Тема 6. 3D графіка.

П20. Тривимірний графік

Усі класи для роботи з тривимірними графіками знаходяться у пакеті mpl_toolkits.mplot3d, з якого потрібно їх імпортувати.

Для того щоб намалювати тривимірний графік, в першу чергу треба створити тривимірні осі.

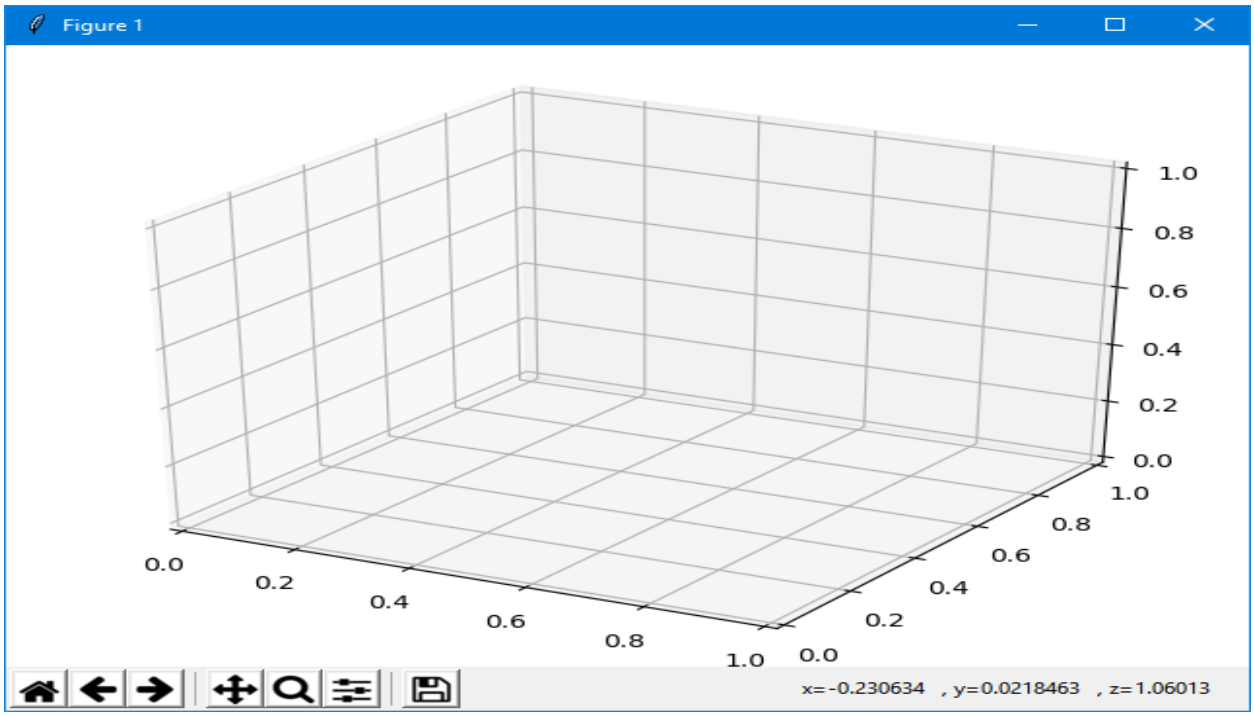
Щоб створити їх, потрібно створити екземпляр класу mpl_toolkits.mplot3d.Axes3D. Його конструктор чекає, як мінімум, один параметр – екземпляр класу matplotlib.figure.Figure. Цей об'єкт створюється pylab.figure(). Конструктор класу matplotlib.figure.Figure має ще й інші необов'язкові параметри (за матеріалами сайту http://jenyay.net/Programming/Python3d).

Намалюємо порожні осі:

```
importnumpy as np
importpylab
frommpl_toolkits.mplot3d import Axes3D

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

В результаті побачимо наступне вікно:



Отримані осі можна обертати мишкою.

П21. Тривимірна лінія

Лінія в просторі задається параметрично: x = x (t), y = y (t), z = z (t). Наприклад 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 побудови та візуалізації використовується об'єкт класу Axes3D з пакету mpl_toolkits.mplot3d:

```
importpylab
importmpl_toolkits.mplot3d as A3D
```

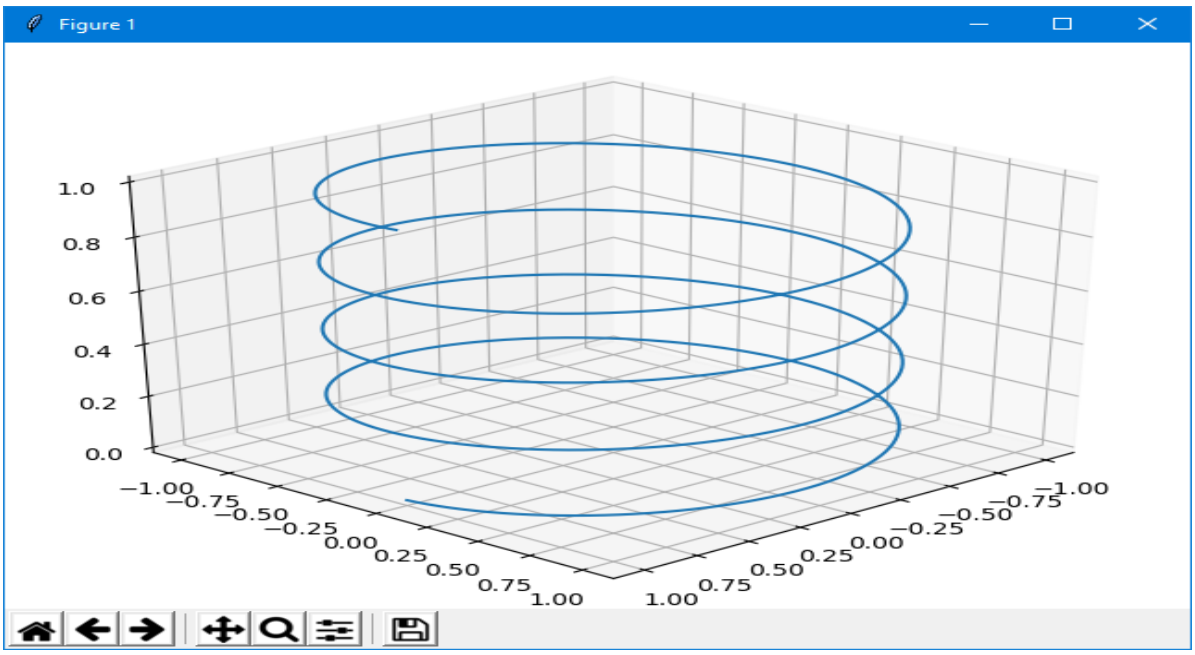
figure() - це поточний малюнок, створюємо в ньому об'єкт ax, потім використовуємо його методи для візуалізації

```
importpylab
importnumpy as np
importmatplotlib as mpl
importmatplotlib.pyplot as plt
##from mpl_toolkits.mplot3d import Axes3D
importmpl_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 потрібен об'єкт класу Axes3D з пакету
mpl_toolkits.mplot3d.
# figure() - це поточний малюнок, створюємо в ньому
об'єкт ax,
потім використовуємо його методи.
fig=pylab.figure()
ax=A3D.Axes3D(fig)
ax.elev,ax.azim=30,45 # задати, з якого боку дивимося.

