Scipy可视化实例

SV12

黄天羽

www.python123.org

实例1：mlab绘制洛伦茨吸引子轨迹

洛伦茨吸引子轨迹

回顾洛伦茨吸引子轨迹的微分方程：

$$\begin{cases} \dfrac{dx}{dt} = \sigma \cdot (y - x) \\ \dfrac{dy}{dt} = x \cdot (\rho - z) - y \\ \dfrac{dz}{dt} = xy - \beta z \end{cases}$$

“混沌”与“随机”不同

许多过程看起来像随机的：

- 从山上坠落的石头的翻滚
- 被海岸击碎的浪花

事实上并不是随机的，使用“决定性的混沌”来描述这一现象

洛伦茨吸引子轨迹

回顾洛伦茨吸引子轨迹的微分方程：

$$\begin{cases} \dfrac{dx}{dt} = \sigma \cdot (y - x) \\ \dfrac{dy}{dt} = x \cdot (\rho - z) - y \\ \dfrac{dz}{dt} = xy - \beta z \end{cases}$$

描述了：

- 水车的运动模式
- 电动发动机的运动模式
- … …

洛伦茨吸引子轨迹

定义函数lorenz（ ），计算出某个坐标点在各个方向上的微分值

```
import numpy as np

def lorenz(w, t, a, b, c):
    # 给出位置矢量w, 和三个参数a, b, c计算出
    # dx/dt, dy/dt, dz/dt的值
    x, y, z = w.tolist()
    # 直接与lorenz的计算公式对应
    return np.array([a * (y - x), x * (b - z) - y, x * y - c * z])
```

洛伦茨吸引子轨迹

使用不同的位移初始值，调用odeint，对微分方程求解。

```
t = np.arange(0, 30, 0.01)  # 创建时间点
# 调用ode对lorenz进行求解，用两个不同的初始值
track1 = odeint(lorenz, (0.0, 1.00, 0.0), t, args=(10.0, 28.0, 3.0))
track2 = odeint(lorenz, (0.0, 1.01, 0.0), t, args=(10.0, 28.0, 3.0))
```

洛伦茨吸引子轨迹

使用不同的位移初始值，调用odeint，对微分方程求解。

```
t = np.arange(0, 30, 0.01)  # 创建时间点
# 调用ode对lorenz进行求解，用两个不同的初始值
track1 = odeint(lorenz, (0.0, 1.00, 0.0), t, args=(10.0, 28.0, 3.0))
track2 = odeint(lorenz, (0.0, 1.01, 0.0), t, args=(10.0, 28.0, 3.0))
```

洛伦茨吸引子轨迹

使用mayavi对轨迹进行三维可视化：

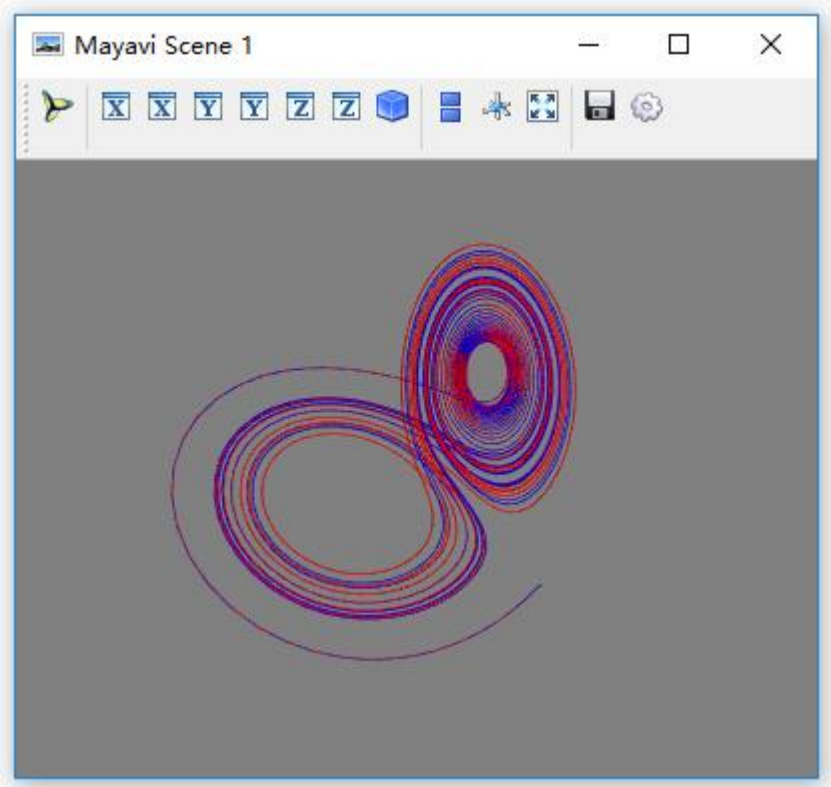
```
#绘制图形
from mayavi import mlab
mlab.plot3d(track1[:, 0], track1[:, 1], track1[:, 2], color=(1, 0, 0), tube_radius=0.1)
```

