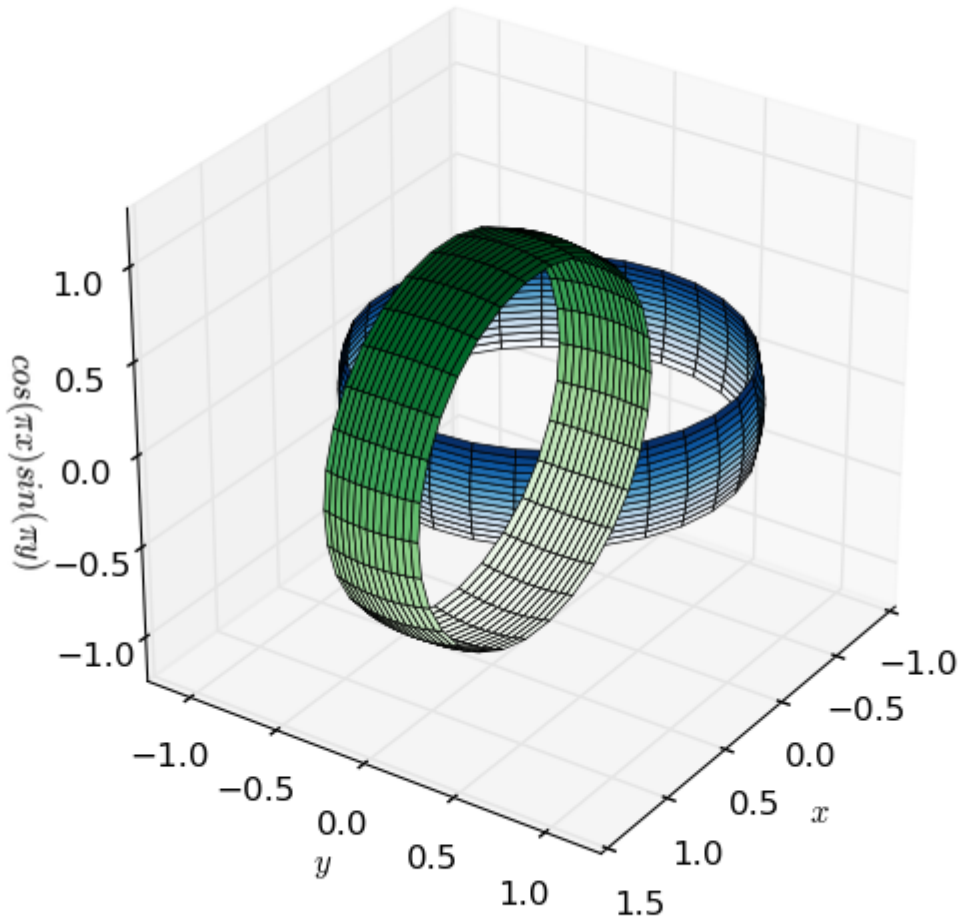
```
# Create the figure and axes objects
fig, ax = plt.subplots(1, figsize=(8, 6))
fig.suptitle('Example Of Plot With Major and Minor Grid Lines')

# Plot the data
ax.plot(x,y)

# Show the major grid lines with dark grey lines
plt.grid(b=True, which='major', color='#666666', linestyle='-')

# Show the minor grid lines with very faint and almost transparent grey lines
plt.minorticks_on()
plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)

plt.show()
```

Прочитайте Строки сетки и метки онлайн: https://riptutorial.com/ru/matplotlib/topic/4029/строки-сетки-и-метки

глава 17: Трехмерные графики

замечания

Трехмерное построение в matplotlib исторически было немного клочья, поскольку движок рендеринга по своей сути 2d. Тот факт, что 3D-установки визуализируются путем построения одного 2-го блока после другого, подразумевает, что часто возникают проблемы, связанные с кажущейся глубиной объектов. Ядро проблемы состоит в том, что два несвязанных объекта могут либо полностью отстать, либо полностью друг над другом, что приводит к артефактам, как показано на рисунке ниже двух взаимосвязанных колец (щелкните для анимированного gif):