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



Побудуємо три прямі по осях координат:

```
від 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()
```

Побудуємо лінію, задану рівняннями:

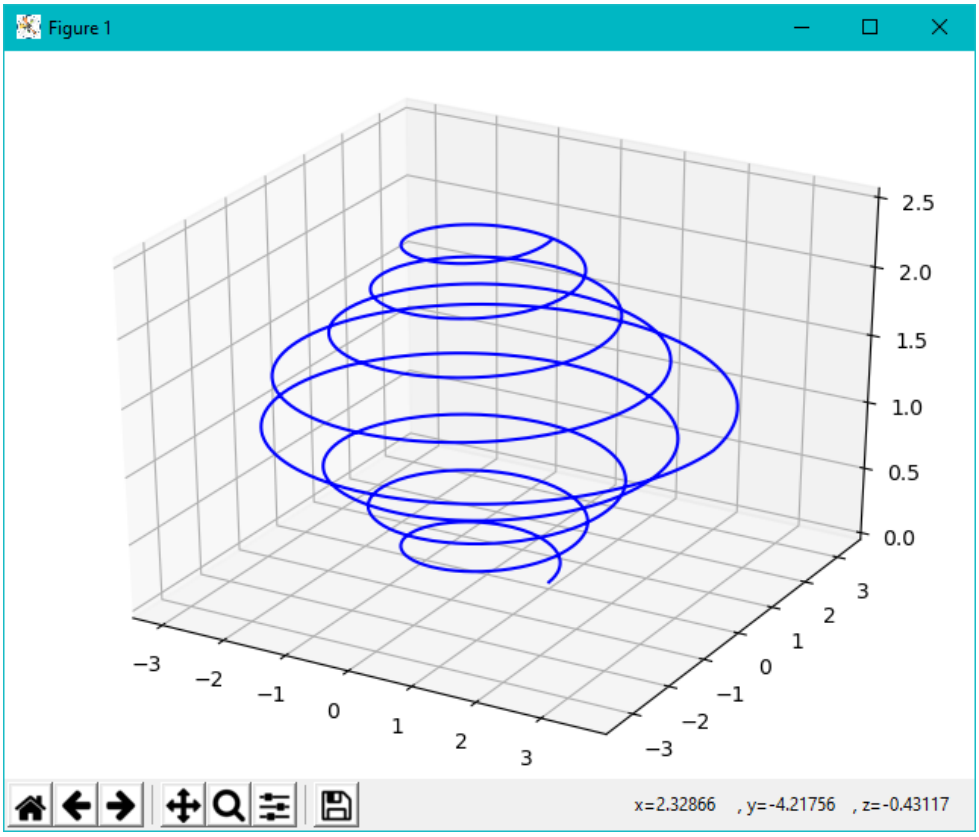
$x(t) = \cos(2 \cdot t) \cdot e^{\frac{a \cdot t}{10}}$ $y(t) = \sin(2 \cdot t) \cdot e^{\frac{a \cdot t}{10}}$ $z(t) = \frac{t}{10}$

$t \in [0, 8 \cdot \pi]$ $a = \{-1, \text{при } t \in [0, 4 \cdot \pi)|$

```
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. Поверхні

Усі поверхні задаються параметричні: x = x (u, v), y = y (u, v), z = z (u, v).

Якщо хочемо задати поверхню «явно» z=z(x,y), то зручно створити масиви x=u і y=v функцією meshgrid.

Для всіх прикладів будемо використовувати наступну функцію від двох координат.

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

Спочатку потрібно підготувати дані для малювання. Нам знадобляться три двомірні матриці:

- матриці X і Y зберігатимуть координати сітки точок, в яких обчислюватиметься наведена вище функція,
- матриця Z зберігатиме значення цієї функції у відповідній точці.

Якщо ми хочемо намалювати тривимірний графік на еквідистантній сітці (на сітці, яка має відстань між точками однакова), то для створення матриць, які зберігатимуть координати, будемо використовувати функцію meshgrid() з бібліотеки numpy.

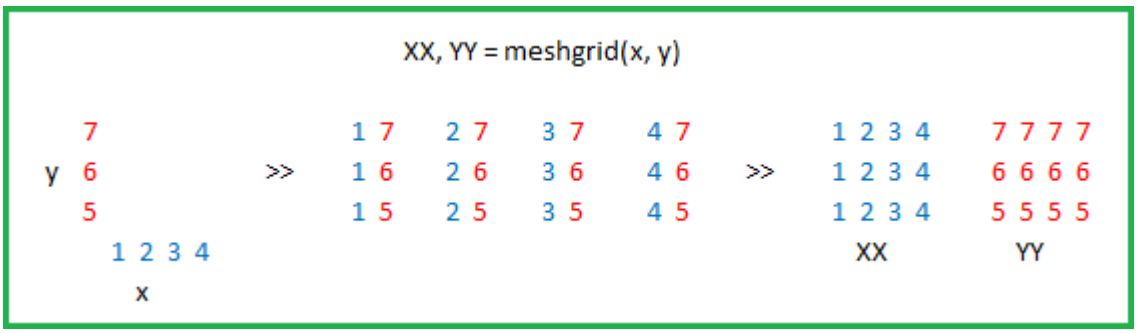
Ця функція створює двовимірні матриці сіток за одновимірними масивами. Робота цієї функції дуже наочно показана нижче:

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

>>> Yarray([[4, 4, 4], [5, 5, 5], [6, 6, 6], [7, 7,
7]])
```

Тепер індексу вузла сітки можна дізнатися реальні координати: X[0][0] = 1, Y[0][0] = 4 тощо.

Фактично функція [X, Y] = meshgrid(x, y) задає сітку на площині xy у вигляді двовимірних масивів X, Y, які визначаються одновимірними масивами x та y. Рядки масиву X є копіями вектора x, а стовпці - копіями вектора y. Формування таких масивів полегшує обчислення функцій двох змінних, дозволяючи застосовувати операції над масивами. Нижче наведемо малюнок, що пояснює побудову масивів X і Y:



Щоб відокремити підготовку даних від самого малювання, створення сітки та розрахунок функції виділимо в окрему функцію:

```
defmakeData (): # Будуємо сітку в інтервалі від -10 до
10 # з кроком 0.1 по обох координатах
    x = numpy.arange (-10, 10, 0.1)
    y = numpy.arange (-10, 10, 0.1)
    # Створюємо двомірну матрицю-сітку
    xgrid, ygrid = numpy.meshgrid(x, y)
    # У вузлах розраховуємо значення функції
    zgrid = numpy.sin (xgrid) * numpy.sin (ygrid) /
                                          (xgrid * ygrid)
    returnxgrid, ygrid, zgrid
```

Ця функція повертає три двовимірні матриці: x, y, z. Координати x та y лежать в інтервалі від -10 до 10 з кроком 0.1.

Тепер повертаємось безпосередньо до малювання. Щоб відобразити наші дані, достатньо викликати метод plot_surface() екземпляра класу Axes3D, який передамо отримані за допомогою функції makeData() двомірні матриці.