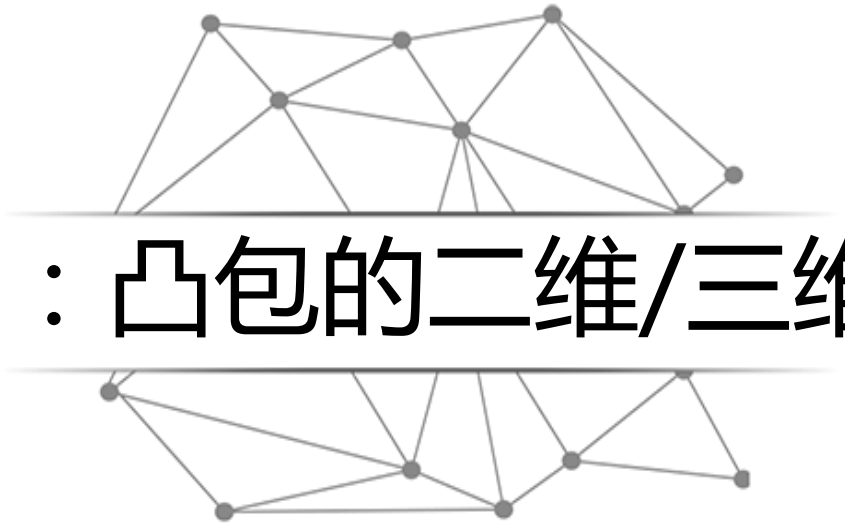
洛伦茨吸引子轨迹

```
from scipy.integrate import odeint
import numpy as np
from mayavi import mlab

def lorenz(w, t, a, b, c):
    # 给出位置矢量w, 和三个参数a, b, c计算出
    # dx/dt, dy/dt, dz/dt的值
    x, y, z = w.tolist()
    # 直接与lorenz的计算公式对应
    return np.array([a * (y - x), x * (b - z) - y, x * y - c * z])

t = np.arange(0, 30, 0.01)  # 创建时间点
# 调用ode对lorenz进行求解, 用两个不同的初始值
track1 = odeint(lorenz, (0.0, 1.00, 0.0), t, args=(10.0, 28.0, 3.0))
track2 = odeint(lorenz, (0.0, 1.01, 0.0), t, args=(10.0, 28.0, 3.0))

#绘制图形
mlab.plot3d(track1[:, 0], track1[:, 1], track1[:, 2], color=(1, 0, 0), tube_radius=0.1)
mlab.plot3d(track2[:, 0], track2[:, 1], track2[:, 2], color=(0, 0, 1), tube_radius=0.1)
```