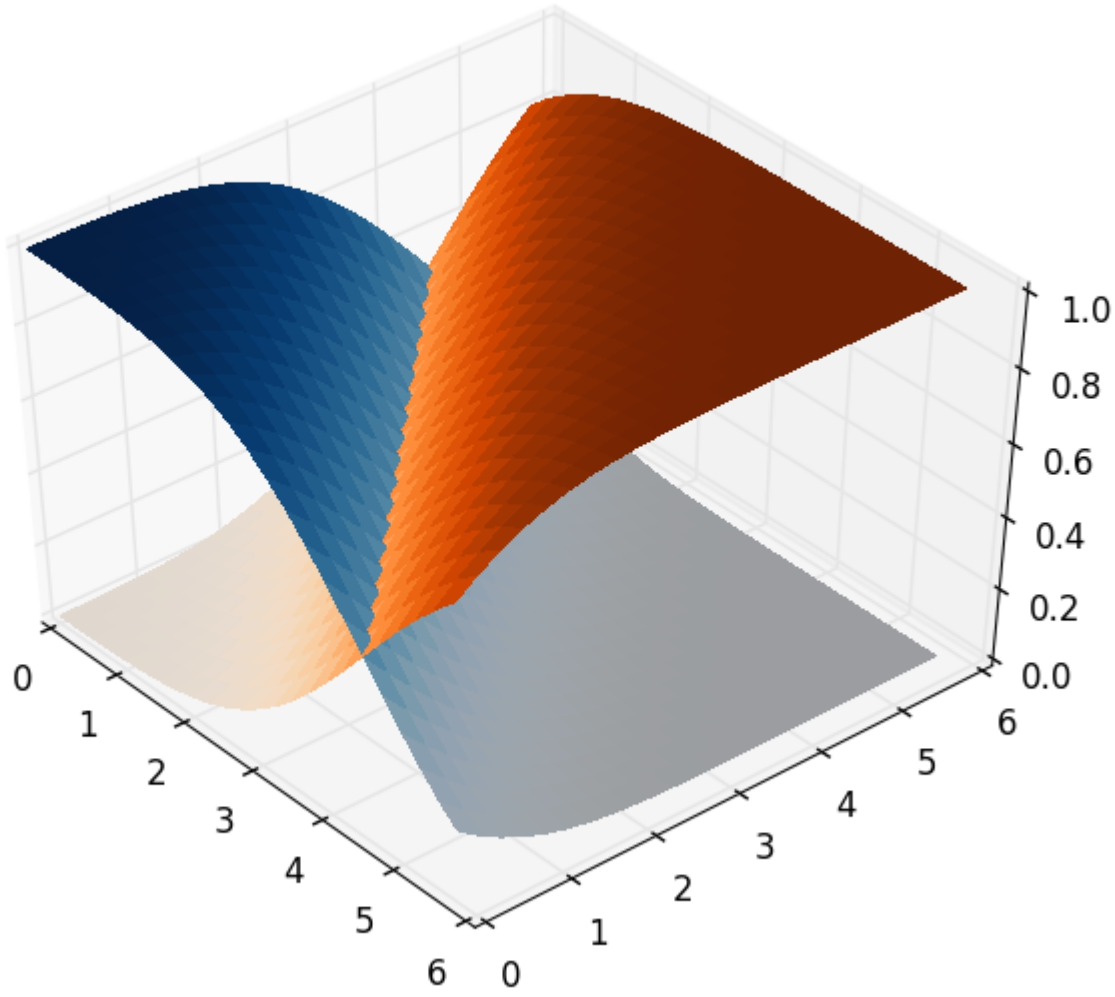


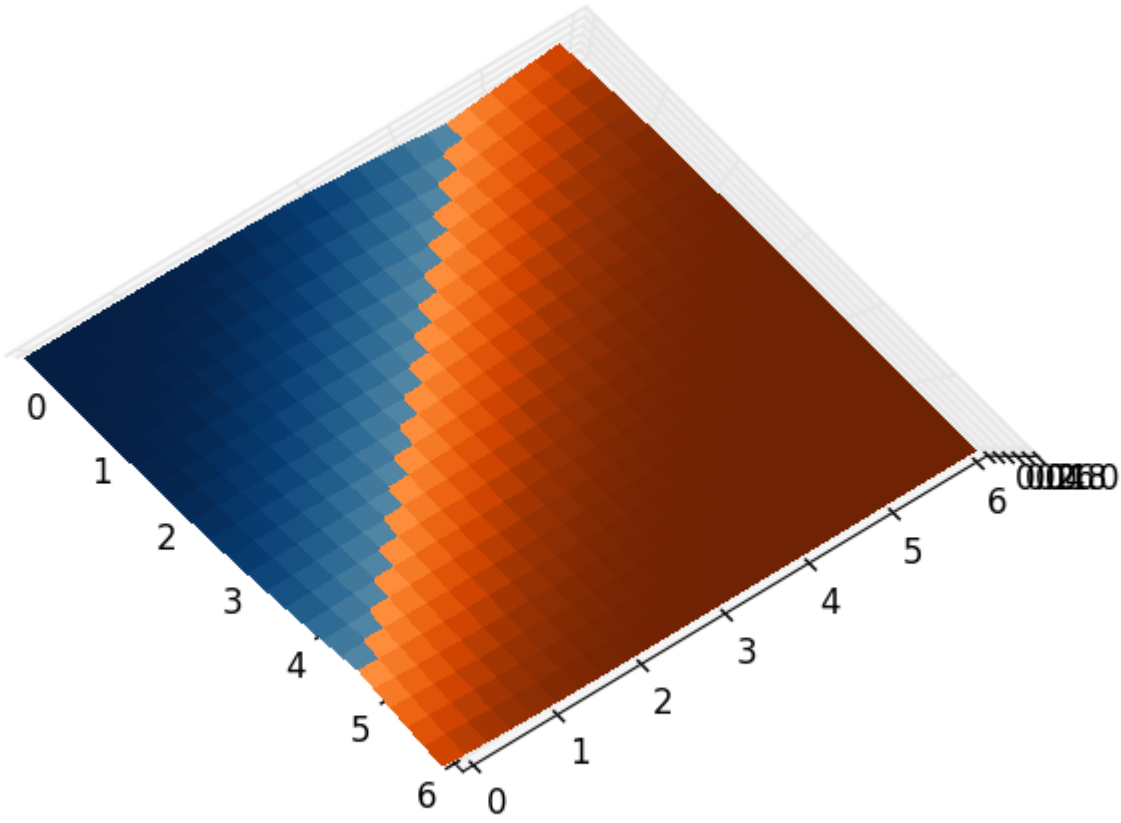
Однако это может быть исправлено. Этот артефакт существует только при построении нескольких поверхностей на одном и том же участке - каждый из них отображается как плоская двумерная фигура с одним параметром, определяющим расстояние обзора. Вы

заметите, что одна сложная поверхность не испытывает такой же проблемы.

