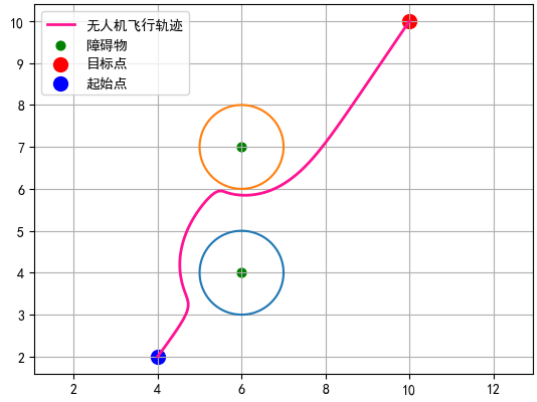
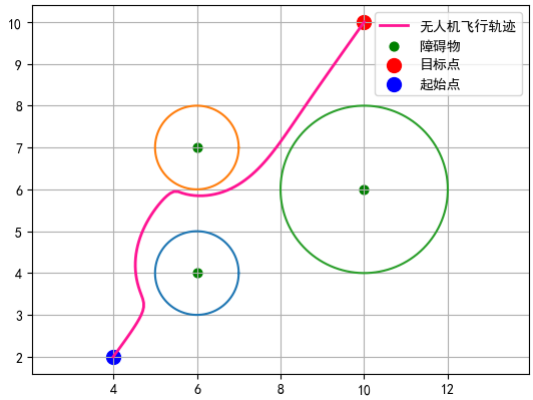
APF Python实现

二维

二维的APF比较简单，参见博客https://blog.csdn.net/junshen1314/article/details/50472410即可。





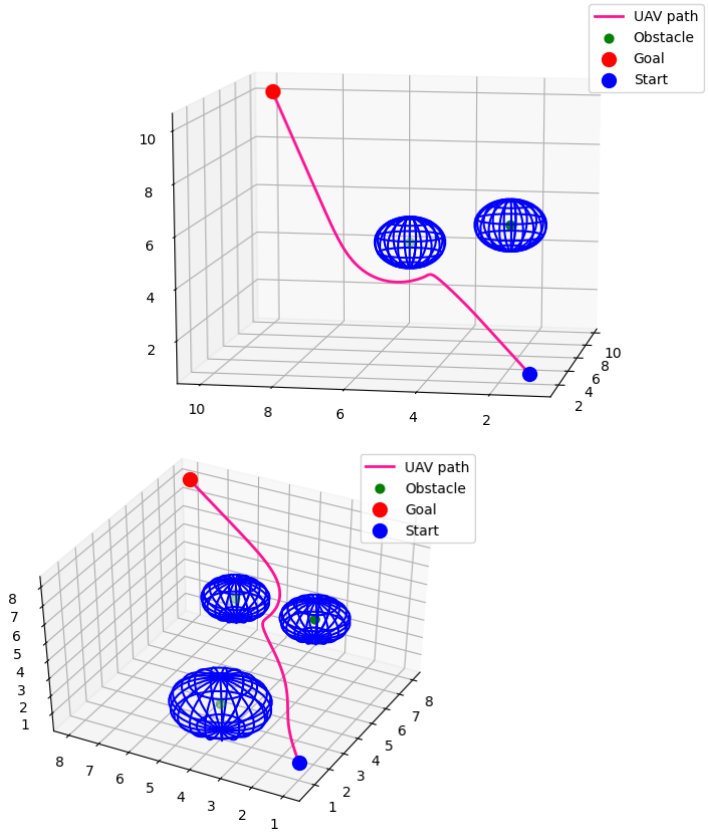

```python
import numpy as np
import matplotlib.pyplot as plt
from pylab import *                         #解决matplotlib无法显示中文问题
mpl.rcParams['font.sans-serif'] = ['SimHei']

class APF:
    def __init__(self):
        self.obstacle = np.array([[6, 4],
                                  [6,7],
                                  ])
        self.Robstacle = np.array([1,1])  #仅做测试
        self.qgoal = np.array([10,10])
        self.x0 = np.array([4,2])
        self.stepSize = 0.1
        self.iter = 1000
        self.epsilon = 0.8      #引力因子
        self.eta = 0.2           #斥力因子
        self.dgoal = 5
        self.r0 = 4
        self.path = self.x0.copy()
        self.path = self.path[np.newaxis,:]
        self.threshold = 0.5