Тепер наш приклад виглядає так:

```
importpylabfrom mpl_toolkits.mplot3d import
Axes3Dimport numpydef 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)
    returnxgrid, ygrid, zgrid

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

У нових версіях matplotlib замість

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

рекомендується кодувати:

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

Так, наприклад, можна "підписати" осі координат:

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

Якщо ми запустимо цей скрипт, то з'явиться вікно:

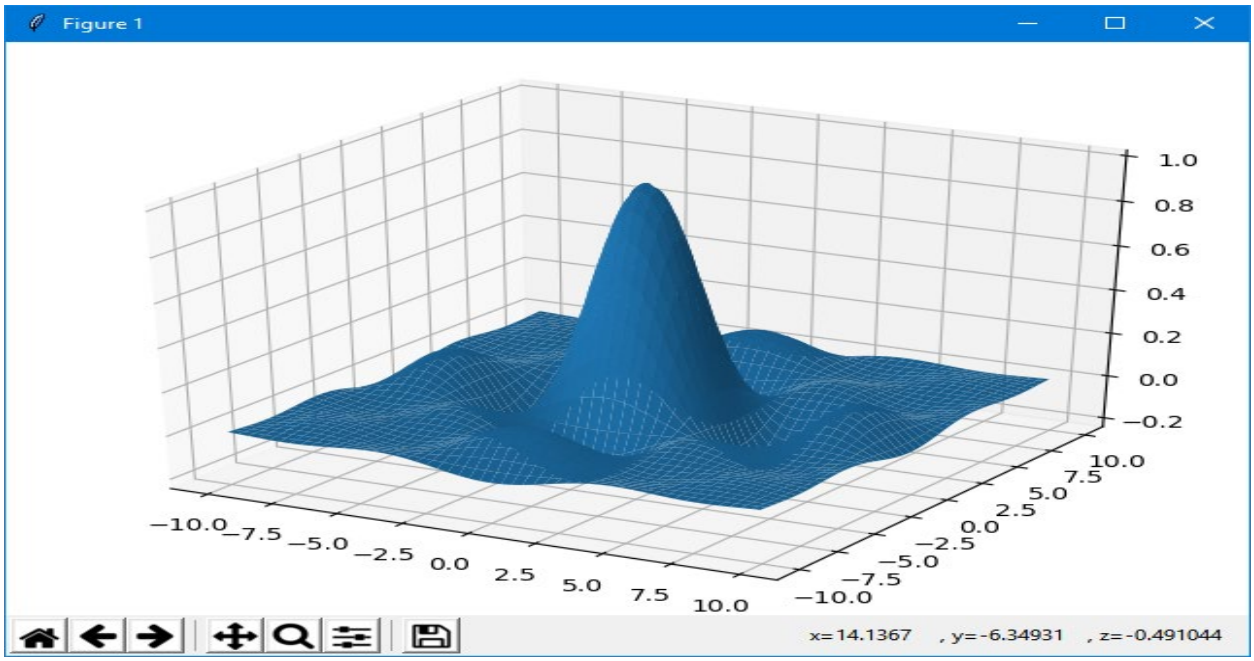


Рисунок можна обертати за допомогою мишки. Matplotlib не використовує графічний прискорювач, тому обертання відбувається досить повільно, хоча швидкість залежить кількості точок на поверхні.

Розглянемо інші параметри методу Axes3D.plot_surface(X, Y, Z, args, kwargs), їх не так багато:

- *X*, Y та Z - ці параметри задають сітку та значення функції у вузлах.
- *rstride* та cstride задають крок виведення графіка. Чим менший крок, тим точніше відображається функція, але довше відбувається малювання.
- *color* - задає колір графіка
- *cmap* - задає градієнт кольорів, щоб колір комірки графіка залежав від значення функції у цій галузі.

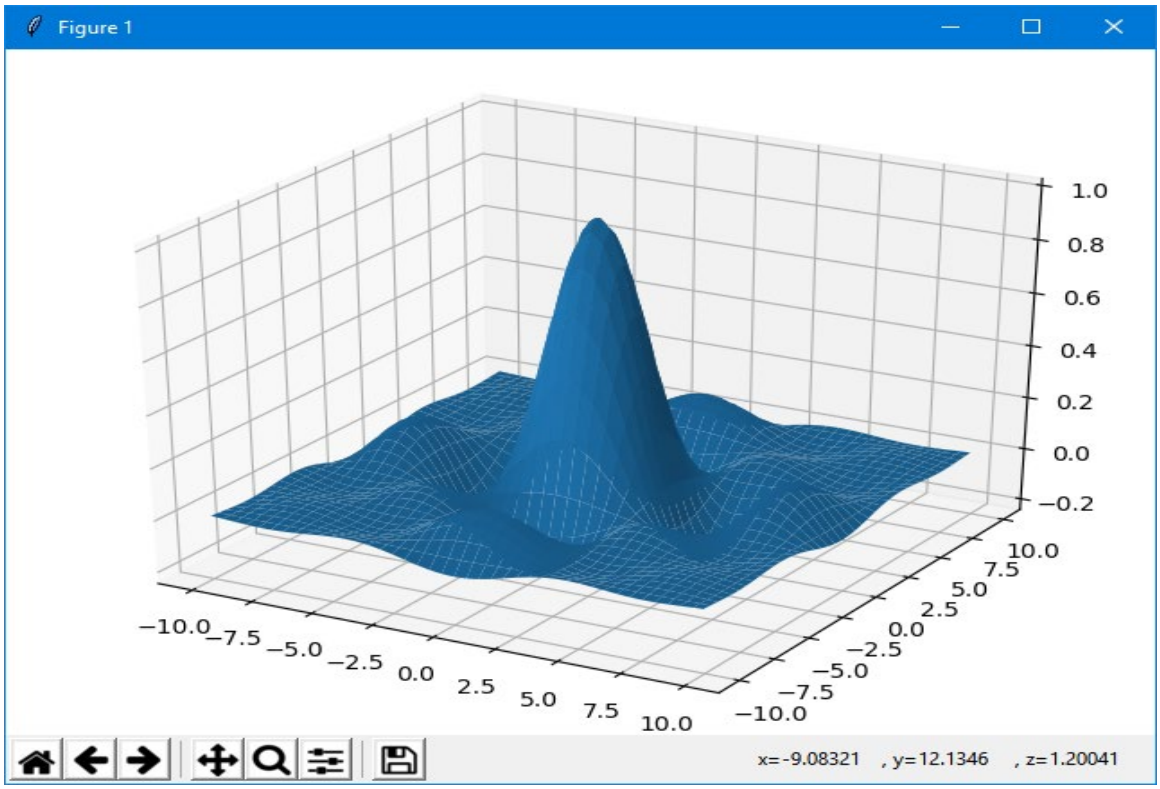
Розглянемо ці параметри.

Крок сітки

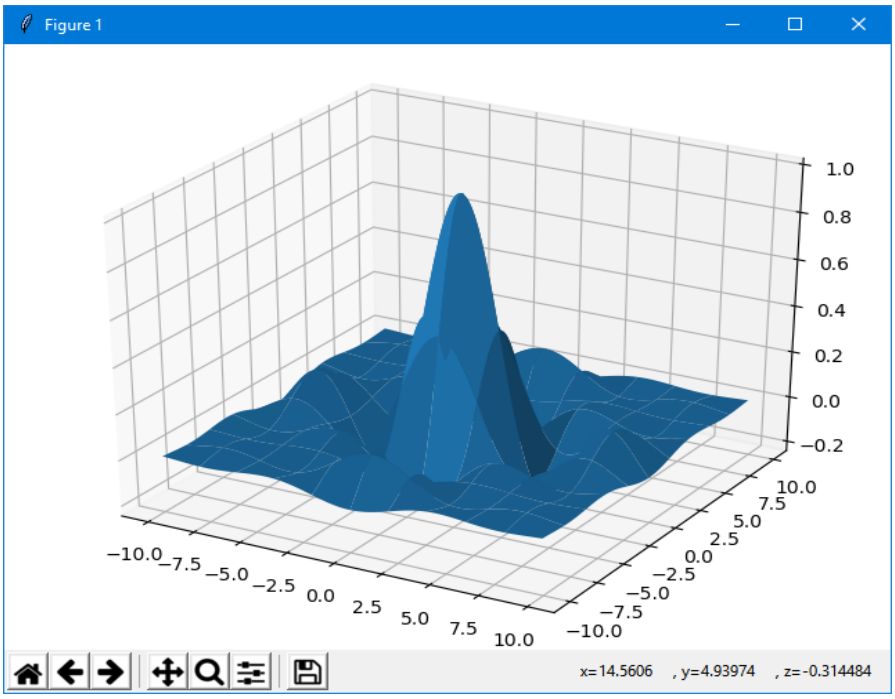
Спочатку подивимося як впливають зовнішній вигляд параметри rstride і cstride. Змінимо у попередньому прикладі рядок з використанням методу plot_surface() наступним чином:

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

В результаті ми отримаємо дрібнішу сітку на графіку:



Якщо так само встановити значення цих параметрів в 20, то сітка буде навпаки більша:



Зміна кольору

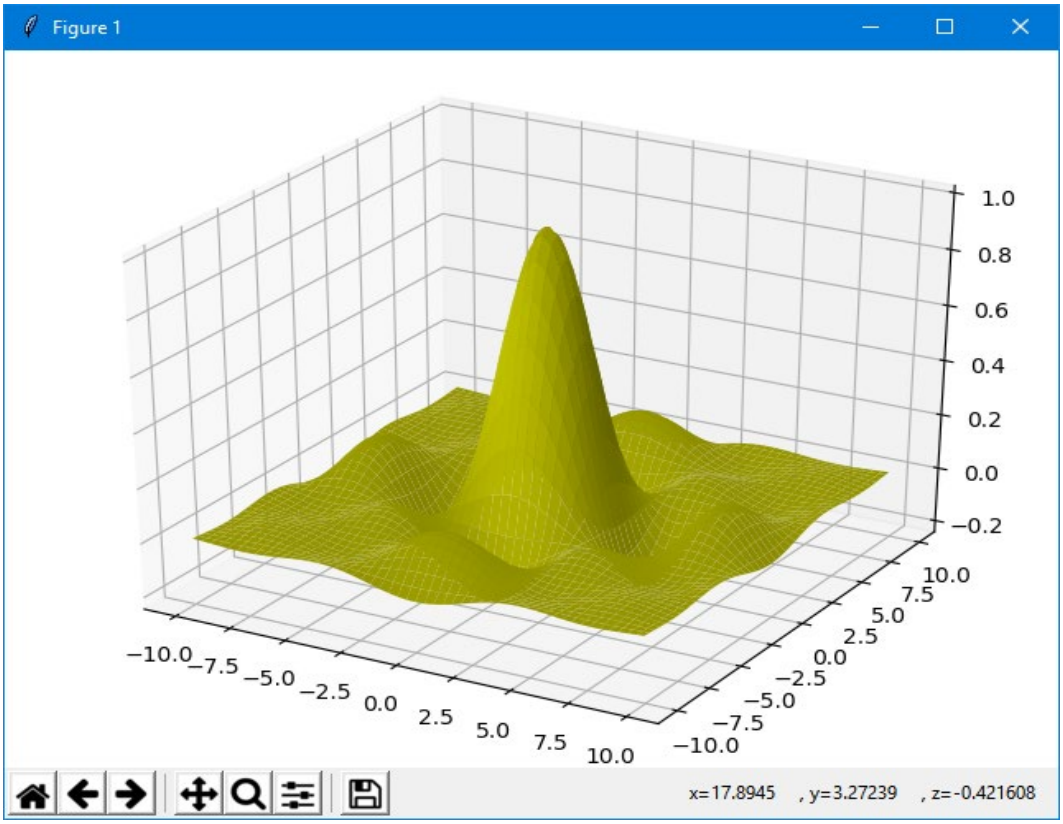
Тепер змінимо колір поверхні за допомогою параметра color. Цей параметр є рядком, який описує колір. Рядок кольору може задаватися різними способами:

Колір можна визначити англійським словом для відповідного кольору або однією літерою. Таких квітів небагато:

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

Для прикладу зробимо поверхню жовтою:

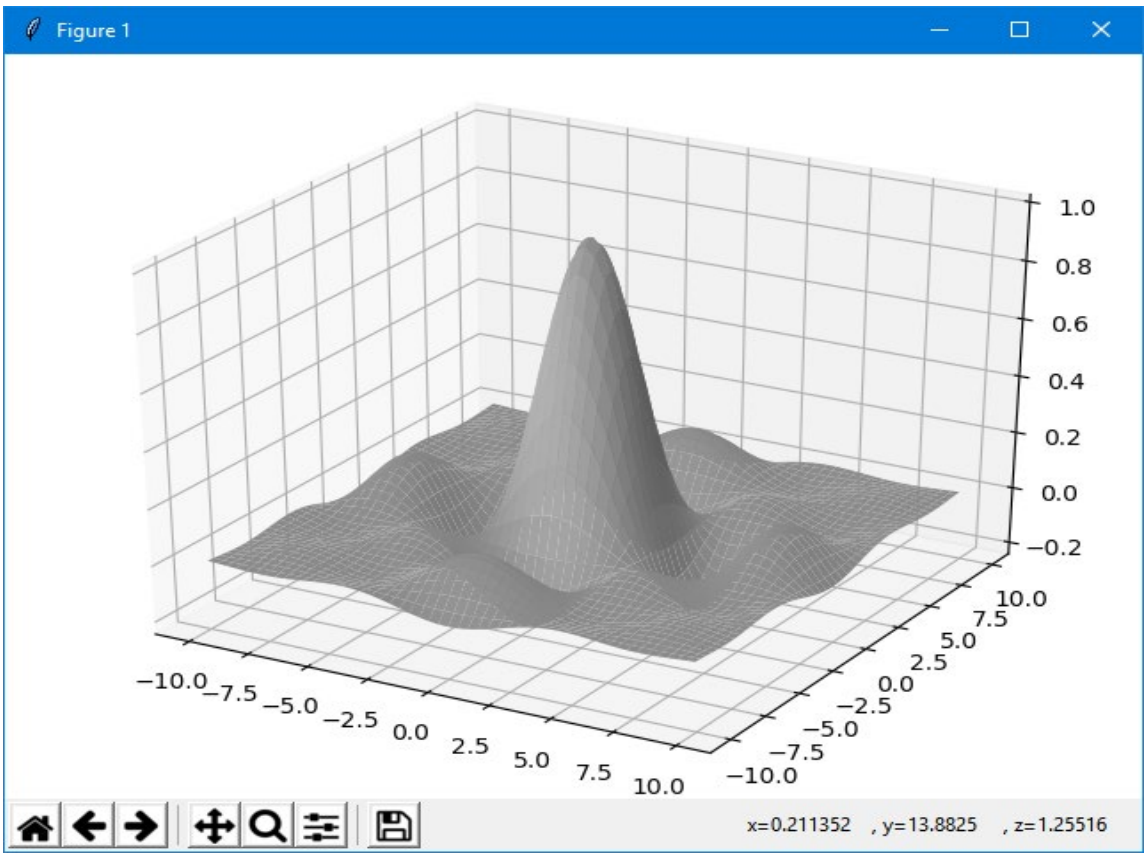
```
axes.plot_surface(x, y, z, color='yellow')
   або
axes.plot_surface(x, y, z, color='y')
```



Якщо нам потрібен сірий колір, його яскравість можемо задати за допомогою рядка, що містить число в інтервалі від 0.0 до 1.0 (0 - білий, 1 - чорний). Наприклад, можна написати наступний рядок:

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

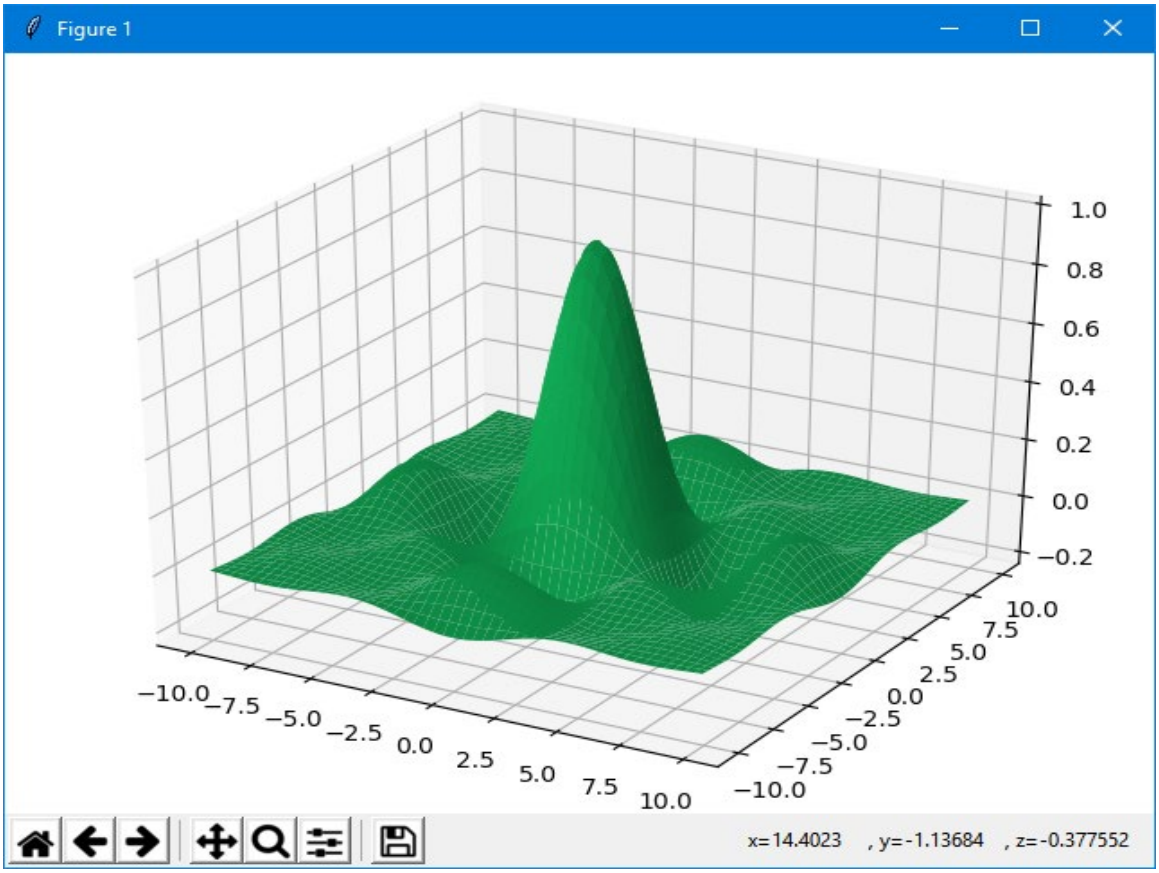
У цьому випадку ми побачимо такий ось сірий графік:



Крім того, ми можемо задавати колір, оскільки це прийнято в HTML після символу решітки ('#'). Наприклад, можемо задати колір так:

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

Тоді графік позеленіє:



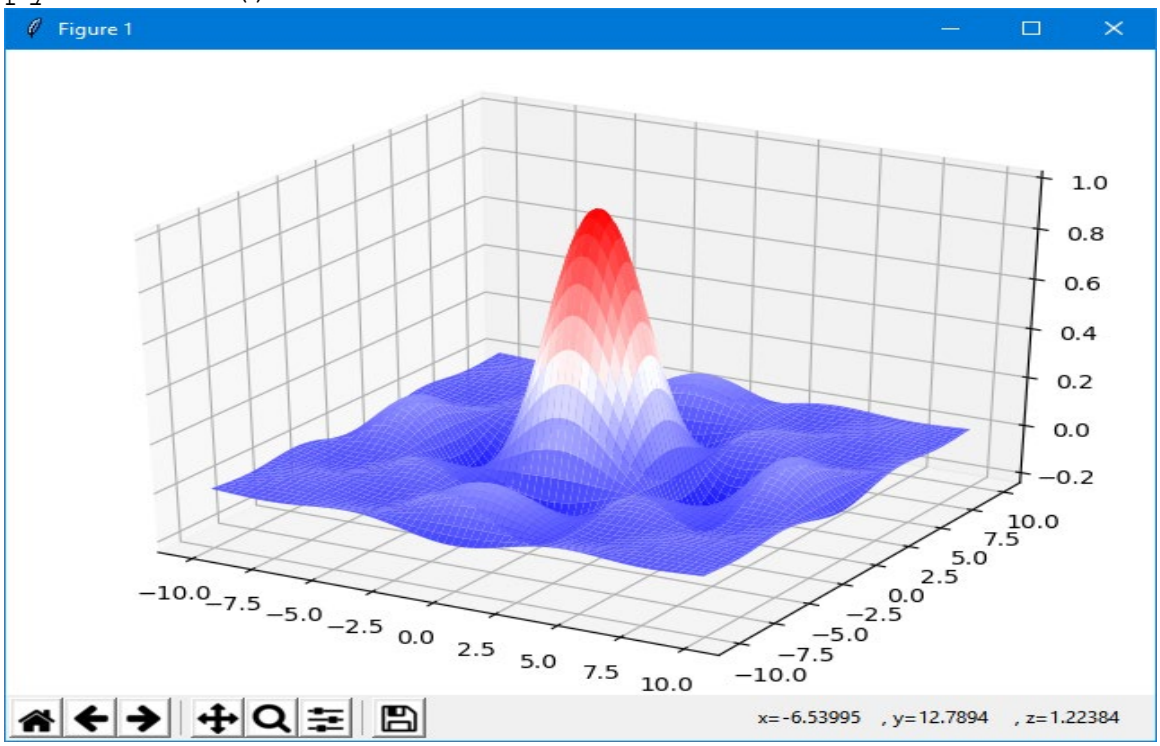
Використання колірних карток (colormap)

Кольорові карти використовуються, якщо потрібно вказати в які кольори мають фарбуватись ділянки тривимірної поверхні залежно від значення Z у цій галузі (завдання кольорового градієнта).

Щоб при виведенні графіка використовувався градієнт, як значення параметра cmap (від слова colormap, колірна карта) потрібно передати екземпляр класу matplotlib.colors.Colormap або похідного від нього.

Наступний приклад використовує клас LinearSegmentedColormap, похідний від Colormap, щоб створити градієнт переходу від синього кольору до червоного через білий.