Способ исправить это - объединить объекты сюжета вместе с использованием прозрачных мостов:

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import erf

fig = plt.figure()
ax = fig.gca(projection='3d')

X = np.arange(0, 6, 0.25)
Y = np.arange(0, 6, 0.25)
X, Y = np.meshgrid(X, Y)

Z1 = np.empty_like(X)
Z2 = np.empty_like(X)
C1 = np.empty_like(X, dtype=object)
C2 = np.empty_like(X, dtype=object)

for i in range(len(X)):
  for j in range(len(X[0])):
    z1 = 0.5*(erf((X[i,j]+Y[i,j]-4.5)*0.5)+1)
    z2 = 0.5*(erf((-X[i,j]-Y[i,j]+4.5)*0.5)+1)
    Z1[i,j] = z1
    Z2[i,j] = z2

    # If you want to grab a colour from a matplotlib cmap function,
    # you need to give it a number between 0 and 1. z1 and z2 are
    # already in this range, so it just works as is.
    C1[i,j] = plt.get_cmap("Oranges")(z1)
    C2[i,j] = plt.get_cmap("Blues")(z2)


# Create a transparent bridge region
X_bridge = np.vstack([X[-1,:],X[-1,:]])
Y_bridge = np.vstack([Y[-1,:],Y[-1,:]])
Z_bridge = np.vstack([Z1[-1,:],Z2[-1,:]])
color_bridge = np.empty_like(Z_bridge, dtype=object)

color_bridge.fill((1,1,1,0)) # RGBA colour, onlt the last component matters - it represents
the alpha / opacity.

# Join the two surfaces flipping one of them (using also the bridge)
X_full = np.vstack([X, X_bridge, np.flipud(X)])
Y_full = np.vstack([Y, Y_bridge, np.flipud(Y)])
Z_full = np.vstack([Z1, Z_bridge, np.flipud(Z2)])
color_full = np.vstack([C1, color_bridge, np.flipud(C2)])

surf_full = ax.plot_surface(X_full, Y_full, Z_full, rstride=1, cstride=1,
                            facecolors=color_full, linewidth=0,
                            antialiased=False)

plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
0
1
2
3
4
5
6



Examples

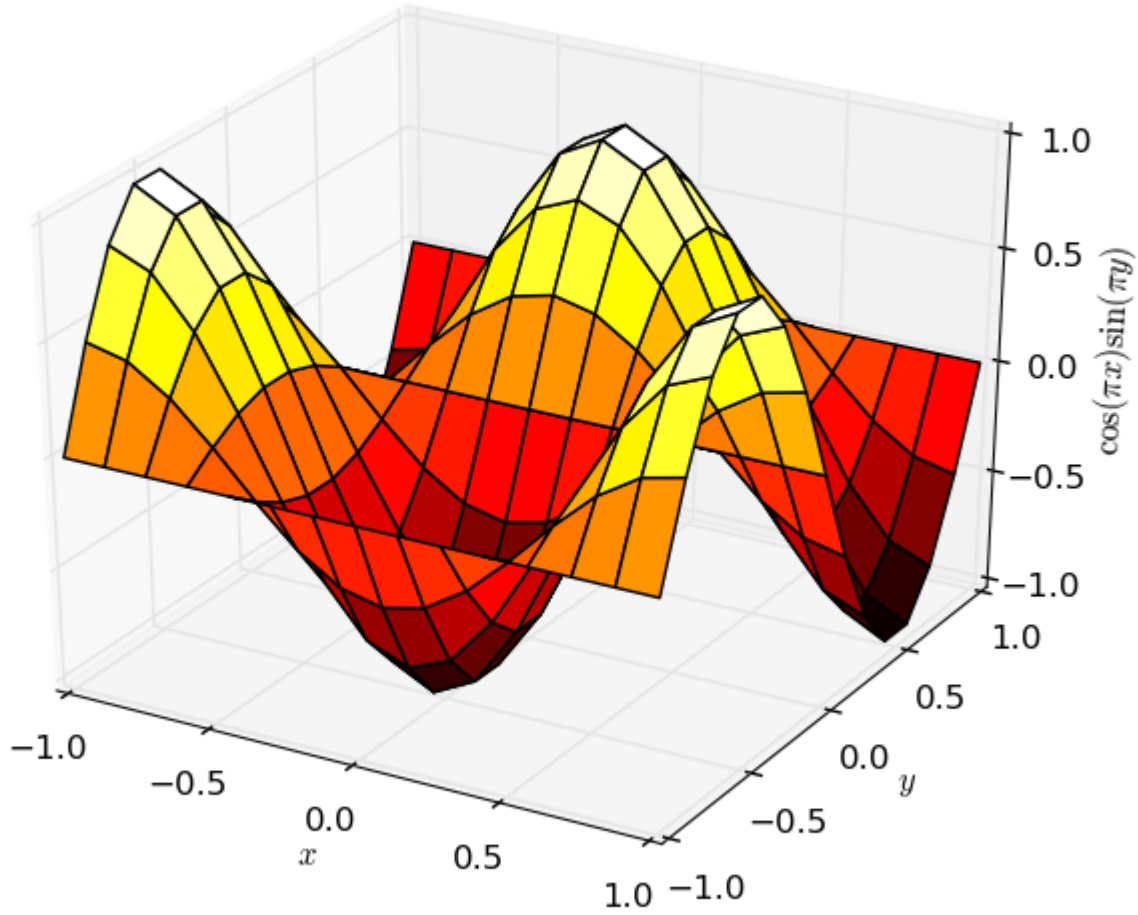
Создание трехмерных осей

По умолчанию оси Matplotlib являются двумерными. Чтобы создать трехмерные графики, нам нужно импортировать класс `Axes3D` из инструментария mplot3d , который позволит использовать новый вид проекции для осей, а именно `'3d'` :

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

Помимо прямых обобщений двумерных графиков (таких как линейные графики , диаграммы рассеяния , штриховые графики , контурные графики), доступны несколько методов построения поверхности , например `ax.plot_surface` :

```
# generate example data
import numpy as np
x,y = np.meshgrid(np.linspace(-1,1,15),np.linspace(-1,1,15))
z = np.cos(x*np.pi)*np.sin(y*np.pi)

# actual plotting example
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# rstride and cstride are row and column stride (step size)
ax.plot_surface(x,y,z,rstride=1,cstride=1,cmap='hot')
ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$y$')
ax.set_zlabel(r'$\cos(\pi x) \sin(\pi y)$')
plt.show()
```



Прочитайте Трехмерные графики онлайн: https://riptutorial.com/ru/matplotlib/topic/1880/трехмерные-графики

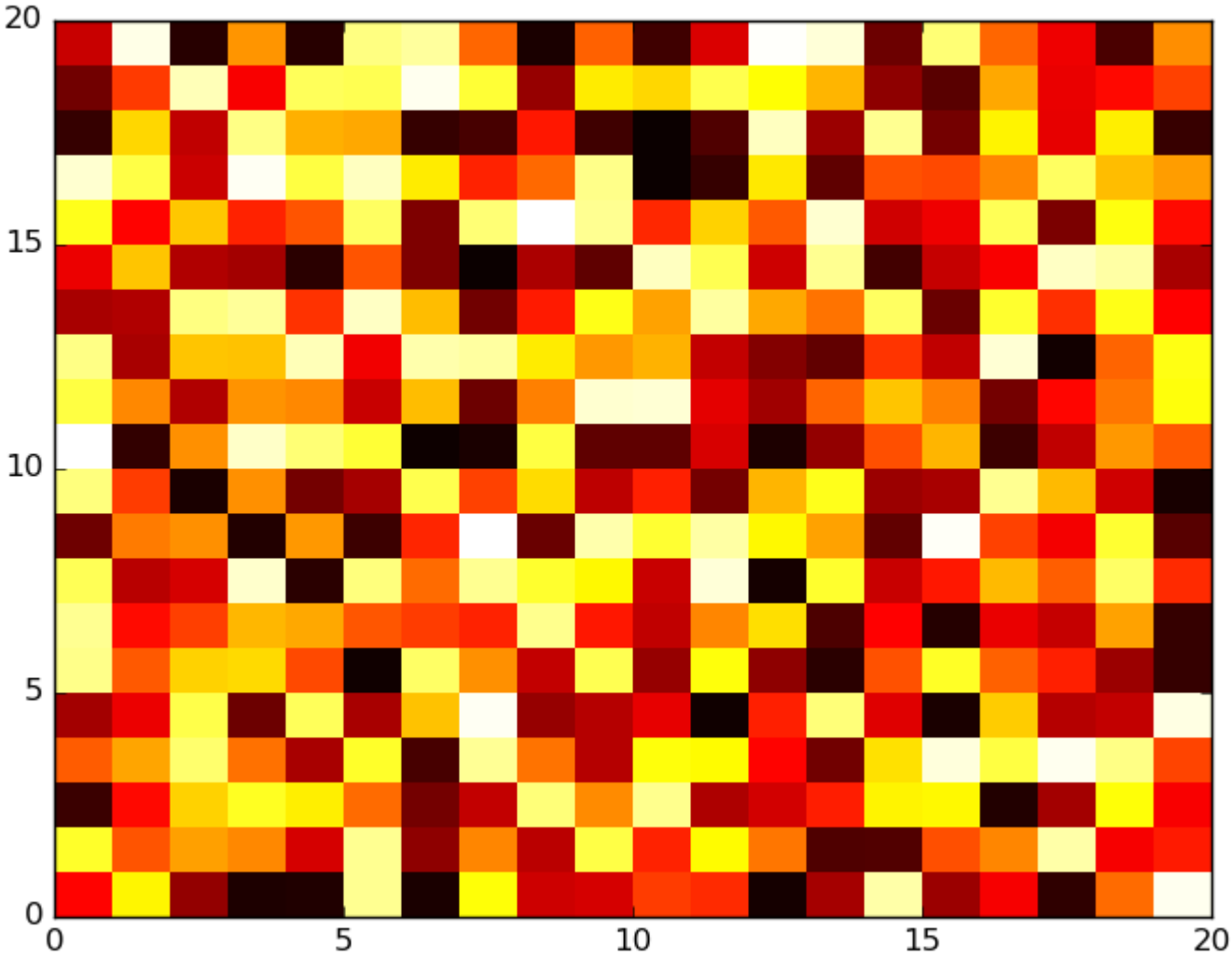
глава 18: цветовые карты

Examples

Основное использование

Использование встроенных цветовых папок так же просто, как передача имени требуемой цветовой карты (как указано в ссылке `contourf`) функции построения графика (например, `pcolormesh` или `contourf`), которая ожидает ее, как правило, в виде аргумента ключевого слова `cmap` :