```

```
    def distanceCost(self,point1,point2):
        return np.sqrt(np.sum((point1 - point2)**2))

    def attraction(self,q,qgoal,dgoal,epsilon):
        r = self.distanceCost(q,qgoal)
        if r <= dgoal:
            fx = epsilon * (qgoal[0] - q[0])
            fy = epsilon * (qgoal[1] - q[1])
        else:
            fx = dgoal * epsilon * (qgoal[0] - q[0]) / r
            fy = dgoal * epsilon * (qgoal[1] - q[1]) / r
        return np.array([fx,fy])

    def differential(self,q,other):
        output1 = (q[0] - other[0]) / self.distanceCost(q,other)
        output2 = (q[1] - other[1]) / self.distanceCost(q,other)
        return np.array([output1,output2])

    def repulsion(self,q,obstacle,r0,eta,qgoal):
        f0 = np.array([0,0])
        Rq2qgoal = self.distanceCost(q,qgoal)
        for i in range(obstacle.shape[0]):
            r = self.distanceCost(q,obstacle[i,:])
            if r <= r0:
                tempfvec = eta * (1 / r - 1 / r0) * Rq2qgoal ** 2 / r ** 2 *
self.differential(q, obstacle[i,:]) \
                           + eta * (1/r - 1/r0) ** 2 * Rq2qgoal *
self.differential(q,qgoal)
                f0 = f0 + tempfvec
            else:
                tempfvec = np.array([0,0])
                f0 = f0 + tempfvec
        return f0

    def loop(self):
        q = self.x0.copy()           #初始化位置
        for i in range(self.iter):
            Attraction =
self.attraction(q,self.qgoal,self.dgoal,self.epsilon)
            Repulsion =
self.repulsion(q,self.obstacle,self.r0,self.eta,self.qgoal)
            compositeForce = Attraction + Repulsion
            unitCompositeForce = compositeForce /
np.sqrt(np.sum((compositeForce) ** 2))
            q = q + self.stepSize * unitCompositeForce
            self.path = np.vstack((self.path,q))
            if self.distanceCost(q,self.qgoal) < self.threshold:
                self.path = np.vstack((self.path,self.qgoal))
                break
    def draw(self):
        plt.scatter(self.obstacle[:,0], self.obstacle[:,1], marker='o',
color='green', s=40, label='障碍物')
        plt.scatter(self.qgoal[0], self.qgoal[1], marker='o', color='red',
s=100, label='目标点')
        plt.scatter(self.x0[0], self.x0[1], marker='o', color='blue', s=100,
label='起始点')
```

```
 plt.plot(self.path[:,0],self.path[:,1],color="deeppink",linewidth=2,label = '无人机飞行轨迹')
        plt.legend(loc='best')  # 设置 图例所在的位置 使用推荐位置
        plt.grid()
        for i in range(self.Robstacle.shape[0]):
            self.drawCircle(self.obstacle[i,:],self.Robstacle[i])
        plt.axis('equal')
        plt.show()

    def drawCircle(self,pos,r):    #仅做测试
        theta = np.arange(0, 2 * np.pi, 0.01)
        a = pos[0]
        b = pos[1]
        x = a + r * np.cos(theta)
        y = b + r * np.sin(theta)
        plt.plot(x, y)

    def calculateTotalDistance(self):
        sum = 0
        for i in range(self.path.shape[0]-1):
            sum += self.distanceCost(self.path[i,:],self.path[i+1,:])
        return sum

if __name__ == "__main__":
    apf = APF()
    apf.loop()
    apf.draw()
    print("轨迹距离为:",apf.calculateTotalDistance())
```

三维

刚刚将二维APF升级为了三维版本，其实基本就是把所有数据格式增加了一列而已，至于矢量求导那也是很有规律性的，唯一变化大点的就是绘制部分了，绘制球的时候没有采用封闭式，而采用了框架式，这样方便看到轨迹如果与球相交后内部的情况。



看起来轨迹规划得还可以，但是还是会出现局部最优点以及轨迹穿过障碍物的情况。

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

class APF:
    def __init__(self):
        self.obstacle = np.array([[5,5,5],
                                  [4,2,6],
                                   [2,4,2]])  #障碍物坐标
        self.Robstacle = np.array([1,1,1.5]) #在apf中这个障碍物半径不会影响轨迹
        self.qgoal = [8,8,8]    #目标点
        self.x0 = np.array([1,1,1])  #轨迹起始点
        self.stepSize = 0.1  #物体移动的固定步长
        self.iter = 1000     #迭代次数
        self.epsilon = 0.8   #引力因子
        self.eta = 0.2       #斥力因子
        self.dgoal = 5       #当q与qgoal距离超过它时将衰减一部分引力
        self.r0 = 4          #斥力超过这个范围后将不复存在
        self.path = self.x0.copy()
        self.path = self.path[np.newaxis,:] #增加一个维度
        self.threshold = 0.5    #q与qgoal距离小于它时终止训练或者仿真

    def distanceCost(self,point1,point2):   #求两点之间的距离函数
        return np.sqrt(np.sum((point1 - point2) ** 2))

    def attraction(self,q,qgoal,dgoal,epsilon):  #计算引力的函数
        r = self.distanceCost(q,qgoal)
        if r <= dgoal:
            fx = epsilon * (qgoal[0] - q[0])
            fy = epsilon * (qgoal[1] - q[1])
            fz = epsilon * (qgoal[2] - q[2])
        else:
```

