Тема 6. 3D графики.

П20. Трехмерный график

Все классы для работы с трехмерными графиками находятся в пакете `mpl_toolkits.mplot3d`, из которого нужно будет их импортировать.

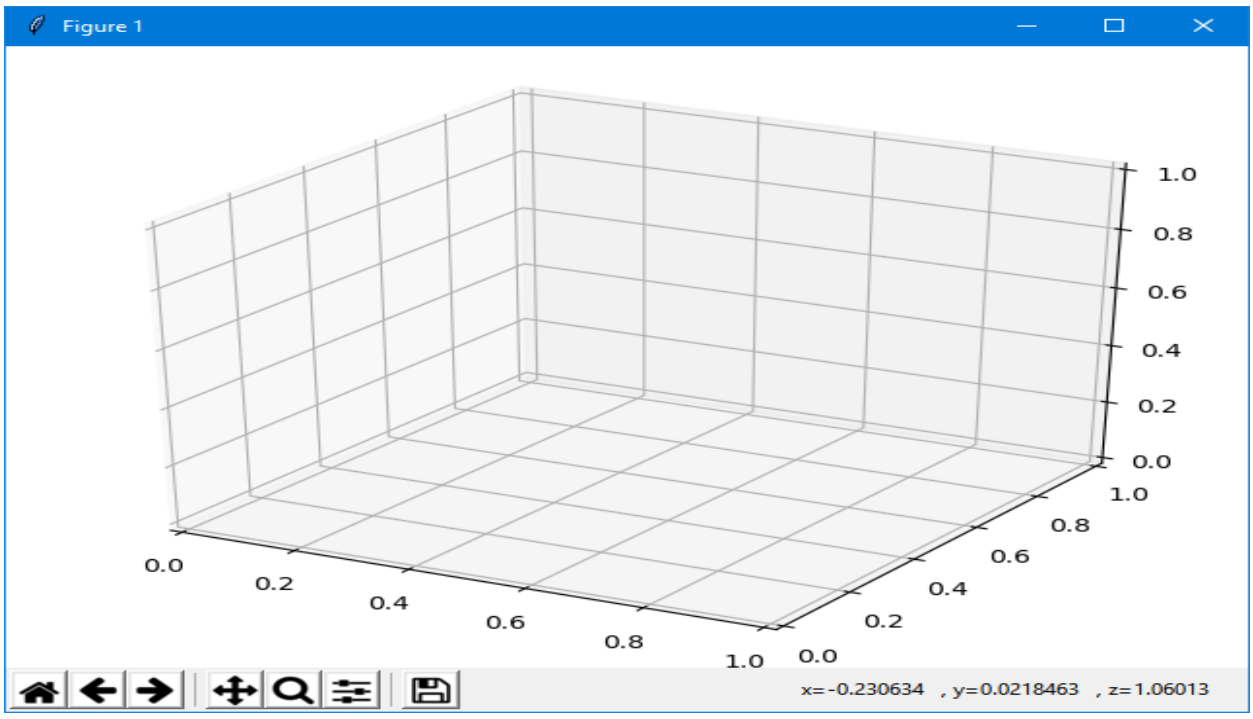
Для того, чтобы нарисовать трехмерный график, в первую очередь надо создать трехмерные оси.

Чтобы их создать, нужно создать экземпляр класса *mpl_toolkits.mplot3d.Axes3D.* Его конструктор ожидает, как минимум, один параметр - экземпляр класса *matplotlib.figure.Figure.* Этот объект создается вызовом *pylab.figure()*. У конструктора класса *matplotlib.figure.Figure* есть еще и другие необязательные параметры (по материалам сайта http://jenyay.net/Programming/Python3d).

Нарисуем пустые оси:

```
import numpy as np
import pylab
from mpl_toolkits.mplot3d import Axes3D

fig = pylab.figure()
Axes3D(fig)
pylab.show()
```

В результате увидим следующее окно:



Полученные оси можно вращать мышкой.

П21. Трёхмерная линия

Линия в пространстве задаётся параметрически: $x=x(t)$, , $y=y(t)$, $z=z(t)$. Например так:

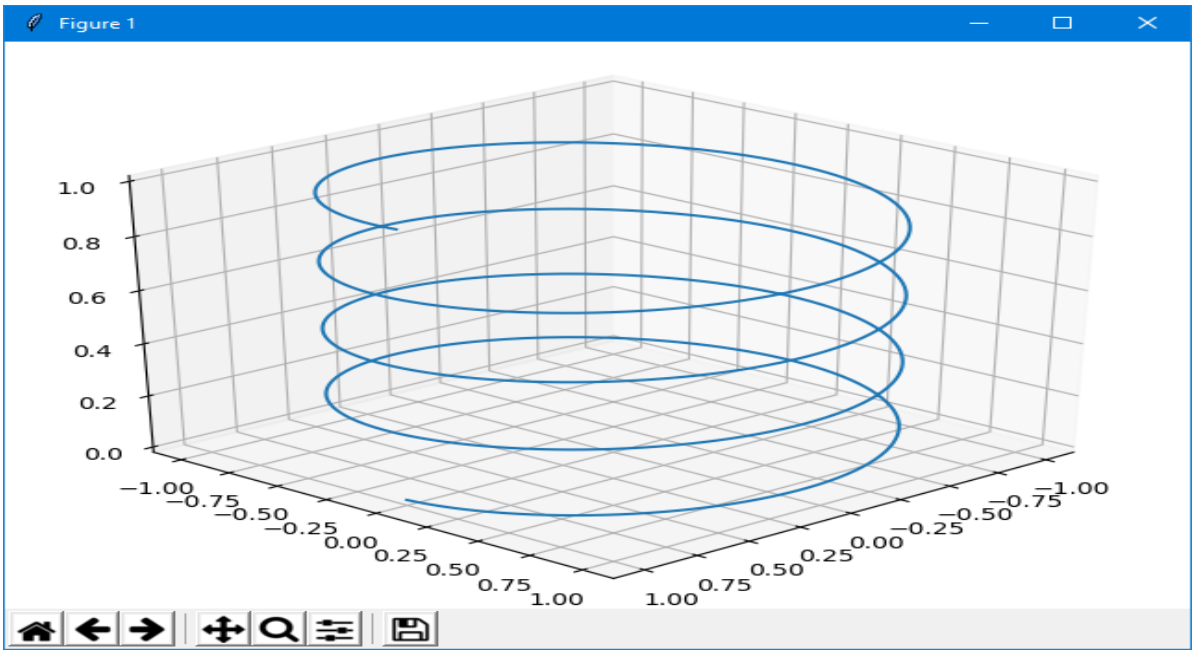
```
t=np.linspace(0, 4*np.pi,1000)
x=np.cos(2*t)
y=np.sin(2*t)
z=t /(4*np.pi)
```

Для построения и визуализации используется объект класса Axes3D из пакета `mpl_toolkits.mplot3d`:

```
import pylab
import mpl_toolkits.mplot3d as A3D
```

`figure()` - это текущий рисунок, создаём в нём объект ax, потом используем его методы для визуализации

```
import pylab
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
##from mpl_toolkits.mplot3d import Axes3D
import mpl_toolkits.mplot3d as A3D

t=np.linspace(0, 4*np.pi,1000)
x=np.cos(2*t)
y=np.sin(2*t)
z=t /(4*np.pi)

#Тут нужен объект класса Axes3D из пакета
mpl_toolkits.mplot3d.
# figure() - это текущий рисунок, создаём в нём объект
ax,
# потом используем его методы.
fig=pylab.figure()
ax=A3D.Axes3D(fig)
ax.elev,ax.azim=30,45 # задать, с какой стороны
смотрим.

ax.plot3D(x,y,z)
plt.show()
```



Построим три прямые по осям координат:

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
axes = axes3d.Axes3D(fig)

t = np.arange(-1, 1, 0.01)
x, y = np.meshgrid(t, t)

axes.plot3D(t, 0*t, 0*t, color='k' )
axes.plot3D(0*t, t, 0*t, color='k' )
axes.plot3D(0*t, 0*t, t, color='k' )