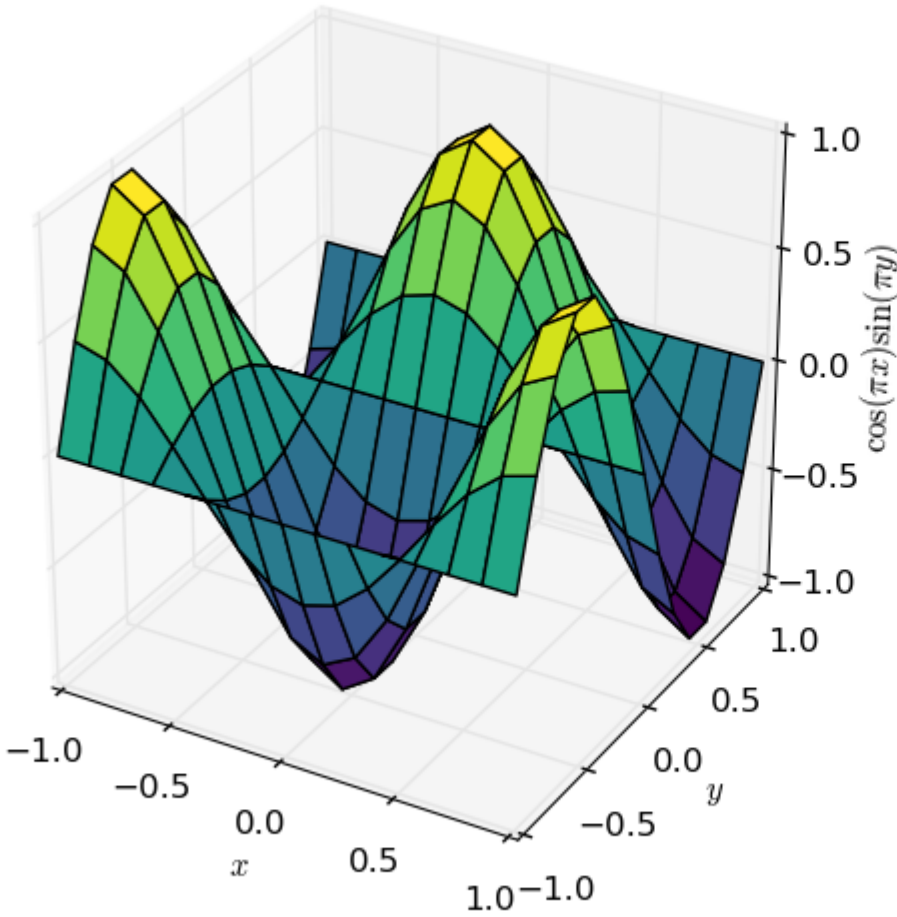
```
import matplotlib.pyplot as plt
import numpy as np

plt.figure()
plt.pcolormesh(np.random.rand(20,20),cmap='hot')
plt.show()
```



Колларды особенно полезны для визуализации трехмерных данных на двумерных графиках, но хорошая цветовая карта также может сделать правильный трехмерный график намного яснее:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.ticker import LinearLocator

# generate example data
import numpy as np
x,y = np.meshgrid(np.linspace(-1,1,15),np.linspace(-1,1,15))
z = np.cos(x*np.pi)*np.sin(y*np.pi)

# actual plotting example
fig = plt.figure()
ax1 = fig.add_subplot(121, projection='3d')
ax1.plot_surface(x,y,z,rstride=1,cstride=1,cmap='viridis')
ax2 = fig.add_subplot(122)
cf = ax2.contourf(x,y,z,51,vmin=-1,vmax=1,cmap='viridis')
cbar = fig.colorbar(cf)
cbar.locator = LinearLocator(numticks=11)
```

```
cbar.update_ticks()
for ax in {ax1, ax2}:
    ax.set_xlabel(r'$x$')
    ax.set_ylabel(r'$y$')
    ax.set_xlim([-1,1])
    ax.set_ylim([-1,1])
    ax.set_aspect('equal')

ax1.set_zlim([-1,1])
ax1.set_zlabel(r'$\cos(\pi x) \sin(\p    i y)$')