```
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()
```

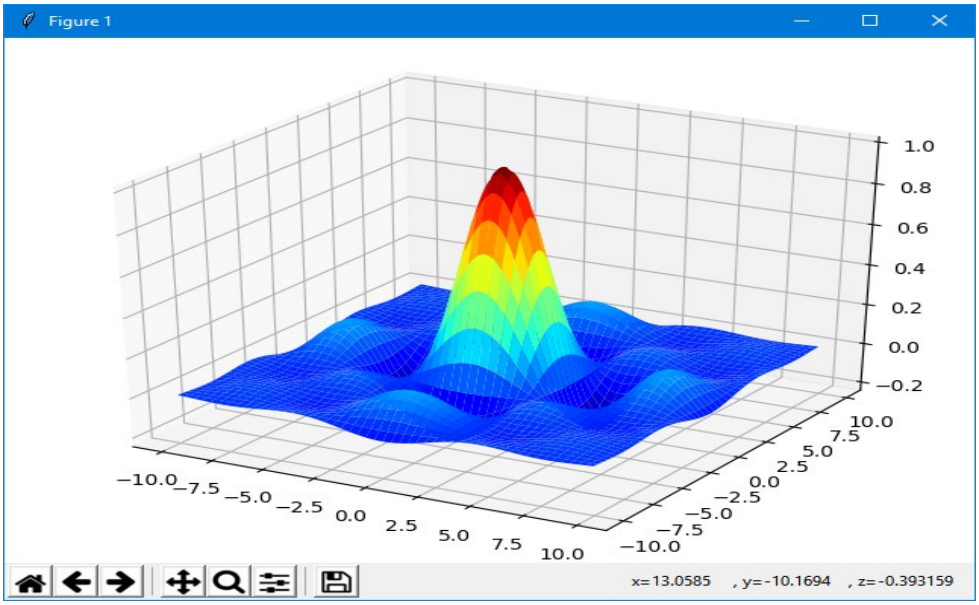


Тут використовується статичний метод from_list(), який приймає три параметри:

- Ім'я створюваної картки
- Список кольорів, починаючи з кольору для мінімального значення на графіці (блакитний – 'b'), через проміжні кольори (у нас це білий – 'w') до кольору для максимального значення функції (червоний – 'r').
- Кількість перехідів кольорів. Чим це число більше, тим паче плавний градієнт, але тим більше пам'яті він займає.

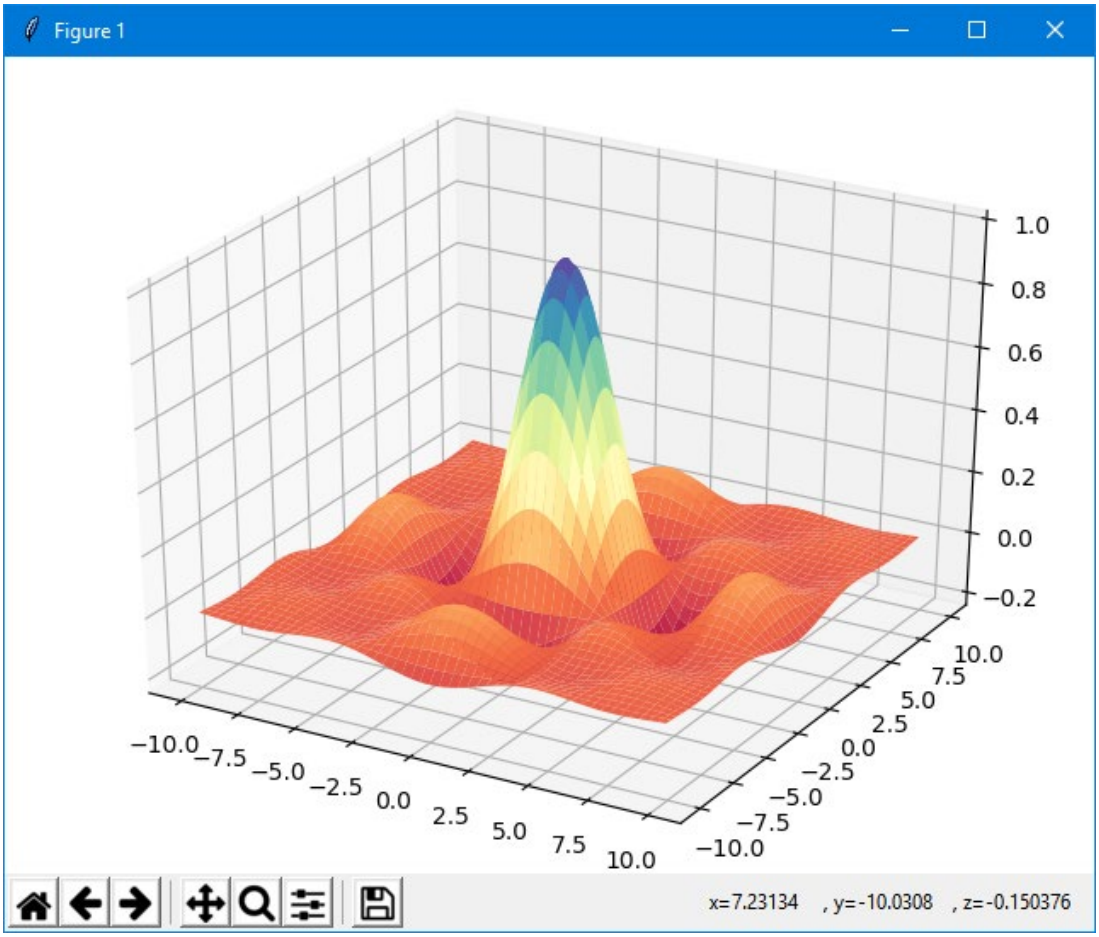
Карта cm.jet - це, напевно, найпоширеніша карта в прикладах.