plt.show()
```

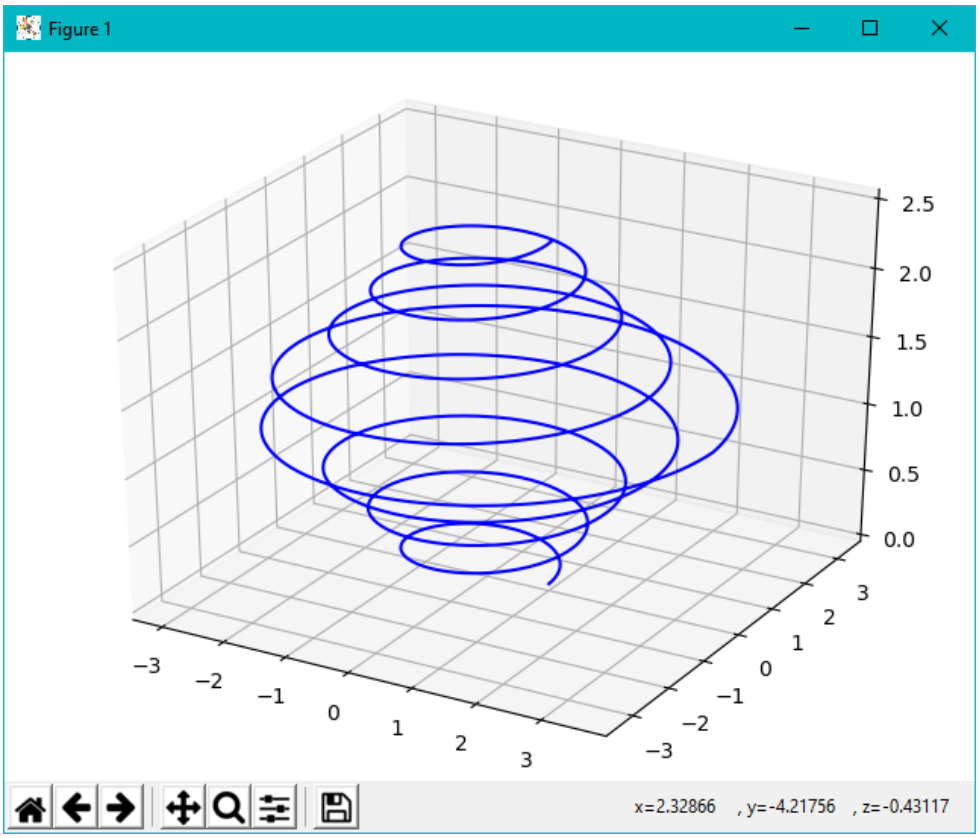
Построим линию, заданную уравнениями:

$$x(t)=\cos(2\cdot t)\cdot e^{\frac{a\cdot t}{10}} \qquad y(t)=\sin(2\cdot t)\cdot e^{\frac{a\cdot t}{10}} \qquad z(t)=\frac{t}{10}$$

$$t\in[0,8\cdot\pi] \qquad a=\begin{cases}-1, & npu \quad t\in[0,4\cdot\pi)\\ +1, & npu \quad t\in[4\cdot\pi,\ \ 8\cdot\pi]\end{cases}$$

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d as A3D

tt1=0; tt2=4*np.pi; tt3=8*np.pi; n1=500; n2=500
```

```
#-------
t1=np.linspace(tt1, tt2, n1)
x1=np.cos(2*t1) *np.exp(0.1*t1)
y1=np.sin(2*t1) *np.exp(0.1*t1)
z1=0.1*t1
#------
ttt=t1[500-1]
t2=np.linspace(tt2, tt3, n2)
x2=np.cos(2*(t2)) *np.exp(-0.1*(t2-ttt)+0.1*ttt) #+
x1[500-2]
y2=np.sin(2*(t2)) *np.exp(-0.1*(t2-ttt)+0.1*ttt) #+
y1[500-2]
z2=0.1*t2
#-----
#print("x=  ", x1[500-1], x2[0], t1[500-1], t2[0])
#print("y=  ", y1[500-1], y2[0], t1[500-1], t2[0])
#-----
x=np.concatenate((x1[:-1],x2))
y=np.concatenate((y1[:-1],y2))
z=np.concatenate((z1[:-1],z2))
#----
fig=plt.figure()
ax=A3D.Axes3D(fig)
ax.plot3D(x,y,z,'b-')
plt.show()
```



П22. Поверхности

Все поверхности задаются параметрические: ***x=x(u,v), , y=y(u,v), z=z(u,v).***

Если мы хотим задать поверхность «явно» $z=z(x,y)$, то удобно создать массивы $x=u$ и $y=v$ функцией **`meshgrid`**.

Для всех примеров будем использовать следующую функцию, от двух координат.

$$f(x,y)=\frac{\sin(x)\sin(y)}{xy}$$

Для начала нужно подготовить данные для рисования. Нам понадобятся *три двумерные матрицы*:

- матрицы *X* и *Y* будут хранить координаты сетки точек, в которых будет вычисляться приведенная выше функция,
- матрица *Z* будет хранить значения этой функции в соответствующей точке.

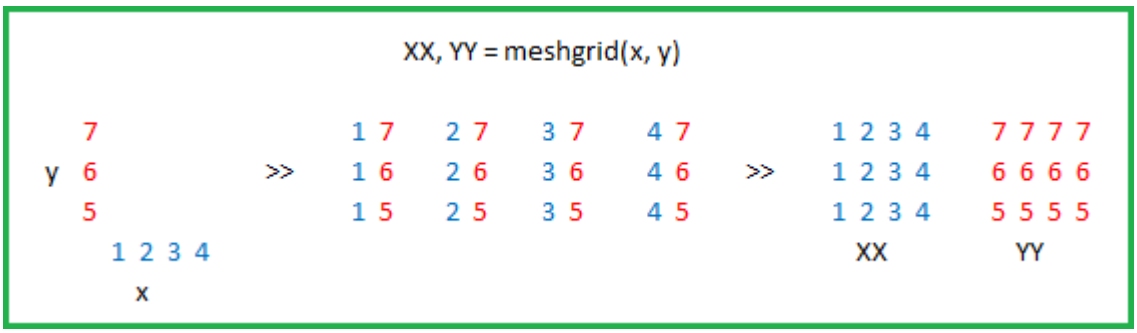
Если мы хотим нарисовать трехмерный график на эквидистантной сетке (на сетке, у которой расстояние между точками одинаковое), то для создания матриц, которые будут хранить координаты, будем использовать функцию **`meshgrid()`** из библиотеки `numpy`.

Эта функция создает двумерные матрицы сеток по одномерным массивам. Работа этой функции очень наглядно показана ниже:

```
>>> X, Y = numpy.meshgrid([1,2,3], [4,5,6,7])
>>> X
array([[1, 2, 3],
       [1, 2, 3],
       [1, 2, 3],
       [1, 2, 3]])
>>> Y
array([[4, 4, 4],
       [5, 5, 5],
       [6, 6, 6],
       [7, 7, 7]])
```

Теперь по индексу узла сетки можно узнать реальные координаты: X[0][0] = 1, Y[0][0] = 4 и т.п.

Фактически функция **`[X, Y] = meshgrid(x, y)`** задает сетку на плоскости x-y в виде двумерных массивов X, Y, которые определяются одномерными массивами x и y. Строки массива X являются копиями вектора x, а столбцы - копиями вектора y. Формирование таких массивов упрощает вычисление функций двух переменных, позволяя применять операции над массивами. Ниже приведем рисунок поясняющий построение массивов X и Y:



Чтобы отделить подготовку данных от самого рисования, создание сетки и расчет функции выделим в отдельную функцию:

```
def makeData ():
    # Строим сетку в интервале от -10 до 10
    # с шагом 0.1 по обоим координатам
    x = numpy.arange (-10, 10, 0.1)
    y = numpy.arange (-10, 10, 0.1)
    # Создаем двумерную матрицу-сетку
    xgrid, ygrid = numpy.meshgrid(x, y)
    # В узлах рассчитываем значение функции
    zgrid = numpy.sin (xgrid) * numpy.sin (ygrid) /
                                          (xgrid * ygrid)
    return xgrid, ygrid, zgrid
```

Эта функция возвращает три двумерные матрицы: x, y, z. Координаты x и y лежат в интервале от -10 до 10 с шагом 0.1.

Теперь возвращаемся непосредственно к рисованию. Чтобы отобразить наши данные, достаточно вызвать метод *plot_surface()* экземпляра класса *Axes3D*, в который передадим полученные с помощью функции *makeData()* двумерные матрицы.

Теперь наш пример выглядит следующим образом:

```
import pylab
from mpl_toolkits.mplot3d import Axes3D
import numpy

def makeData ():
    x = numpy.arange (-10, 10, 0.1)
    y = numpy.arange (-10, 10, 0.1)
    xgrid, ygrid = numpy.meshgrid(x, y)
    zgrid = numpy.sin (xgrid) * numpy.sin (ygrid) /
                                          (xgrid * ygrid)
    return xgrid, ygrid, zgrid

x, y, z = makeData()
fig = pylab.figure()
axes = Axes3D(fig)
axes.plot_surface(x, y, z)
pylab.show()
```

В новых версиях matplotlib вместо

```
axes = Axes3D(fig)
```

рекомендуется кодировать:

```
fig = pylab.figure()
axes = Axes3D(fig,auto_add_to_figure=False)
fig.add_axes(axes)
```

Так, например, можно «подписать» оси координат:

```
axes.set_xlabel("-- x -->")
axes.set_ylabel("-- y -->")
axes.set_zlabel("-- z -->")
```

Если мы запустим этот скрипт, то появится окно:

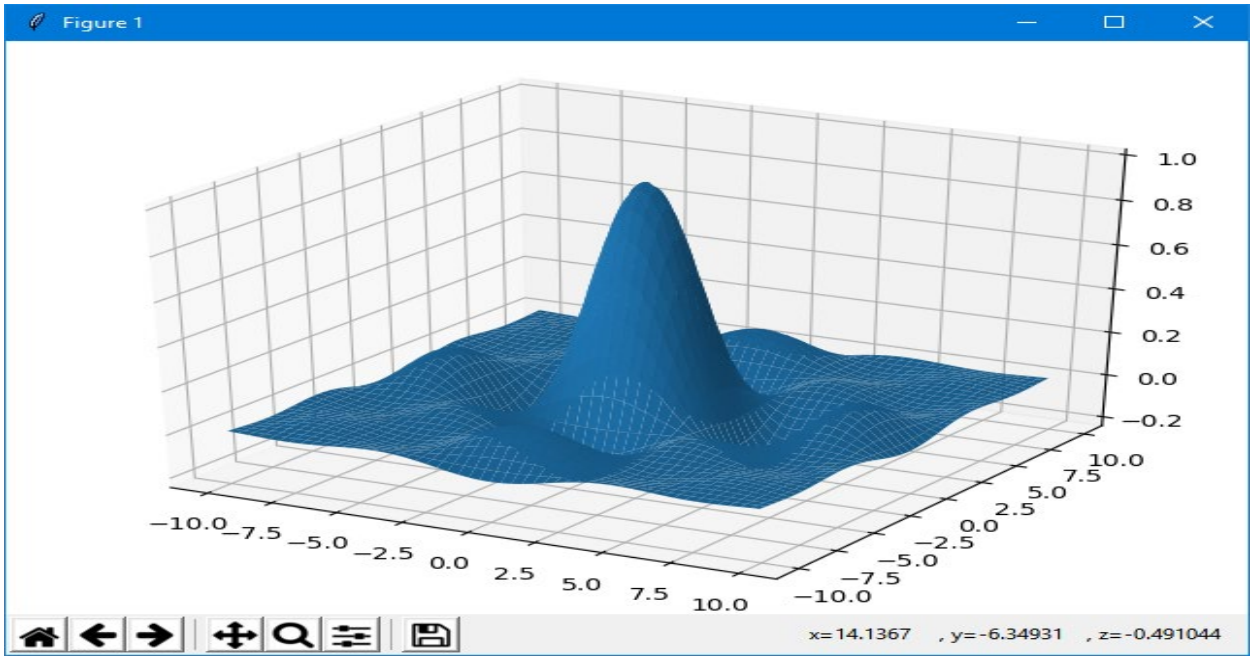


Рисунок можно вращать с помощью мышки. Matplotlib не использует графический ускоритель, поэтому вращение происходит довольно медленно, хотя скорость зависит от количества точек на поверхности.

