import numpy as np  
from scipy.integrate import odeint  
import matplotlib.pyplot as plt  
import pickle # 添加导入语句  
from bayesian\_inference import simulate\_bayesian\_inference  
import random  
import rasterio  
import math  
  
  
  
"计算深度===================================================================="  
# 打开文件  
'"latitude维度"'  
'"longitude经度"'  
def open\_tiff\_file(tif\_path):  
 return rasterio.open(tif\_path)  
dataset = open\_tiff\_file("D:\MCM\_ICM\DEM\_1\gebco\_2023\_sub\_ice\_n90.0\_s0.0\_w0.0\_e90.0.tif")  
  
  
def calculate\_coordinates(origin\_lat, origin\_lon, deviation\_lat, deviation\_lon, unit='meters'):  
 # 单位转换  
 if unit == 'kilometers':  
 deviation\_lat /= 1000.0  
 deviation\_lon /= 1000.0  
  
 # 计算实际经纬度  
 new\_lat = origin\_lat + (deviation\_lat / 111.32) # 1 度纬度约为 111.32 公里  
 new\_lon = origin\_lon + (deviation\_lon / (111.32 \* abs(math.cos(math.radians(origin\_lat))))) # 1 度经度的长度随纬度变化而变化  
  
 return new\_lat, new\_lon  
  
  
# 获取指定位置的高程信息  
def get\_elevation(lat\_Y, lon\_X, dataset):  
 lat, lon = calculate\_coordinates(37.50000000, 18.00000000, lat\_Y, lon\_X)  
 row, col = dataset.index(lon, lat)  
 elevation = dataset.read(1, window=((row, row + 1), (col, col + 1)))  
 return -elevation[0]  
  
  
  
  
  
"密度函数计算========================================================"  
# 导入密度插值器  
def load\_interpolator(file\_path):  
 with open(file\_path, 'rb') as file:  
 interpolator = pickle.load(file)  
 return interpolator  
# 加载插值器文件  
density\_interpolator = load\_interpolator('density\_interpolator.pkl')  
density\_values = density\_interpolator(np.arange(0, 6001, 10))  
def get\_density(depth\_value):  
 density\_value = density\_interpolator(depth\_value)  
 min\_density, max\_density = 1025, 1070  
 density\_range = max\_density - min\_density  
 density = min\_density + density\_range \* (density\_value - min(density\_values)) / (max(density\_values) - min(density\_values))  
 return density  
  
  
  
"主模型==============================================================="  
def generate\_gaussian\_noise(factor):  
 return np.random.normal(0, factor)  
  
  
  
print(get\_elevation(-200, 600, dataset))  
def model(state, t):  
 x, y, z, vx, vy, vz = state  
  
 # 示例参数  
 length = 8.0  
 diameter = 2.7 # 圆柱体的直径，单位：米  
 height = 3.8 # 圆柱体的高度，单位：米  
 volume = length\*diameter\*height # 圆柱体的体积，单位：立方米  
 water\_density = get\_density(z)  
 gravity = 9.8 # 重力加速度，单位：m/s^2  
 Cd = 0.47 # 阻力系数  
  