```
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()
```



Зауважте, що як аргумент cmap ми передаємо не рядок, а ім'я змінної з модуля cm, причому це вже створений екземпляр класу, а не ім'я класу. Так, наприклад, cm.jet - це екземпляр класу matplotlib.colors.LinearSegmentedColormap.

Для прикладу колірна карта cm.Spectral виглядає так:



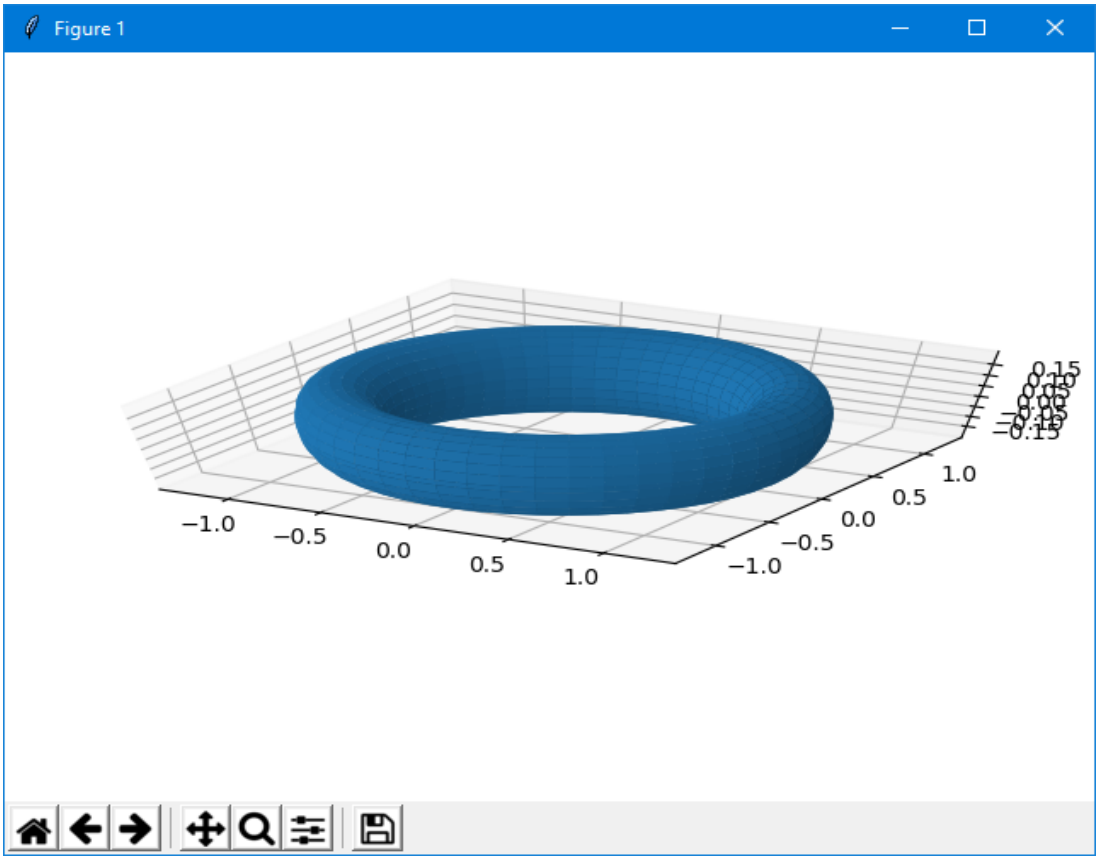
П23. Параметричні поверхні з параметрами ϑ φ