Рассмотрим остальные параметры метода *Axes3D.plot_surface(X, Y, Z, *args, **kwargs)*, их не так много:

- *X*, *Y* и *Z* - эти параметры задают сетку и значение функции в узлах.
- *rstride* и *cstride* задают шаг вывода графика. Чем меньше шаг, тем точнее отображается функция, но тем дольше происходит рисование.
- *color* - задает цвет графика
- *cmap* - задает градиент цветов, чтобы цвет ячейки графика зависел от значения функции в этой области.

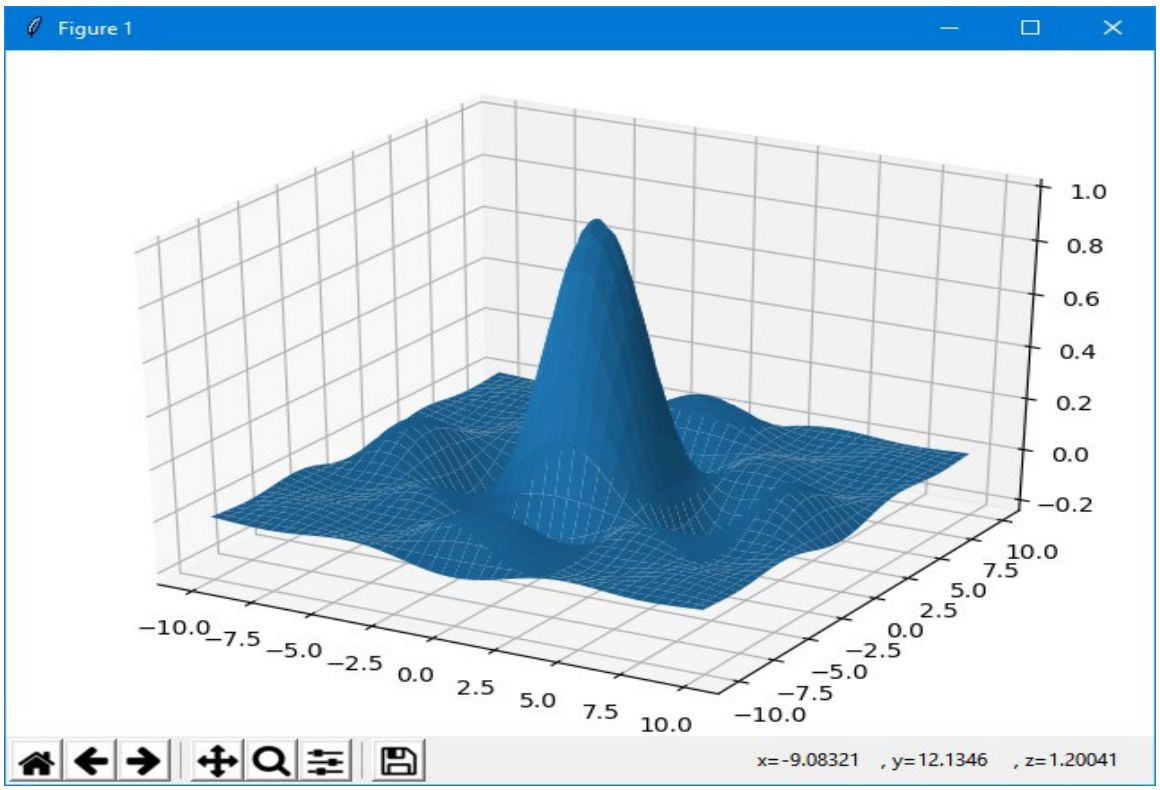
Рассмотрим эти параметры.

Шаг сетки

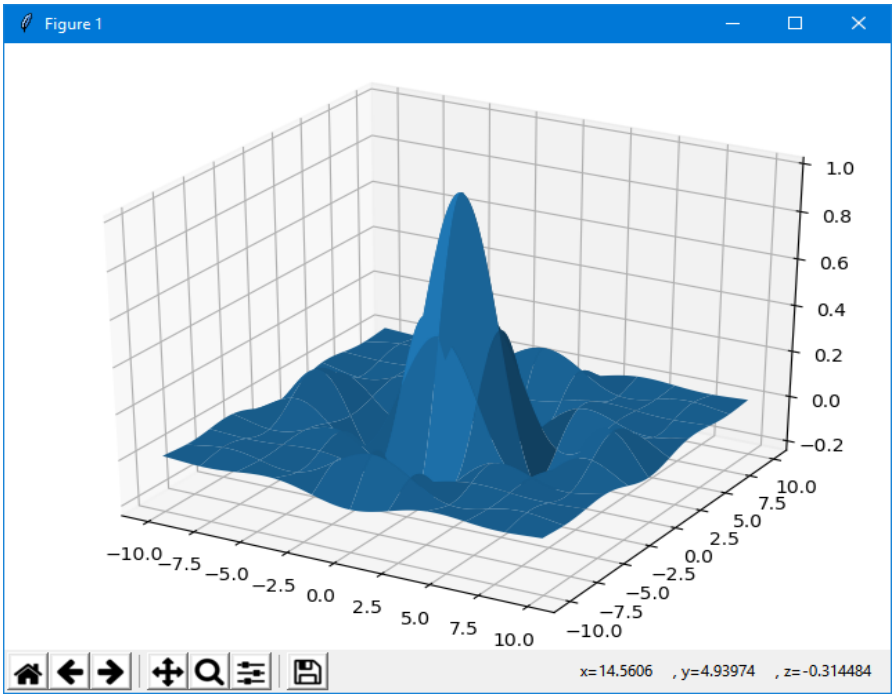
Сначала посмотрим как влияют на внешний вид параметры *rstride* и *cstride*. Изменим в предыдущем примере строку с использованием метода *plot_surface()* следующим образом:

```
axes.plot_surface(x, y, z, rstride=5, cstride=5)
```

В результате мы получим более мелкую сетку на графике:



Если таким же образом установить значения этих параметров в 20, то сетка будет наоборот более крупная:



Изменение цвета

Теперь изменим цвет поверхности с помощью параметра *color*. Этот параметр представляет собой строку, которая описывает цвет. Строка цвета может задаваться разными способами:

Цвет можно определить английским словом для соответствующего цвета или одной буквой. Таких цветов не много:

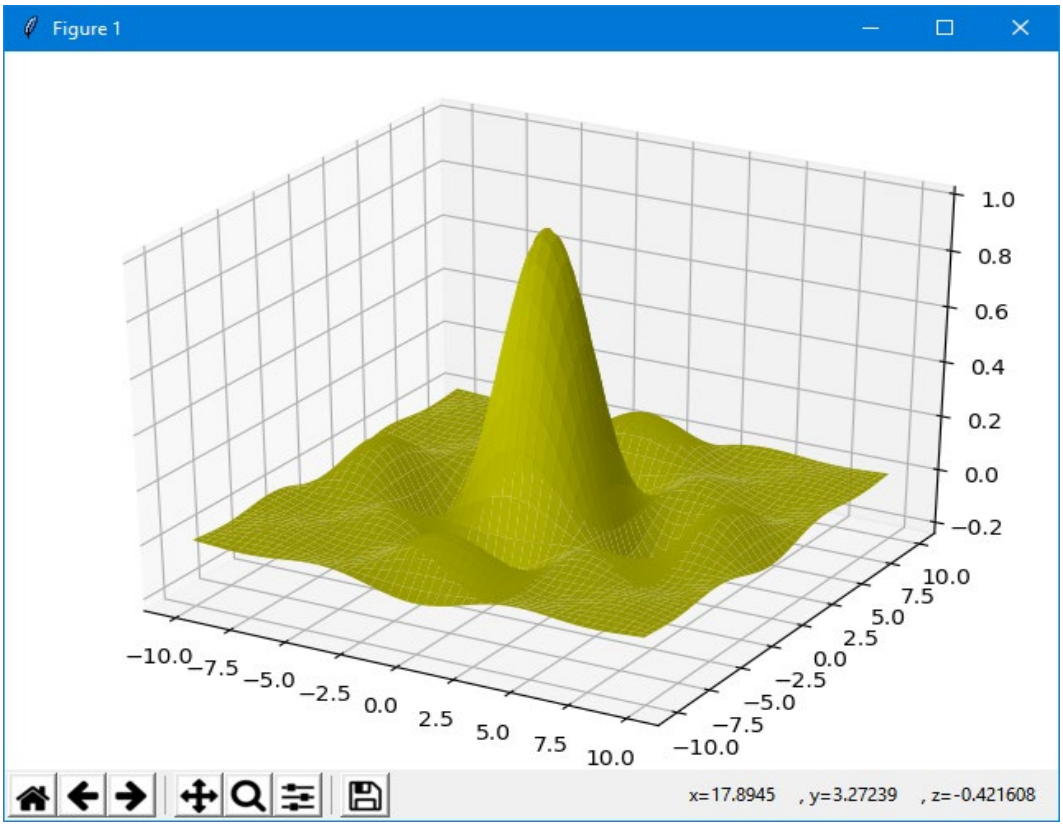
- 'b' или 'blue'
- 'g' или 'green'
- 'r' или 'red'
- 'c' или 'cyan'
- 'm' или 'magenta'
- 'y' или 'yellow'
- 'k' или 'black'
- 'w' или 'white'