洛伦茨吸引子轨迹

即使初始值只相差0.01，两条运动轨迹也完全不相同。





实例2：凸包的二维/三维可视化

SciPy的空间算法库-spatial

spatial模块提供了空间算法类：

- K-d树
- 凸包
- Voronoi图
- Delaunay三角化

SciPy的空间算法库-spatial

凸包：指N维空间中的一个区域，该区域中任意两点之间的线段都完全被包含在该区域之中，二维平面上的凸多边形就是典型的凸包。

`ConvexHull`可以快速计算包含N维空间点的集合的最小凸包。

Spatial-ConvexHull二维凸包

```
>>> import numpy as np
>>> np.random.seed(42)
>>> points2d = np.random.rand(10, 2) #一组二维平面的随机点
>>> from scipy import spatial
>>> ch2d = spatial.ConvexHull(points2d)#上述点的凸包对象
>>> ch2d.simplices #每条边线的两个顶点在 points2d的下标
array([[2, 5],
       [2, 6],
       [0, 5],
       [1, 6],
       [1, 0]], dtype=int32)
>>> ch2d.vertices  #多边形每个顶点在 points2d的下标
array([5, 2, 6, 1, 0], dtype=int32)
>>>
```

Spatial-ConvexHull二维凸包

写到文件里：

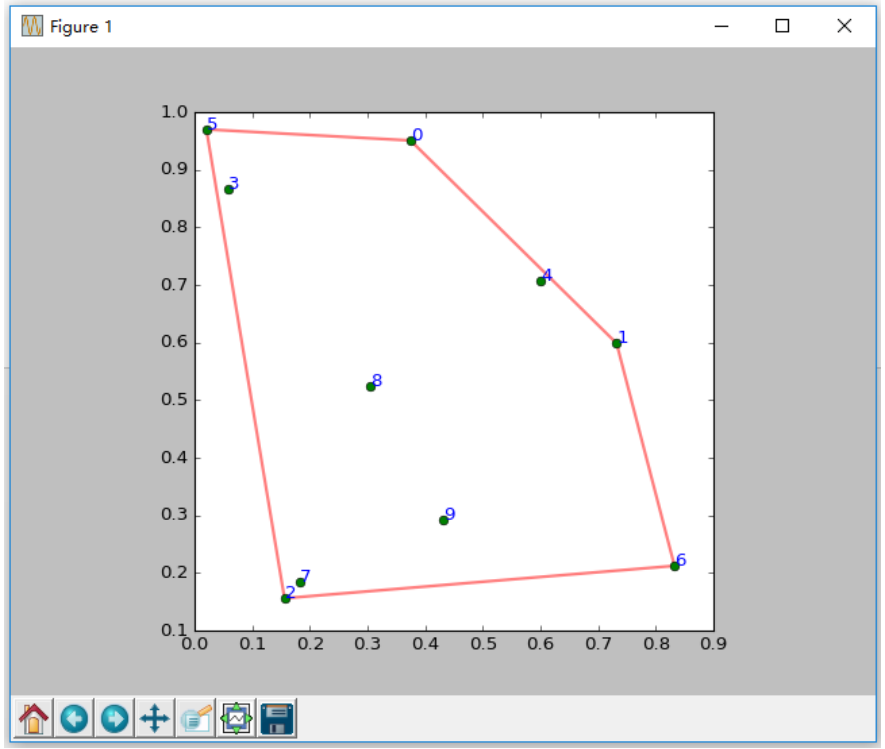
```
import numpy as np
from scipy import spatial

np.random.seed(42)
points2d = np.random.rand(10, 2)
ch2d = spatial.ConvexHull(points2d)
```

Spatial-ConvexHull二维凸包

使用matplotlib中的polygon对象绘制凸包：

```
import pylab as pl
poly = pl.Polygon(points2d[ch2d.vertices], fill=None, lw=2, color='r', alpha=0.5
ax = pl.subplot(aspect='equal')
pl.plot(points2d[:, 0], points2d[:, 1], 'go')
for i, pos in enumerate(points2d):
    pl.text(pos[0], pos[1], str(i), color='blue')
ax.add_artist(poly)
pl.show()
```