```
importpylab
frommpl_toolkits.mplot3d import Axes3D
frommatplotlib.colors import inearSegmentedColormap
frommatplotlib import cm
importnumpy as np

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

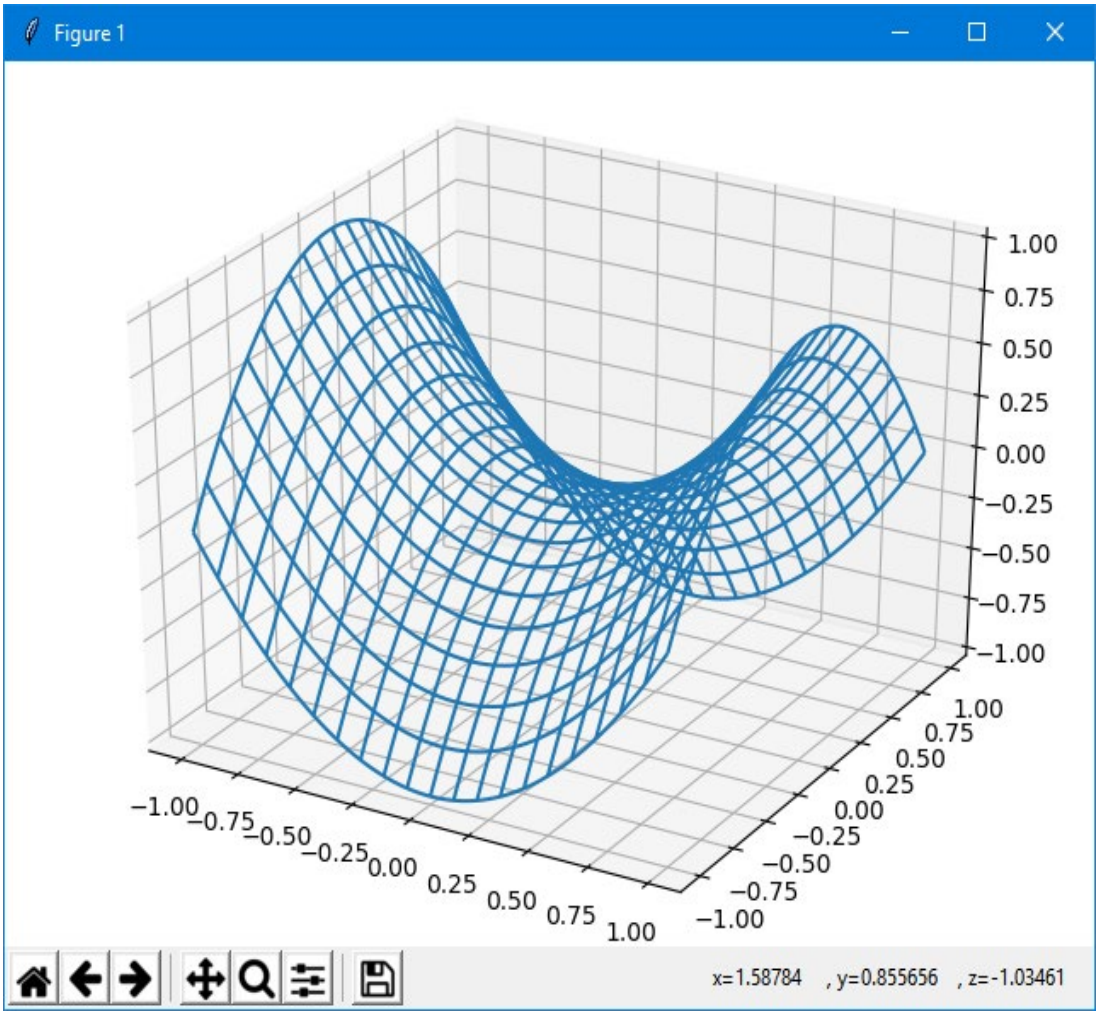
```
   x,y,z=(1+r*np.cos(ph))*np.cos(th),(1+r*np.cos(ph))*
np.sin(th),r*np.sin(ph)

   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. Побудова графіка «сітки» функції двох змінних