plt.show()
```



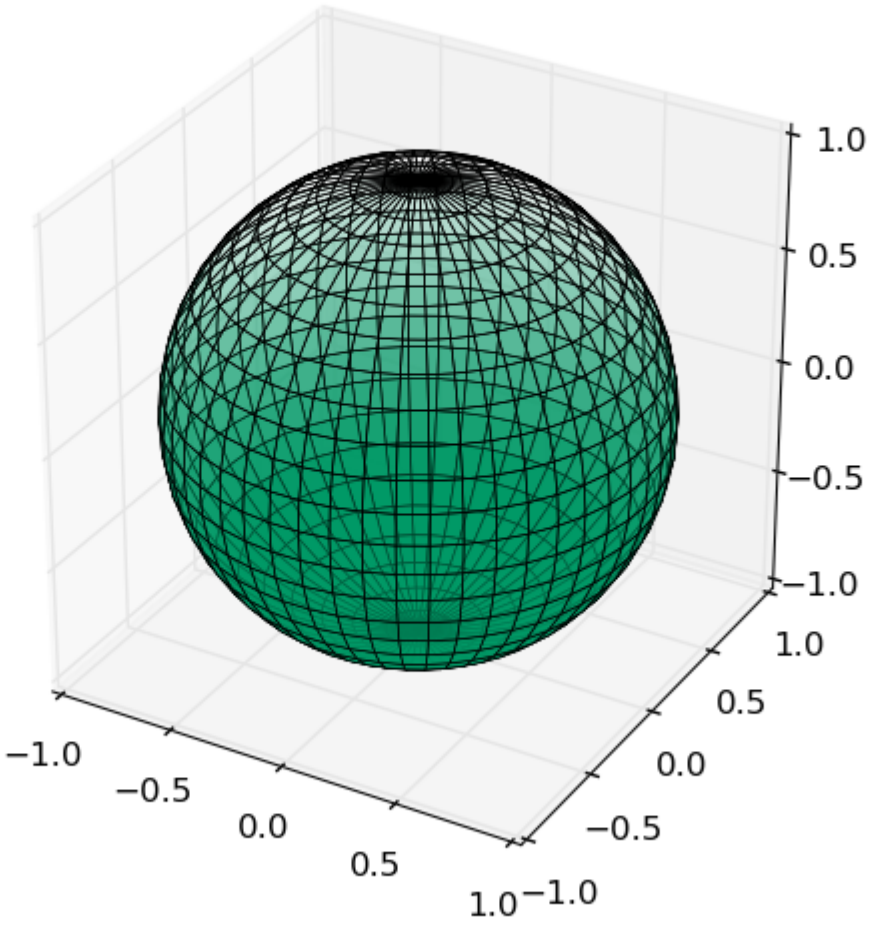
Использование пользовательских цветовых палитр

Помимо встроенных цветовых кодов, определенных в ссылке colormaps (и их обратных картах, при добавлении `'_r'` к их имени), могут также быть определены пользовательские цветовые карты. Ключом является модуль `matplotlib.cm` .

В приведенном ниже примере определяется очень простая `cm.register_cmap` , использующая

`cm.register_cmap` , содержащую один цвет, с непрозрачностью (альфа-значение) интерполяции цвета между полностью непрозрачным и полностью прозрачным в диапазоне данных. Обратите внимание, что важными линиями с точки зрения цветовой карты являются импорт `cm` , вызов `register_cmap` и прохождение `plot_surface` карты в `plot_surface` .

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.cm as cm

# generate data for sphere
from numpy import pi,meshgrid,linspace,sin,cos
th,ph = meshgrid(linspace(0,pi,25),linspace(0,2*pi,51))
x,y,z = sin(th)*cos(ph),sin(th)*sin(ph),cos(th)

# define custom colormap with fixed colour and alpha gradient
# use simple linear interpolation in the entire scale
cm.register_cmap(name='alpha_gradient',
                 data={'red':   [(0.,0,0),
                                 (1.,0,0)],

                       'green': [(0.,0.6,0.6),
                                 (1.,0.6,0.6)],

                       'blue':  [(0.,0.4,0.4),
                                 (1.,0.4,0.4)],

                       'alpha': [(0.,1,1),
                                 (1.,0,0)]})

# plot sphere with custom colormap; constrain mapping to between |z|=0.7 for enhanced effect
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x,y,z,cmap='alpha_gradient',vmin=-
0.7,vmax=0.7,rstride=1,cstride=1,linewidth=0.5,edgecolor='b')
ax.set_xlim([-1,1])
ax.set_ylim([-1,1])
ax.set_zlim([-1,1])
ax.set_aspect('equal')

plt.show()
```



В более сложных сценариях можно определить список значений R / G / B (/ A), в которые matplotlib интерполирует линейно, чтобы определить цвета, используемые на соответствующих графиках.