Для примера сделаем поверхность желтой:

```
axes.plot_surface(x, y, z, color='yellow')
```

или

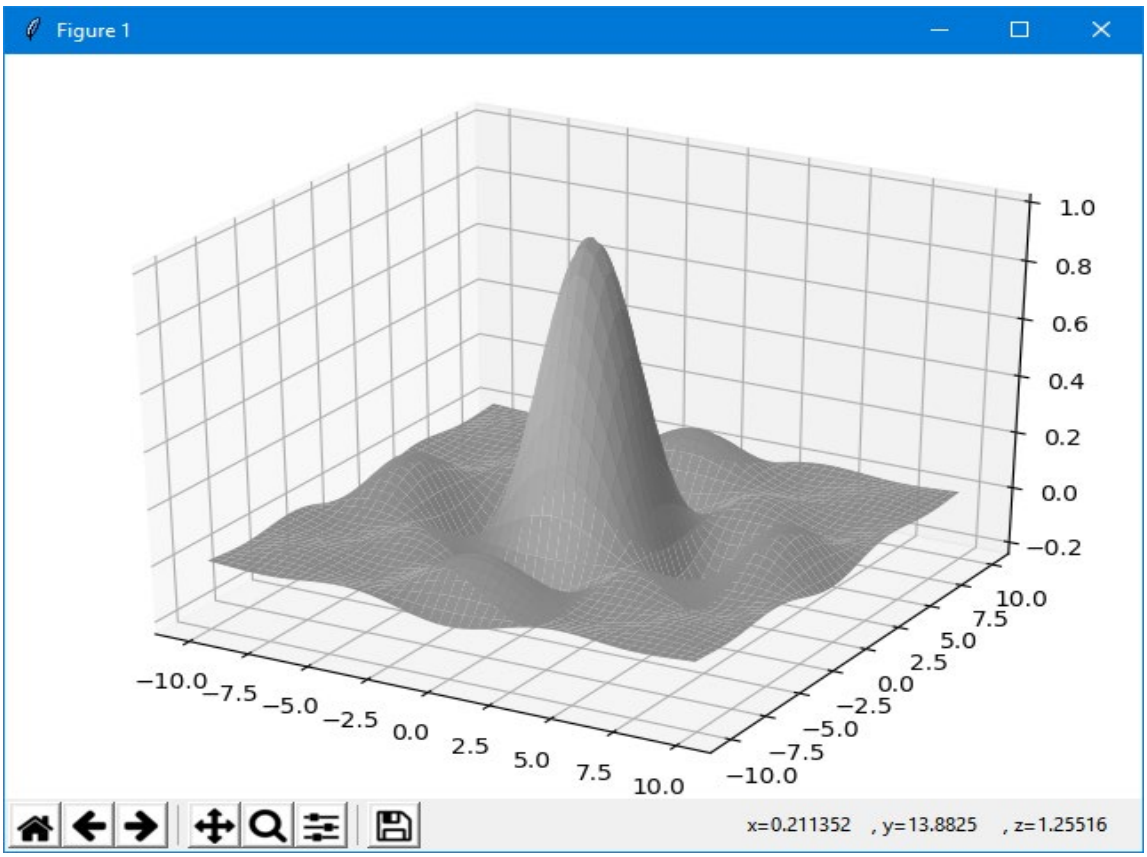
```
axes.plot_surface(x, y, z, color='y')
```



Если нам нужен серый цвет, то его яркость можем задать с помощью строки, содержащей число в интервале от 0.0 до 1.0 (0 - белый, 1 - черный). Например, можно написать следующую строку:

```
axes.plot_surface(x, y, z, color='0.7')
```

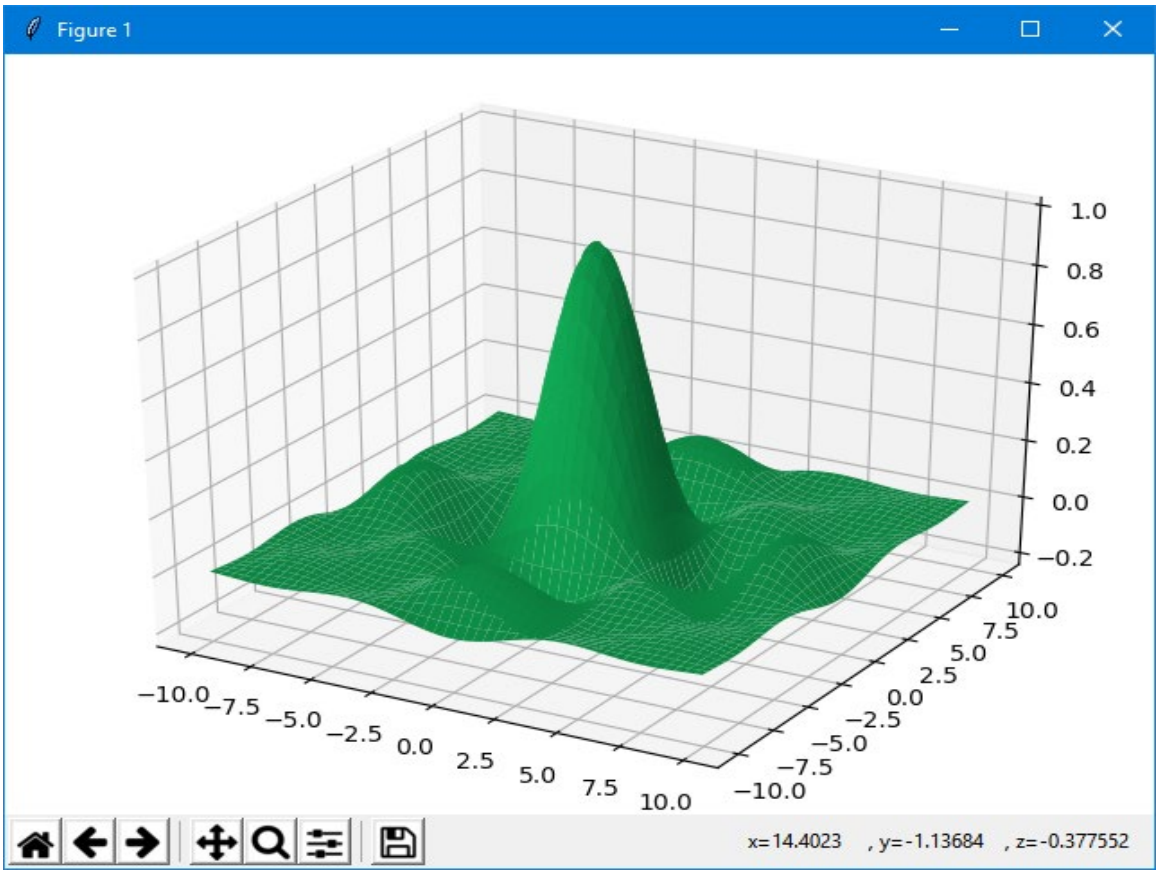
В этом случае мы увидим такой вот серый график:



Кроме того мы можем задавать цвет так как это принято в HTML после символа решетки ('#'). Например, можем задать цвет следующим образом:

```
axes.plot_surface(x, y, z, color='#11aa55')
```

Тогда график позеленеет:



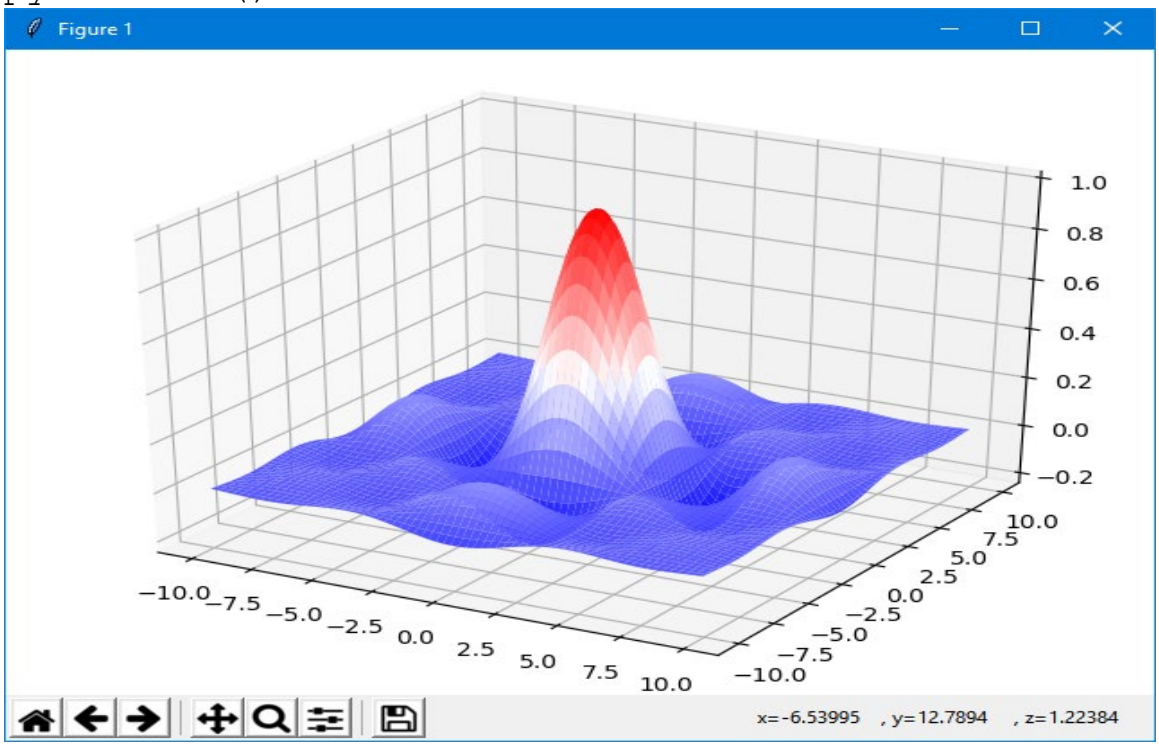
Использование цветовых карт (colormap)

Цветовые карты используются, если нужно указать в какие цвета должны окрашиваться участки трехмерной поверхности в зависимости от значения Z в этой области (задание цветового градиента).

Чтобы при выводе графика использовался градиент, в качестве значения параметра *cmap* (от слова colormap, цветовая карта) нужно передать экземпляр класса *matplotlib.colors.Colormap* или производного от него.

Следующий пример использует класс *LinearSegmentedColormap*, производный от *Colormap*, чтобы создать градиент перехода от синего цвета к красному через белый.

```
import pylab
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import LinearSegmentedColormap
import numpy
def makeData ():
    x = numpy.arange (-10, 10, 0.1)
    y = numpy.arange (-10, 10, 0.1)
    xgrid, ygrid = numpy.meshgrid(x, y)
    zgrid = numpy.sin (xgrid) * numpy.sin (ygrid) /
            (xgrid * ygrid)
    return xgrid, ygrid, zgrid
x, y, z = makeData()
fig = pylab.figure()
axes = Axes3D(fig)
axes.plot_surface(x, y, z, rstride=3, cstride=3,  \
cmap = LinearSegmentedColormap.from_list ("red_blue",
                                  ['b', 'w', 'r'], 256))
pylab.show()
```

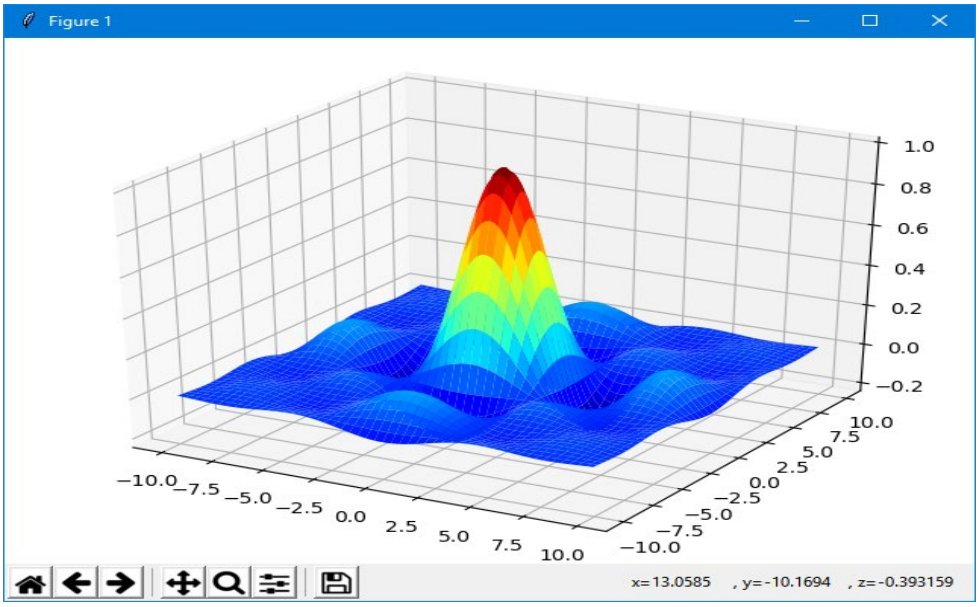


Здесь используется статический метод `from_list()`, который принимает три параметра:

- Имя создаваемой карты
- Список цветов, начиная с цвета для минимального значения на графике (голубой - 'b'), через промежуточные цвета (у нас это белый - 'w') к цвету для максимального значения функции (красный - 'r').
- Количество цветовых переходов. Чем это число больше, тем более плавный градиент, но тем больше памяти он занимает.

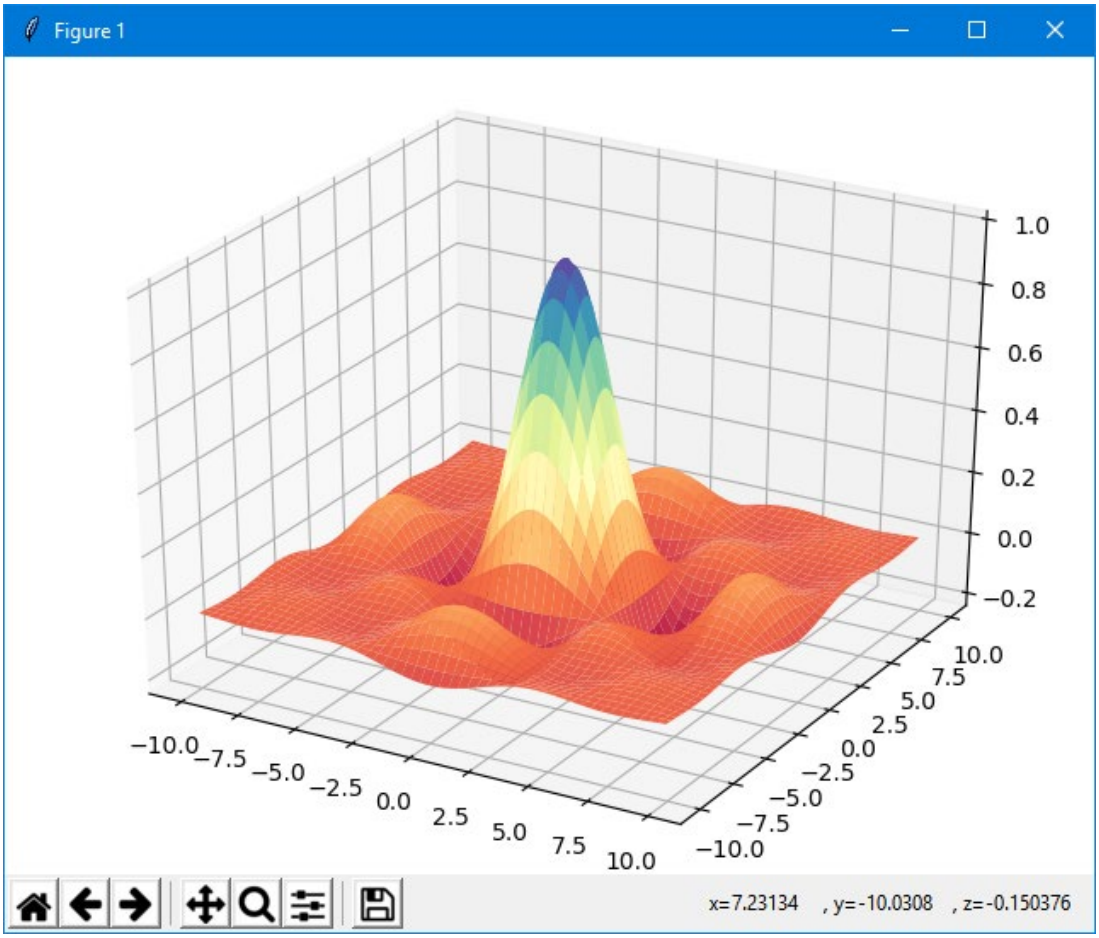
Карта ***cm.jet*** - это, наверное, самая часто используемая карта в примерах.

```
import pylab
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import LinearSegmentedColormap
from matplotlib import cm
import numpy
def makeData():
    x = numpy.arange (-10, 10, 0.1)
    y = numpy.arange (-10, 10, 0.1)
    xgrid, ygrid = numpy.meshgrid(x, y)
    zgrid = numpy.sin (xgrid) * numpy.sin (ygrid) /
                                    (xgrid * ygrid)
    return xgrid, ygrid, zgrid
x, y, z = makeData()
fig = pylab.figure()
axes = Axes3D(fig)
axes.plot_surface(x, y, z, rstride=4, cstride=4,
                                    cmap = cm.jet)
pylab.show()
```



Обратите внимание, что в качестве аргумента *cmap* мы передаем не строку, а имя переменной из модуля *cm*, причем это уже созданный экземпляр класса, а не имя класса. Так, например, *cm.jet* - это экземпляр класса *matplotlib.colors.LinearSegmentedColormap*.