```
   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 прикладу збудує три площини паралельні координатним площинам.

```
від 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 Побудова графіка функції двох змінних, заданої параметрично x = x (u, v), y = y (u, v), z = z (u, v).

Поверхня задається параметричні: x = x (u, v), y = y (u, v), z = z (u, v).

Класичне рівняння сфери, задане параметрично:

$x(u, v) = \cos(u) \cdot \cos(v)$

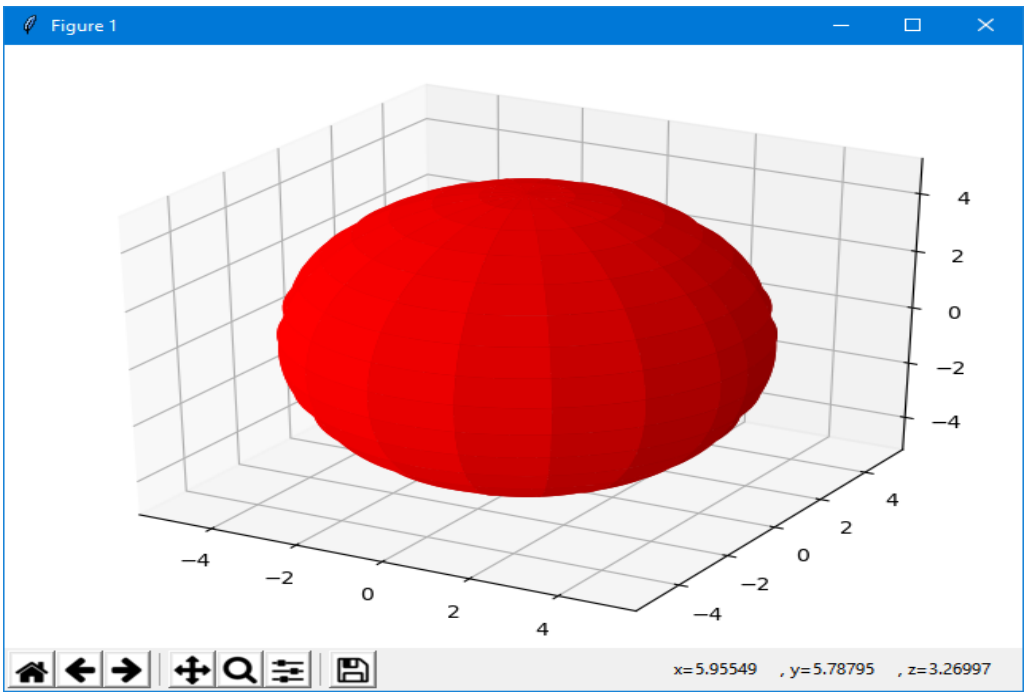
$y(u, v) = \sin(u) \cdot \cos(v)$

$z(u, v) = \sin(v)$

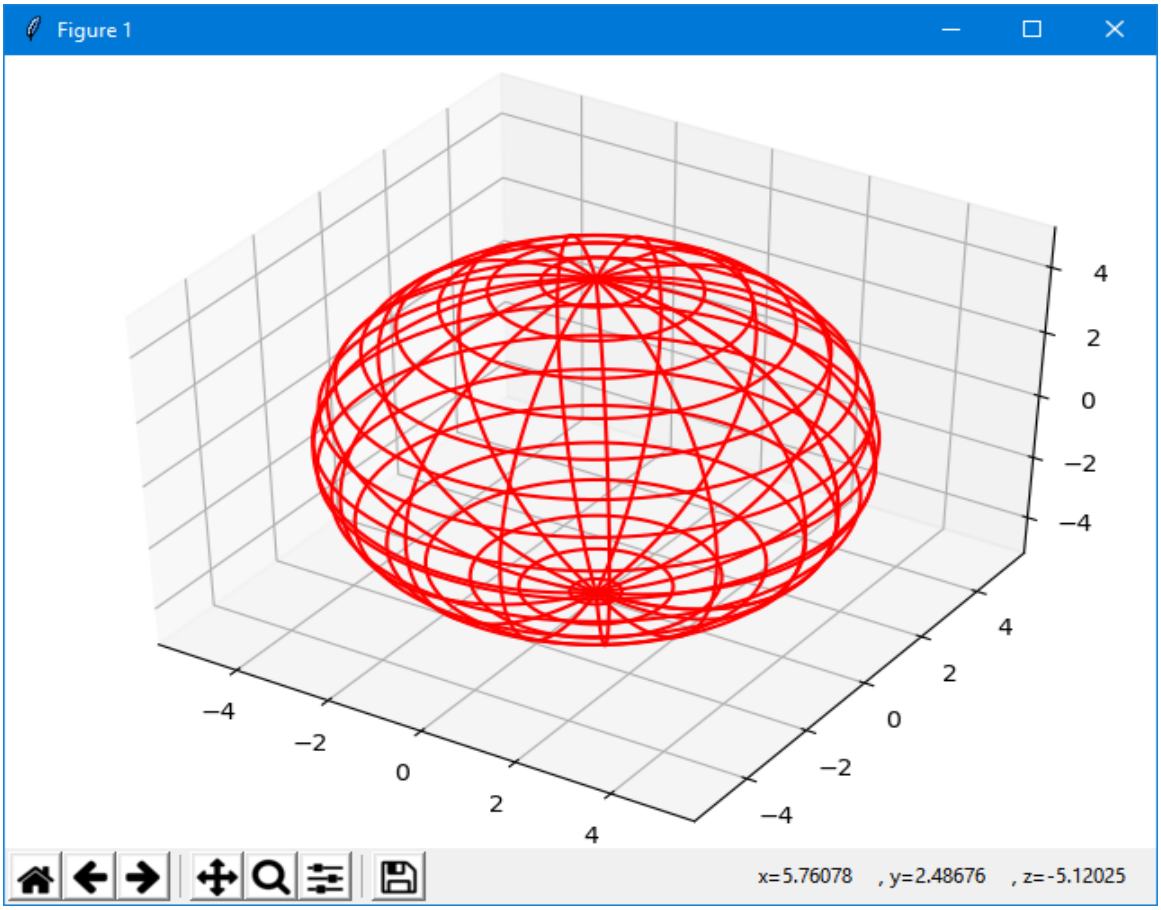
$u \in [-\pi, +\pi]$

$v \in [-2\pi, +2\pi]$