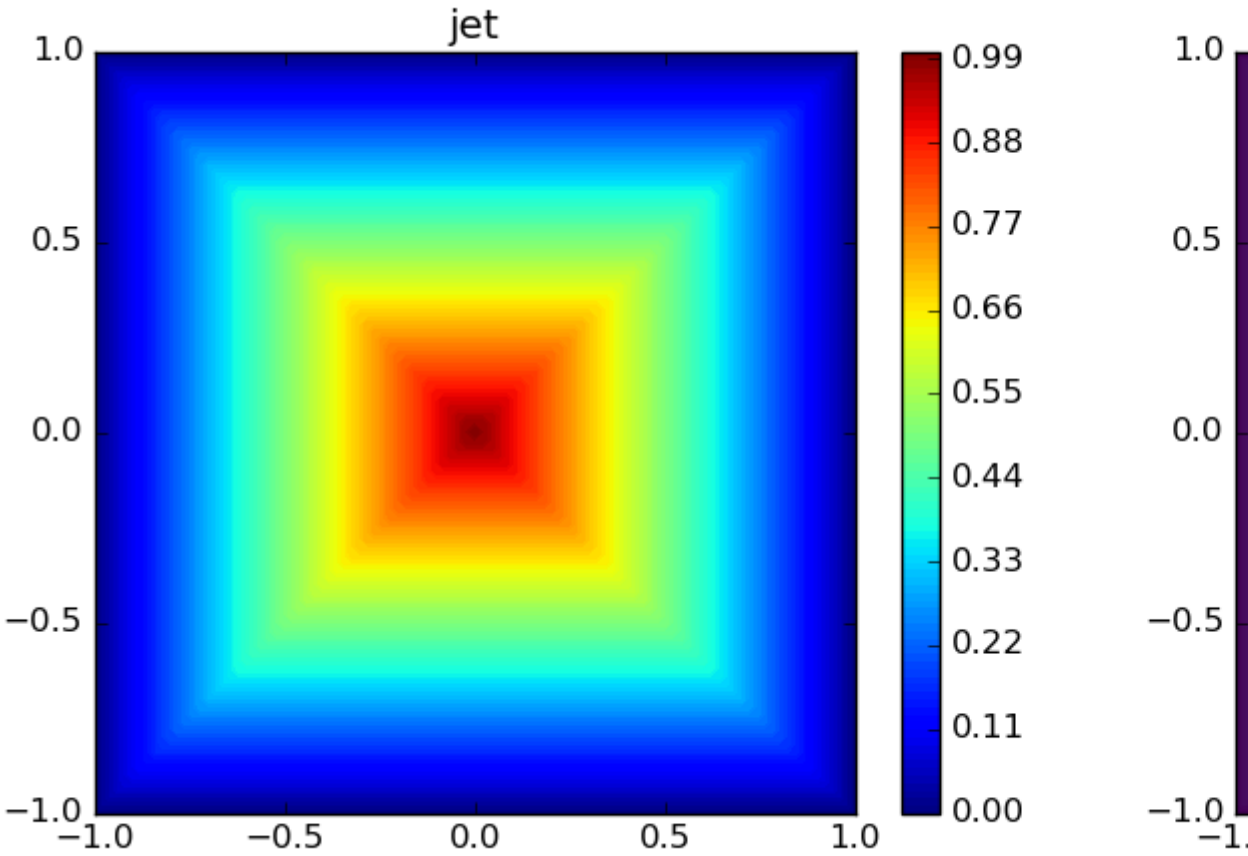
Порядочно однородные цветовые карты

Исходная по умолчанию цветовая карта MATLAB (замененная в версии R2014b), называемая `jet` является повсеместной из-за ее высокой контрастности и знакомости (и по умолчанию была использована matplotlib по соображениям совместимости). Несмотря на свою популярность, традиционные цветовые карты часто имеют недостатки, когда дело доходит до представления данных точно. Произошедшее изменение в этих цветовых картах не соответствует изменениям данных; и преобразование цветовой карты в оттенки серого (например, путем печати фигуры с использованием черно-белого принтера) может привести к потере информации.

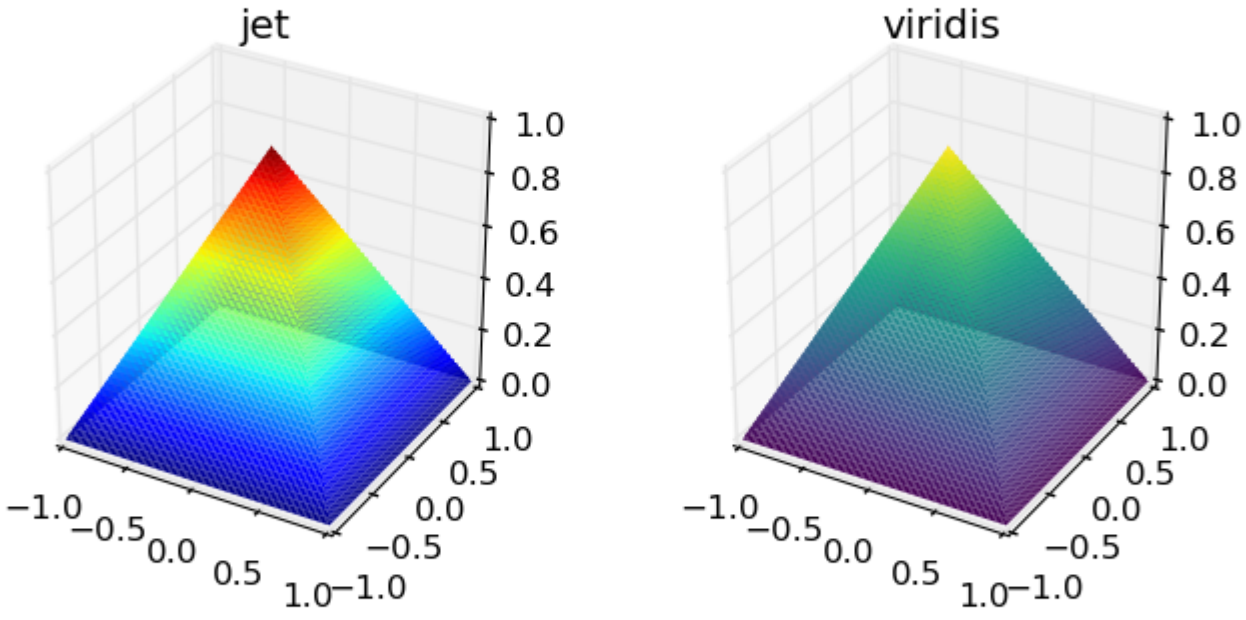
Были введены единые цветовые карты с восприятием, чтобы сделать визуализацию