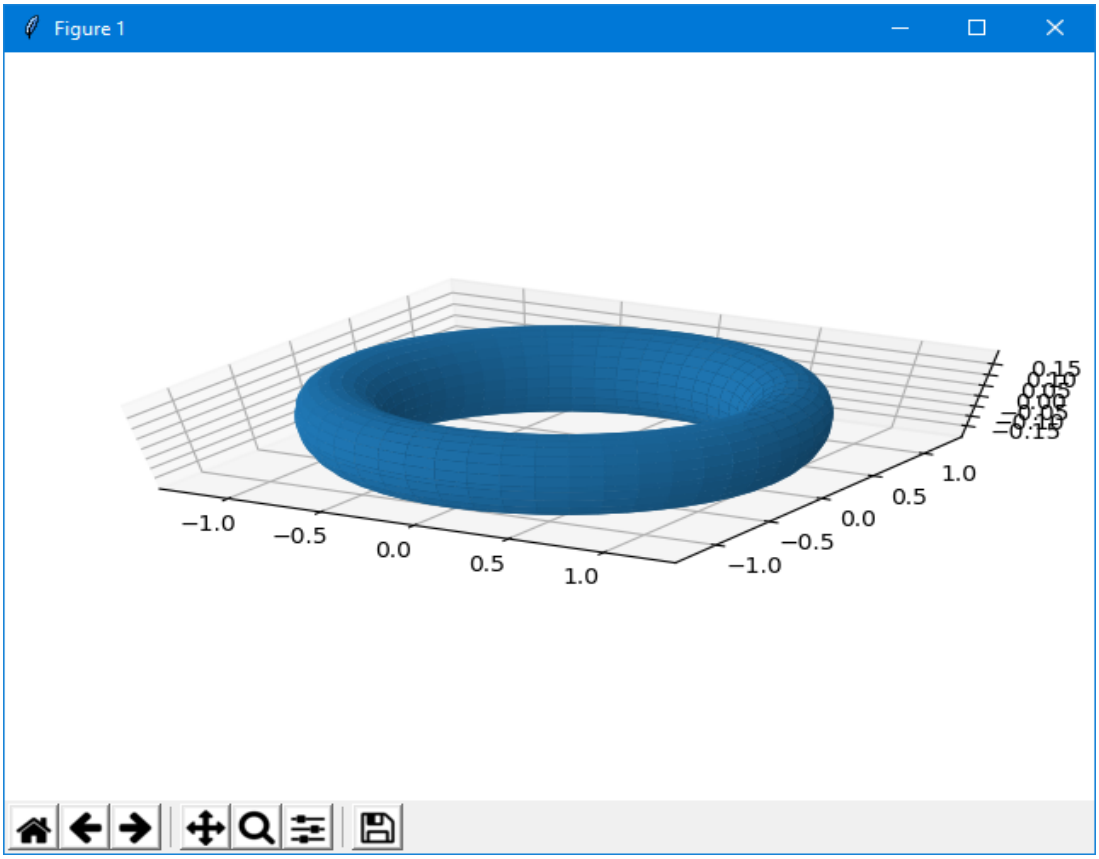
Для примера цветовая карта *cm.Spectral* выглядит следующим образом:



П23. Параметрические поверхности с параметрами ϑ φ

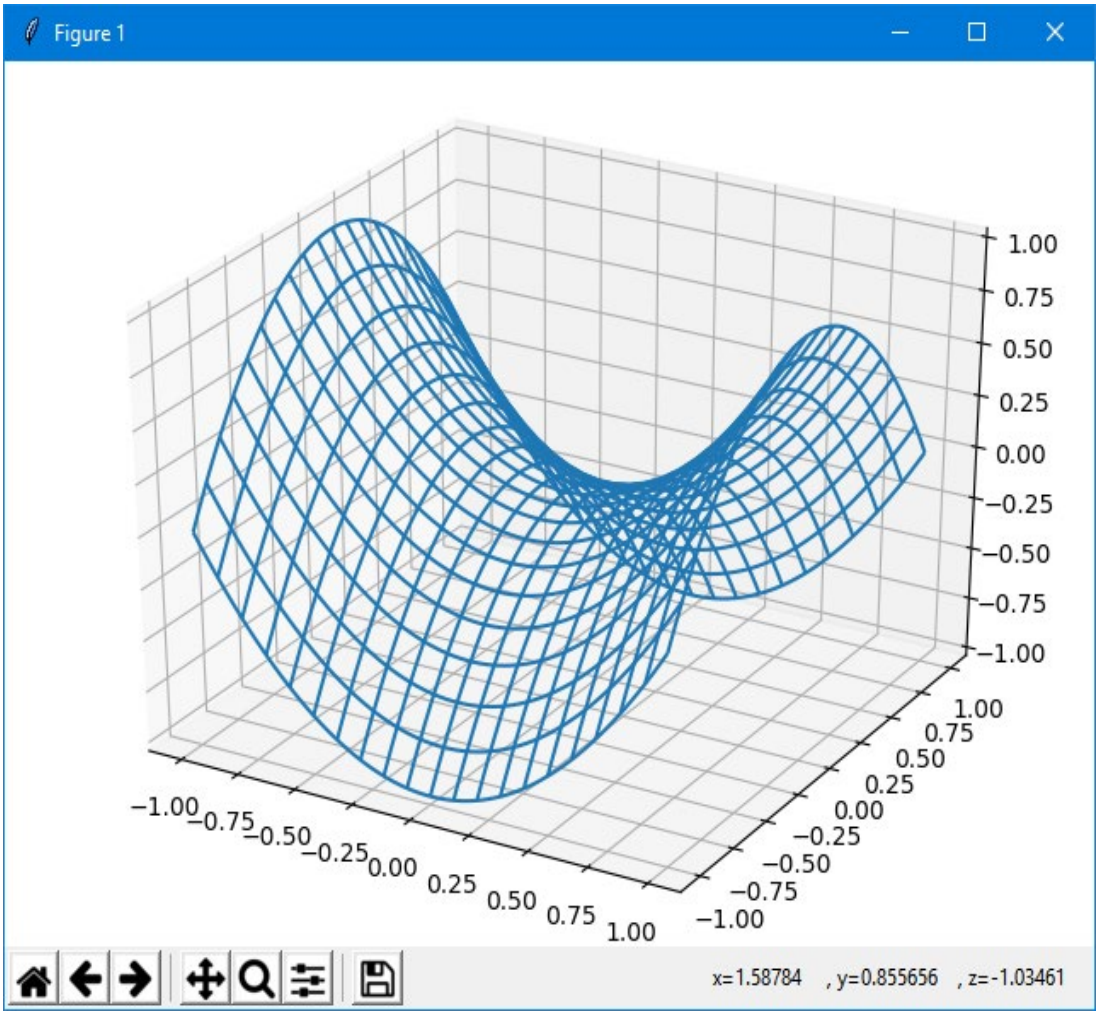
```
import pylab
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import inearSegmentedColormap
from matplotlib import cm
import numpy as np

t=np.linspace(0,2*np.pi,50)
th,ph=np.meshgrid(t,t)
r=0.2
```

```
x,y,z=(1+r*np.cos(ph))*np.cos(th),(1+r*np.cos(ph))*
                                  np.sin(th),r*np.sin(ph)
fig=pylab.figure()
ax=Axes3D(fig)
ax.elev=60
ax.set_aspect(0.3)
ax.plot_surface(x,y,z,rstride=2,cstride=1)
pylab.show()
```