```
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()
```



Замінивши plot_surface на plot_wireframe отримаємо:



У програмі використовувалася функція модуля numpy.outer, яка обчислює зовнішній (прямий або тензорний) добуток двох векторів:

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

Для двох векторів a довгою N та b довгою M зовнішній твір визначається наступним правилом:

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

Це правило може бути узагальнено на масиви з більшою розмірністю. Але ця функція стискає всі багатовимірні масиви до однієї осі.

Параметри: a, b - числа, масиви NumPy чи подібні до масивів об'єкти. Одновимірні масиви, необов'язково однакової довжини. Двовимірні та багатовимірні масиви стискуються до однієї осі. out – масив NumPy, необов'язковий параметр.

Масив, в який можна помістити результат функції. Даний масив повинен відповідати формі та типу даних результуючого масиву функції, а також обов'язково бути C-суміжним, тобто. зберігати дані у рядковому З стилі. Вказівка даного параметра дозволяє уникнути зайвої операції присвоювання тим самим трохи прискорюючи роботу вашого коду. Корисний параметр якщо ви часто звертаєтеся до функції в циклі.

Повертає: результат - двовимірний масив NumPy, що є зовнішнім твором двох векторів.

Функція ones() повертає новий масив зазначеної форми та типу, заповнений одиницями. А np.ones(np.size(u)) - повертає новий масив, розміру і так само заповнений одиницями.

У наведеному вище прикладі будувалася так звана «щільна» сітка 3-мірного координатного простору на значеннях сітки координат x, y, z відповідно. Для цього використовувалася функція прямого або тензорного добутку двох векторів значень u та v.

Можна надійти «інше» - побудувати масив щільних координатних сіток 3-мірного координатного простору для зазначених у вигляді діапазонів одновимірних масивів координатних векторів u та v. Потім «просто» застосувати формули, що задають необхідні залежності x = x (u, v), y = y (u, v), z = z (u, v).

Для побудови масиву щільних координатних сіток використовуємо функцію numpy.mgrid:

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

Функція mgrid() повертає масив щільних координатних сіток N-вимірного координатного простору для зазначених у вигляді діапазонів одновимірних масивів координатних векторів.

Параметри: index object - об'єкт індексації

Під об'єктом індексації розуміється список із двох або трьох елементів, наприклад [0:5] або [0:5:10j].

Якщо елементів у списку всього два, це інтерпретується як напіввідкритий інтервал [start, ... , stop), у якому всі елементи відрізняються на 1, а значення stop в сам інтервал не входить.

Як третій елемент вказується уявна частина комплексного числа, яке вказує на кількість рівномірно рознесених елементів усередині закритого інтервалу [start, ... , stop], при цьому значення stop потрапляє в інтервал.

Повертає: результат - масив NumPy, що є масивом щільних координатних сіток N-мірного координатного простору, кількість та розміри яких залежать від зазначених діапазонів.

Побудову виконаємо наступним скриптом:

```
від 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 отримання масиву щільних координатних сіток можна також скористатися (розглянутою вище) функцією np.meshgrid:

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

За потреби можна весь малюнок «розтягнути» в kg разів по горизонталі та в kv разів по вертикалі. Для цього вкажемо параметр методу plt.figure():

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

П26. "Скролінг" по осі X

Matplotlib включає дуже невелику кількість елементів управління, які в термінах Matplotlib називаються віджетами, які розташовуються пакеті matplotlib.widgets. Цей пакет містить також віджети для взаємодії з графіками (віджети для виділення областей) – докладніше див., наприклад, http://jenyay.net/Matplotlib/Widgets.

Використовуючи віджет Slider можна «розтягнути» вісь X:

```
from matplotlib.widgets import Slider
import math
import matplotlib.pyplot as plt
# докладніше див.
# 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 роботи скрипту:

П27. «Динамічні» графіки

Без коментарів наведемо приклад «динамічної» побудови графіка см https://matplotlib.org/2.0.0/examples/animation/animate_decay.html:

```
'''
Цей випадок показуєвипадки виняткового позбавлення
animation.
'''
importnumpy as np
importmatplotlib.pyplot as plt
importmatplotlib.animation as animation

defdata_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.)

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

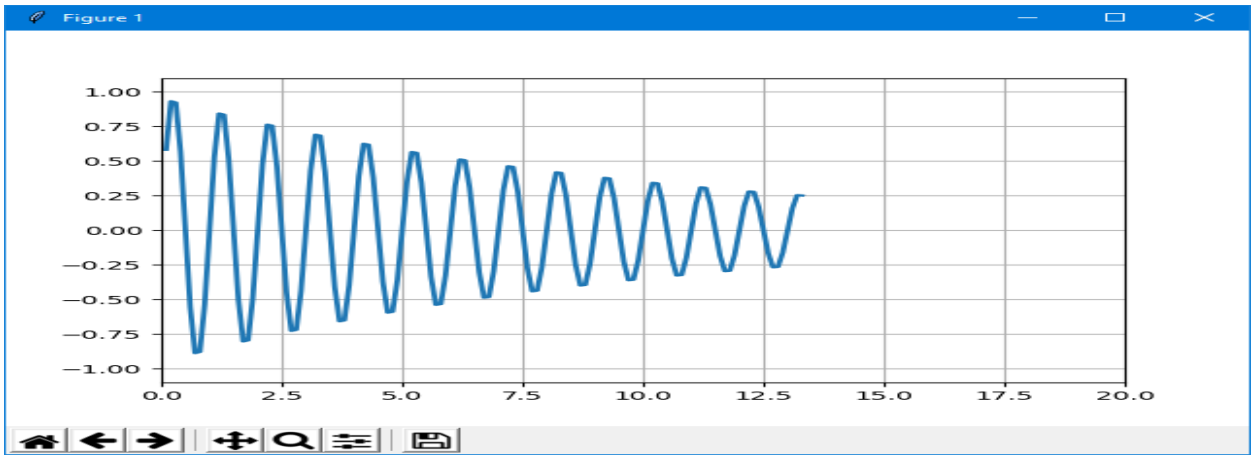
```
fig, ax = plt.subplots()
line, = ax.plot([], [], lw=2)
ax.grid()
xdata, ydata = [], []

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()
```



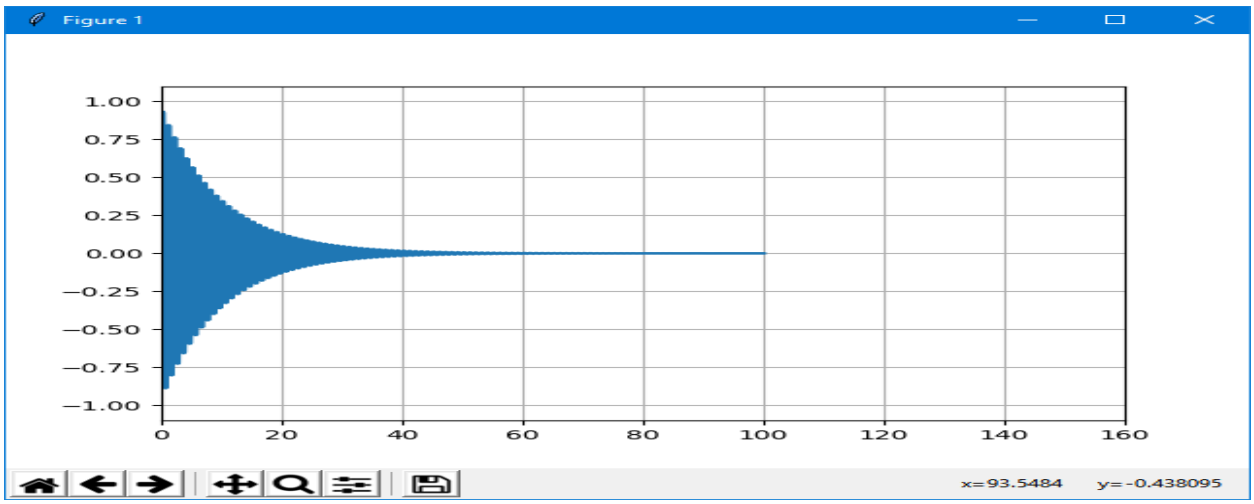


Figure 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100
120
140
160
x=93.5484
y=-0.438095