```
            fx = dgoal * epsilon * (qgoal[0] - q[0]) / r
            fy = dgoal * epsilon * (qgoal[1] - q[1]) / r
            fz = dgoal * epsilon * (qgoal[2] - q[2]) / r
        return np.array([fx,fy,fz])

    def differential(self, q, other):
        output1 = (q[0] - other[0]) / self.distanceCost(q, other)
        output2 = (q[1] - other[1]) / self.distanceCost(q, other)
        output3 = (q[2] - other[2]) / self.distanceCost(q, other)
        return np.array([output1, output2, output3])

    def repulsion(self,q,obstacle,r0,eta,qgoal):  #计算斥力的函数
        f0 = np.array([0,0,0])  #初始化斥力的合力
        Rq2qgoal = self.distanceCost(q,qgoal)
        for i in range(obstacle.shape[0]):
            r = self.distanceCost(q,obstacle[i,:])
            if r <= r0:
                tempfvec = eta * (1 / r - 1 / r0) * Rq2qgoal ** 2 / r ** 2
* self.differential(q, obstacle[i,:]) \
                           + eta * (1/r - 1/r0) ** 2 * Rq2qgoal *
self.differential(q,qgoal)
                f0 = f0 + tempfvec
            else:
                tempfvec = np.array([0,0,0])
                f0 = f0 + tempfvec
        return f0

    def loop(self):              #循环仿真
        q = self.x0.copy()
        for i in range(self.iter):
            Attraction =
self.attraction(q,self.qgoal,self.dgoal,self.epsilon)            #计算引力
            Repulsion =
self.repulsion(q,self.obstacle,self.r0,self.eta,self.qgoal)       #计算斥力
            compositeForce = Attraction + Repulsion
              #合力 = 引力 + 斥力
            unitCompositeForce = compositeForce /
np.sqrt(np.sum((compositeForce) ** 2))  #力单位化，apf中力只用来指示移动方向
            q = q + self.stepSize * unitCompositeForce        #计算下一位置
            self.path = np.vstack((self.path,q))              #记录轨迹
            if self.distanceCost(q,self.qgoal) < self.threshold:   #当与
goal之间距离小于threshold时结束仿真，并将goal的坐标放入path
                self.path = np.vstack((self.path,self.qgoal))
                break

    def draw(self):
        self.ax = plt.axes(projection='3d')
        self.ax.scatter3D(self.obstacle[:,0],
self.obstacle[:,1],self.obstacle[:,2], marker='o', color='green', s=40,
label='Obstacle')
        self.ax.scatter3D(self.qgoal[0], self.qgoal[1],self.qgoal[2],
marker='o', color='red', s=100, label='Goal')
        self.ax.scatter3D(self.x0[0], self.x0[1], self.x0[2], marker='o',
color='blue', s=100, label='Start')

 self.ax.plot3D(self.path[:,0],self.path[:,1],self.path[:,2],color="deeppin
k",linewidth=2,label = 'UAV path')
        plt.legend(loc='best')  # 设置 图例所在的位置 使用推荐位置
```

```
        plt.grid()
        for i in range(self.Robstacle.shape[0]):
            self.drawSphere(self.obstacle[i,:],self.Robstacle[i])
        plt.show()

    def drawSphere(self,center,radius):
        u = np.linspace(0, 2 * np.pi, 20)
        v = np.linspace(0, np.pi, 20)
        x = radius * np.outer(np.cos(u), np.sin(v)) + center[0]
        y = radius * np.outer(np.sin(u), np.sin(v)) + center[1]
        z = radius * np.outer(np.ones(np.size(u)), np.cos(v)) + center[2]
        self.ax.plot_wireframe(x,y,z, cstride = 4, color = 'b')

    def calculateTotalDistance(self):
        sum = 0
        for i in range(self.path.shape[0]-1):
            sum += self.distanceCost(self.path[i,:],self.path[i+1,:])
        return sum

if __name__ == "__main__":
    apf = APF()
    apf.loop()
    apf.draw()
    print("轨迹距离为:",apf.calculateTotalDistance())
```