П24. Построение графика «сетки» функции двух переменных

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
ax = axes3d.Axes3D(plt.figure())
i = np.arange(-1, 1, 0.01)
X, Y = np.meshgrid(i, i)
Z = X**2-Y**2
ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)
plt.show()
```



Для примера построит три плоскости паралельные координатным плоскостям

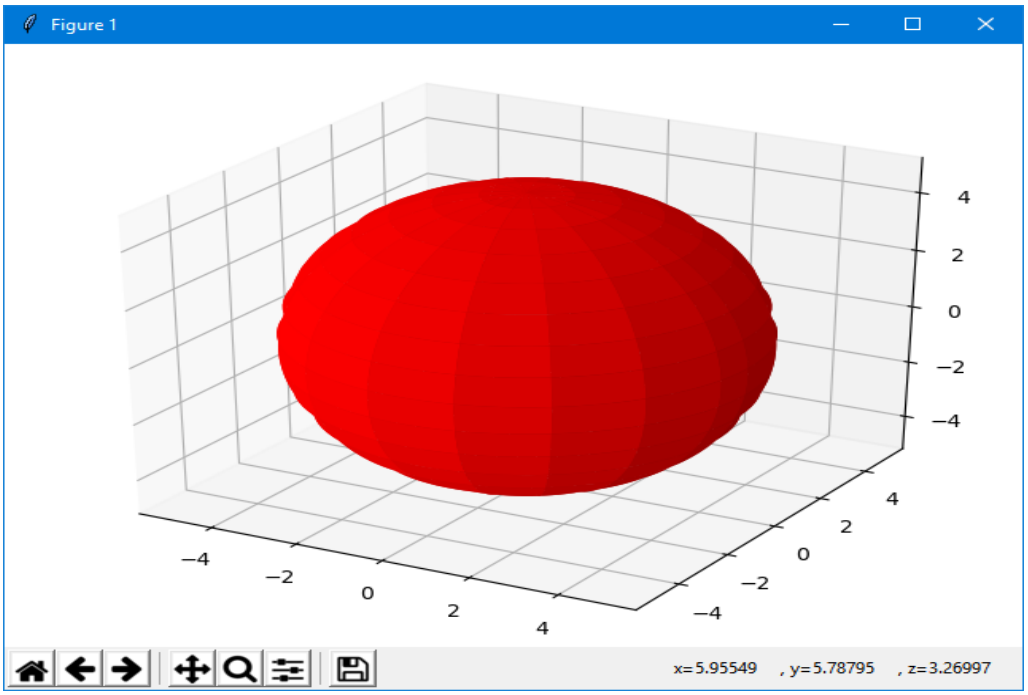
```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
axes = axes3d.Axes3D(fig)
t = np.arange(-1, 1, 0.01)
x, y = np.meshgrid(t, t)
axes.plot_wireframe(0*x, y, x, rstride=5, cstride=5,
                                        color='r')
axes.plot_wireframe(x, 0*y, y, rstride=5, cstride=5,
                                       color='b' )
axes.plot_wireframe(x, y, 0*x, rstride=5, cstride=5,
                                       color='g')
plt.show()
```

П25 Построение графика функции двух переменных, заданной параметрически $x=x(u,v)$, , $y=y(u,v)$, $z=z(u,v)$.

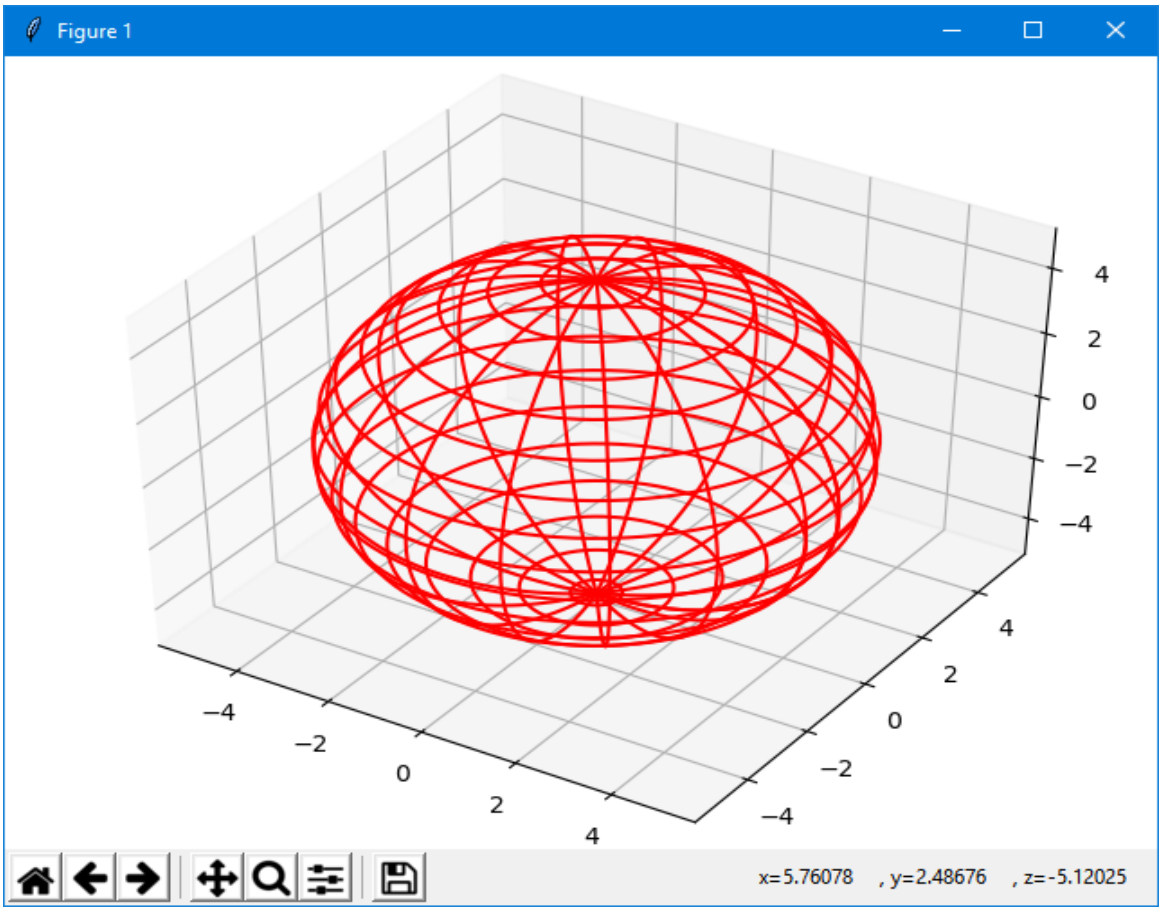
Поверхность задается параметрические: $x=x(u,v)$, , $y=y(u,v)$, $z=z(u,v)$.
Классическое уравнение сферы, заданное параметрически:

$$x(u,v)=\cos(u)\cdot\cos(v)$$
$$y(u,v)=\sin(u)\cdot\cos(v)$$
$$z(u,v)=\sin(v)$$
$$u\in[-\pi,+\pi]$$
$$v\in[-2\pi,+2\pi]$$

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
ax = axes3d.Axes3D(plt.figure())
u = np.linspace(0, 2*np.pi, 100)
v = np.linspace(0, np.pi, 100)
x =5*np.outer(np.cos(u), np.sin(v))
y =5*np.outer(np.sin(u), np.sin(v))
z =5*np.outer(np.ones(np.size(u)), np.cos(v))
ax.plot_surface(x,  y,  z,    rstride=6,  cstride=6,
                                        color='r')
plt.show()
```



Заменив plot_surface на plot_wireframe получим:



В программе использовалась функция модуля ***numpy.outer***, которая вычисляет внешнее (прямое или тензорное) произведение двух векторов:

numpy.outer(a, b, out=None)

Для двух векторов *a* длинной *N* и *b* длинной *M* внешнее произведение определяется следующим правилом:

```
[[a0*b0, a0*b1, ..., a0*bN],
 [a1*b0, a1*b1, ..., a1*bN],
 ...
 [aM*b1, aM*b1, ..., aM*bN]]
```

Данное правило может быть обобщено на массивы с большей размерностью. Но данная функция сжимает все многомерные массивы до одной оси.

Параметры: **a, b** - числа, массивы NumPy или подобные массивам объекты. Одномерные массивы, необязательно одинаковой длины. Двумерные и многомерные массивы сжимаются до одной оси. **out** - массив NumPy, необязательный параметр.

Массив в который можно поместить результат функции. Данный массив должен соответствовать форме и типу данных результирующего массива функции, а так же обязательно быть *C*-смежным, т.е. хранить данные в строчном *C* стиле. Указание данного параметра, позволяет избежать лишней

операции присваивания тем самым немного ускоряя работу вашего кода. Полезный параметр если вы очень часто обращаетесь к функции в цикле.

Возвращает: результат - двумерный массив NumPy являющимся внешним произведением двух векторов..

Функция **`ones()`** возвращает новый массив указанной формы и типа, заполненный единицами. А **`np.ones(np.size(u))`** - возвращает новый массив, размера **`u`** так же заполненный единицами.

В выше приведенном примере строилась так называемая «плотная» сетка 3-мерного координатного пространства на значениях сетки координат x, y, z соответственно. Для этого использовалалась функция прямого или тензорного произведение двух векторов значений u и v.

Можно поступить «по другому» - построить массив плотных координатных сеток 3-мерного координатного пространства для указанных в виде диапазонов одномерных массивов координатных векторов u и v. Затем «просто» применить формулы, задающие требуемые зависимости ***x=x(u,v), , y=y(u,v), z=z(u,v).***

Для построения массива плотных координатных сеток используем функцию **`numpy.mgrid:`**

```
numpy.mgrid[index                object]                =
     <numpy.lib.index_tricks.nd_grid object>
```

Функция **mgrid()** возвращает массив плотных координатных сеток N-мерного координатного пространства для указанных в виде диапазонов одномерных массивов координатных векторов.

Параметры: index object - объект индексации

Под объектом индексации понимается список из двух или трех элементов, например *[0:5]* или *[0:5:10j]*.

Если элементов в списке всего два, то это интерпретируется как полуоткрытый интервал *[start, ... , stop)*, в котором все элементы отличаются на 1, а значение *stop* в сам интервал не входит.

В качестве третьего элемента указывается мнимая часть комплексного числа, которое указывает на количество равномерно разнесенных элементов внутри закрытого интервала *[start, ... , stop]*, при этом значение *stop* попадает в интервал.

Возвращает: результат - массив NumPy, являющимся массивом плотных координатных сеток N-мерного координатного пространства, количество и размеры которых зависят от указанных диапазонов.