данных максимально точной и доступной. Matplotlib представила четыре новых, перцепционно однородных цветовых паттерна в версии 1.5, причем один из них (по имени `viridis`) был по умолчанию от версии 2.0. Эти четыре цветовые карты (`viridis`, `inferno`, `plasma` и `magma`) являются оптимальными с точки зрения восприятия, и они должны использоваться для визуализации данных по умолчанию, если только нет веских причин не делать этого. Эти цветовые карты вводят как можно меньше предвзятости (не создавая функций, где их нет), и они подходят для аудитории с уменьшенным восприятием цвета.

В качестве примера для визуального искажения данных рассмотрим следующие два сюжета верхнего плана пирамидоподобных объектов:



Какая из двух является правильной пирамидой? Ответ конечно, что оба они есть, но это далеко не очевидно из сюжета, использующего `jet` карту:



Эта особенность лежит в основе единообразия восприятия.

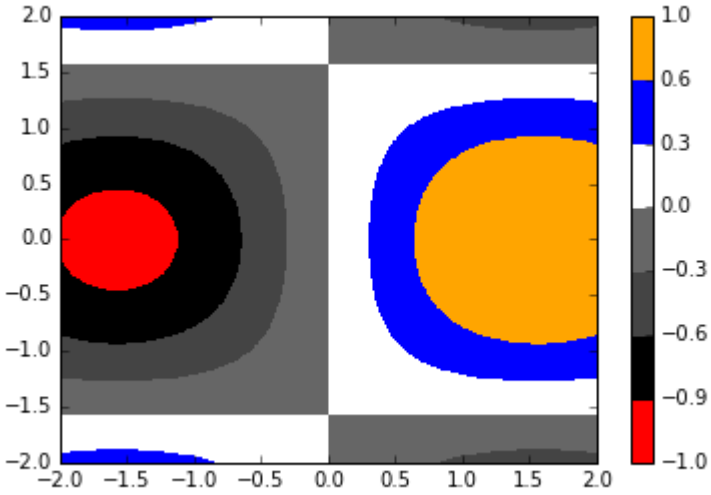
Пользовательская дискретная цветовая палитра

Если у вас есть предопределенные диапазоны и вы хотите использовать определенные цвета для этих диапазонов, вы можете объявить пользовательскую цветовую палитру. Например:

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

x = np.linspace(-2,2,500)
y = np.linspace(-2,2,500)
XX, YY = np.meshgrid(x, y)
Z = np.sin(XX) * np.cos(YY)

cmap = colors.ListedColormap(['red', '#000000','#444444', '#666666', '#ffffff', 'blue',
'orange'])
boundaries = [-1, -0.9, -0.6, -0.3, 0, 0.3, 0.6, 1]
```

```
norm = colors.BoundaryNorm(boundaries, cmap.N, clip=True)

plt.pcolormesh(x,y,Z, cmap=cmap, norm=norm)
plt.colorbar()
plt.show()
```

Производит



Цвет *i* будет использоваться для значений между границами *i* и *i* + *1* . Цвета могут быть указаны по именам (`'red'` , `'green'`), HTML-коды (`'#ffaa44'` , `'#441188'`) или кортежи RGB (`(0.2, 0.9, 0.45)`).

Прочитайте цветовые карты онлайн: https://riptutorial.com/ru/matplotlib/topic/3385/цветовые-карты

кредиты

S. No	Главы	Contributors
1	Начало работы с matplotlib	Amitay Stern, ChaoticTwist, Chr, Chris Mueller, Community, dermen, evtoh, farenorth, Josh, jrjc, pmos, Serenity, tacaswell
2	Анимация и интерактивный график	FiN, smurfendrek123, user2314737
3	Гистограмма	Yegor Kishilov
4	Графики LogLog	ml4294
5	Закрытие окна фигуры	Brian, David Zwicker
6	Интеграция с TeX / LaTeX	Andras Deak, Bosoneando, Chris Mueller, Næreen, Serenity
7	Контурные карты	Eugene Loy, Serenity
8	Легенды	Andras Deak, Franck Dernoncourt, ronrest, saintsfan342000, Serenity
9	Манипуляция изображениями	Bosoneando
10	Несколько участков	Chris Mueller, Robert Branam, ronrest, swatchai
11	Объекты фигур и осей	David Zwicker, Josh, Serenity, tom
12	Основные сюжеты	Franck Dernoncourt, Josh, ml4294, ronrest, Scimonster, Serenity, user2314737
13	присущи рефлективный, вербальный	Luis
14	Системы координат	jure
15	Строки сетки и метки	ronrest

16	Трехмерные графики	Andras Deak, Serenity, will
17	цветовые карты	Andras Deak, Xevaquor