Spatial-ConvexHull二维凸包



Spatial-ConvexHull三维凸包

三维空间中的凸包是一个凸多面体，每个面都是一个三角形。

```
>>> import numpy as np
>>> np.random.seed(42)
>>> points3d = np.random.rand(40, 3)
>>> from scipy import spatial
>>> ch3d = spatial.ConvexHull(points3d)
>>> ch3d.simplices.shape
(38, 3)
>>>
```

所得到凸包由38个三角形面构成。

Spatial-ConvexHull三维凸包

TVTK对凸包进行可视化

```
import numpy as np
from scipy import spatial
from tvtk.api import tvtk

np.random.seed(42)
points3d = np.random.rand(40, 3)
ch3d = spatial.ConvexHull(points3d)
actors=convexhull(ch3d)#定义convexhull的Actor
win = ivtk_scene(actors)#场景用VTK绘制出来
```

Spatial-ConvexHull三维凸包

定义convexhull()的Actor：

```
from tvtk.api import tvtk
def convexhull(ch3d):
        #1 定义凸多面体tvtk的Polydata() 对象
        #2 定义凸多面体顶点的小球
        #3 绘制凸多面体的面，设置半透明度
        #4 绘制凸多面体的边，设置为红色
        #5 绘制凸多面体的顶点，设置为绿色
```

相关知识参考第一周课程。

Spatial-ConvexHull三维凸包

1 定义凸多面体tvtk的Polydata() 对象：

```
poly = tvtk.PolyData()
poly.points = ch3d.points
poly.polys = ch3d.simplices
```

2 定义凸多面体顶点的小球：

```
sphere = tvtk.SphereSource(radius=0.02)
points3d = tvtk.Glyph3D()
points3d.set_source_connection(sphere.output_port)
points3d.set_input_data(poly)
```

Spatial-ConvexHull三维凸包

3 绘制凸多面体的面，设置半透明度：

```
m1 = tvtk.PolyDataMapper()
m1.set_input_data(poly)
a1 = tvtk.Actor(mapper=m1)
a1.property.opacity = 0.3
```

Spatial-ConvexHull三维凸包

4 绘制凸多面体的边，设置为红色：

```
m2 = tvtk.PolyDataMapper()
m2.set_input_data(poly)
a2 = tvtk.Actor(mapper=m2)
a2.property.representation = "wireframe"
a2.property.line_width = 2.0
a2.property.color = (1.0, 0, 0)
```

Spatial-ConvexHull三维凸包

5 绘制凸多面体的顶点，设置为绿色：

```
m3 = tvtk.PolyDataMapper(input_connection=points3d.output_port)
a3 = tvtk.Actor(mapper=m3)
a3.property.color = (0.0, 1.0, 0.0)
```

```
from tvtk.api import tvtk
def convexhull(ch3d):
    poly = tvtk.PolyData()
    poly.points = ch3d.points
    poly.polys = ch3d.simplices

    sphere = tvtk.SphereSource(radius=0.02)
    points3d = tvtk.Glyph3D()
    points3d.set_source_connection(sphere.output_port)
    points3d.set_input_data(poly)
    #绘制凸多面体的面，设置半透明度
    m1 = tvtk.PolyDataMapper()
    m1.set_input_data(poly)
    a1 = tvtk.Actor(mapper=m1)
    a1.property.opacity = 0.3
    #绘制凸多面体的边，设置为红色
    m2 = tvtk.PolyDataMapper()
    m2.set_input_data(poly)
    a2 = tvtk.Actor(mapper=m2)
    a2.property.representation = "wireframe"
    a2.property.line_width = 2.0
    a2.property.color = (1.0, 0, 0)
    #绘制凸多面体的顶点，设置为绿色
    m3 = tvtk.PolyDataMapper(input_connection=points3d.output_port)
    a3 = tvtk.Actor(mapper=m3)
    a3.property.color = (0.0, 1.0, 0.0)
    return [a1, a2, a3]
```

Spatial-ConvexHull三维凸包