Построение выполним следующим скриптом:

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
```

```
axes = axes3d.Axes3D(fig)
u, v = np.mgrid[0:2*np.pi:100j, 0:2*np.pi:100j]
x = 5*np.cos(u)*np.sin(v)
y = 5*np.sin(u)*np.sin(v)
z = 5*np.cos(v)
#axes.plot_surface(x,  y,  z,rstride=6,  cstride=6,
                                          color='r')
axes.plot_wireframe(x,  y,  z,  rstride=5,  cstride=5,
                                          color='r')
plt.show()
```

Для получения массива плотных координатных сеток можно так же воспользоваться (рассмотренной выше) функцией **np.meshgrid:**

```
u1=np.linspace(0,2*np.pi,100)
v1=np.linspace(0,2*np.pi,100)
u, v = np.meshgrid(u1,v1)
```

При необходимости можно весь рисунок «растянуть» в kg раз по горизонтали и в kv раз по вертикали. Для этого укажем параметр метода `plt.figure()`:

```
kg=2; kv=1
fig = plt.figure(figsize=plt.figaspect(1/kg)*kv)
```

П26. «Скроллинг» по оси X

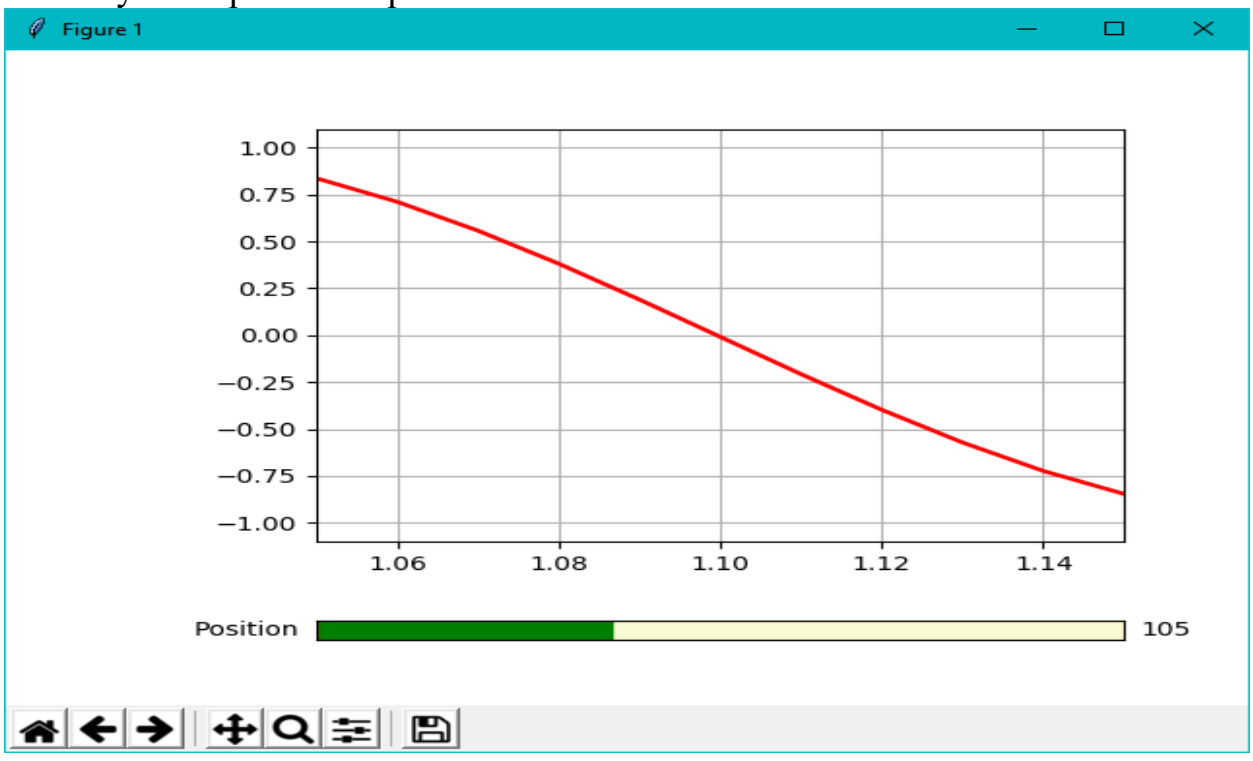
Matplotlib включает в себя очень небольшое количество элементов управления, которые в терминах Matplotlib называются виджетами, которые располагаются пакете *matplotlib.widgets*. В этом пакете содержатся также виджеты для взаимодействия с графиками (виджеты для выделения областей) – подробнее см., например, http://jenyay.net/Matplotlib/Widgets.

Используя виджет `Slider` можно «растянуть» ось X:

```
from matplotlib.widgets import Slider
import math
import matplotlib.pyplot as plt
# подробнее см. например
# http://jenyay.net/Matplotlib/Widgets
fig, ax = plt.subplots()
plt.subplots_adjust(left=0.25, bottom=0.25)
t = [i / 100. for i in range(0, int(math.pi) * 100, 1)]
s = [math.sin(i * 20) for i in t]
l, = plt.plot(t, s, lw=2, color='red')

plt.axis([0, 1, -10, 10])
plt.grid(True)
```

```
axcolor = 'lightgoldenrodyellow' #'gray'
ax_x_pos = plt.axes([0.25, 0.1, 0.65, 0.03],
facecolor=axcolor )
wsize = 10
x_pos = Slider(ax_x_pos, 'Position', 0,
    l            en(t) - wsize - 1, valfmt='%d', \
                                valinit=0,  color="g")

ax.set_xlim(t[0], t[wsize])
ax.set_ylim(-1.1, 1.1)

def update(val):
    #print(x_pos.val)
    pos = int(x_pos.val)
    ax.set_xlim(t[pos], t[pos + wsize])
    fig.canvas.draw_idle()

x_pos.on_changed(update)

plt.show()
```

Результат работы скрипта:



П27. «Динамические» графики

Без комментариев приведем пример «динамического» построения графика

см https://matplotlib.org/2.0.0/examples/animation/animate_decay.html:

```
'''
This example showcases a sinusoidal decay animation.
'''
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

def data_gen(t=0):
    cnt = 0
    while cnt < 1000:
        cnt += 1
        t += 0.1
        yield t, np.sin(2*np.pi*t) * np.exp(-t/10.)

def init():
    ax.set_ylim(-1.1, 1.1)
    ax.set_xlim(0, 10)
    del xdata[:]
    del ydata[:]
    line.set_data(xdata, ydata)
    return line,

fig, ax = plt.subplots()
line, = ax.plot([], [], lw=2)
ax.grid()
xdata, ydata = [], []

def run(data):
    # update the data
    t, y = data
    xdata.append(t)
    ydata.append(y)
    xmin, xmax = ax.get_xlim()

    if t >= xmax:
        ax.set_xlim(xmin, 2*xmax)
        ax.figure.canvas.draw()
    line.set_data(xdata, ydata)
    return line,

ani = animation.FuncAnimation(fig,
            run, data_gen, blit=False, interval=10,
                  repeat=False, init_func=init)
plt.show()
```

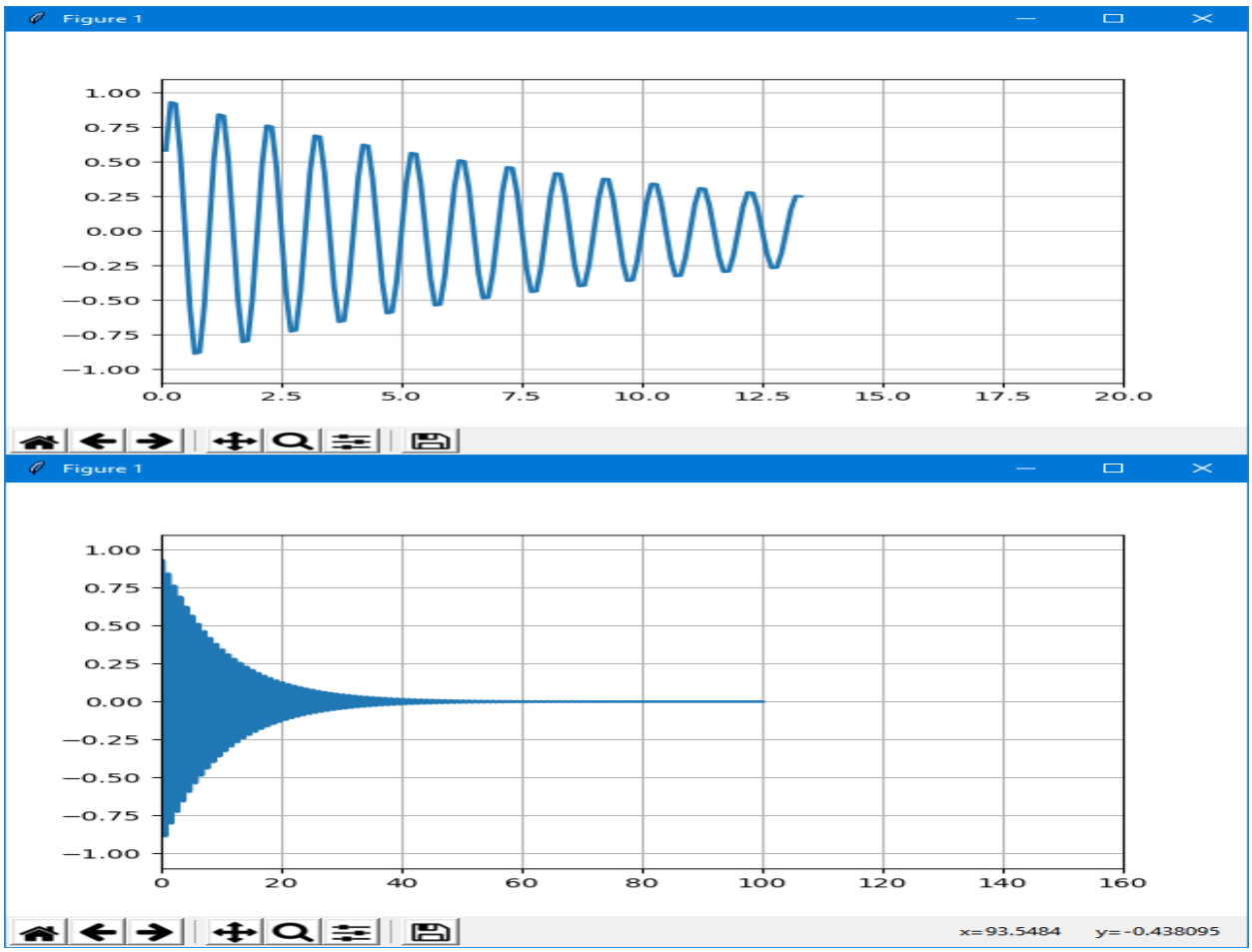


Figure 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Figure 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100
120
140
160
x=93.5484
y=-0.438095