 # 计算质量  
 mass = 18500.0 # 质量，单位：kg  
  
 # 计算重力  
 weight = (mass+1000000) \* gravity # 注意这里是正值，因为重力方向向下  
  
 # 计算浮力  
 buoyancy = water\_density \* gravity \* volume  
  
 # 计算 z 轴方向上的加速度  
 acceleration\_z = (-buoyancy + weight) / mass  
  
 # 计算海水阻力  
 reference\_area = diameter\*height # 参考面积  
  
 resistance\_z = 0.5 \* Cd \* water\_density \* reference\_area \* vz\*\*2  
  
 # 引入高斯随机漫步因子  
 noise\_factor = 0.01 # 调整噪声强度  
 # 计算微分方程  
 elevation = get\_elevation(y, x, dataset)  
 if z > elevation and abs(elevation - z) < 1e-6:  
 vx = 0  
 vy = 0  
 vz = 0  
 dvxdt = 0  
 dvydt = 0  
 dvzdt = 0  
 return [vx, vy, vz, dvxdt, dvydt,dvzdt]  
 dxdt = vx + generate\_gaussian\_noise(noise\_factor)  
 dvxdt = -resistance\_z \* np.sign(vx) / mass  
 dydt = vy + generate\_gaussian\_noise(noise\_factor)  
 dvydt = -resistance\_z \* np.sign(vy) / mass  
 if z < 0:  
 print(z)  
 z = 0  
 vz = 0  
 dzdt = 0  
 dvzdt = 0  
 dxdt = vx + generate\_gaussian\_noise(random.uniform(0, 0.1))  
 dvxdt = -resistance\_z \* np.sign(vx) / mass  
 dydt = vy + generate\_gaussian\_noise(random.uniform(0, 0.1))  
 dvydt = -resistance\_z \* np.sign(vy) / mass  
 return [dxdt, dydt, dzdt, dvxdt, dvydt, dvzdt]  
 dvzdt = acceleration\_z - resistance\_z \* np.sign(vz) / mass  
 dzdt = vz + generate\_gaussian\_noise(noise\_factor)  
 return [dxdt, dydt, dzdt, dvxdt, dvydt, dvzdt]  
  
# 其余部分保持不变  
def simulate\_motion(initial\_state, time\_points):  
 result = odeint(model, initial\_state, time\_points, atol=1e-3, rtol=1e-3)  
 return result  
  
# 设置初始状态  
initial\_state = [600.0, -200.0, 300.0, 80.0, 58.0, -47.0]  
  
# 设置时间点  
time\_points = np.linspace(0, 10, 100000)  
  
# 模拟运动  
result\_front = simulate\_motion(initial\_state, time\_points)  
# 只取前 4900 个点  
result = result\_front[:2993, :]  
  
  
  
"检查与绘图============================================================================================"  
# 将 result\_subset 写入 CSV 文件  
np.savetxt('result.csv', result, delimiter=',', header='X,Y,Z,Vx,Vy,Vz', comments='')  
  
# 输出末尾状态  
print("末尾状态:", result[-1])  
  
  
fig = plt.figure(figsize=(10, 6), dpi=300)  
ax = fig.add\_subplot(111, projection='3d')  
  
# 绘制所有位置点  
ax.scatter(result[:, 0], result[:, 1], -result[:, 2], c='red', s=1, label='Position') # Negating z-coordinates  
# 设置坐标轴标签  
ax.set\_xlabel('X')  
ax.set\_ylabel('Y')  
ax.set\_zlabel('Z')  
  
# 设置标题  
ax.set\_title('Object Motion')  
  
# 显示图形  
plt.show()  
plt.savefig("seabeg\_motion.pdf")  
  
  
  
""""" -----------贝叶斯推理 --------------------------------------------------------------"""  
  
# # 设置观测位置  
# observed\_position = result[-1, :3] # 使用模拟结果的末尾位置作为观测位置  
# std\_deviation = 1.0 # 观测位置的标准差，可以根据实际情况调整  
#  
# # 设置贝叶斯推理的坐标范围  
# x\_range = np.linspace(-10, 10, 100)  
# y\_range = np.linspace(-10, 10, 100)  
# z\_range = np.linspace(-10, 10, 100)  
#  
# # 设置贝叶斯推理的坐标范围  
# center\_x, center\_y, center\_z = result[-1, :3]  
# x\_range = np.linspace(center\_x - 2.5, center\_x + 2.5, 100)  
# y\_range = np.linspace(center\_y - 2.5, center\_y + 2.5, 100)  
# z\_range = np.linspace(center\_z - 2.5, center\_z + 2.5, 100)  
#  
# # 进行贝叶斯推理  
# posterior = simulate\_bayesian\_inference(observed\_position, std\_deviation, x\_range, y\_range, z\_range)  
#  
# # 将 X, Y, Z 合并为一个3D坐标数组  
# X, Y, Z = np.meshgrid(x\_range, y\_range, z\_range)  
#  
# # 将 X, Y, Z 合并为一个2D坐标数组  
# coords\_2d = np.c\_[X.ravel(), Y.ravel(), Z.ravel()]  
# posterior\_2d = posterior[:, :, 0]  
#  
# # 使用 posterior 作为高度值  
# # 使用 posterior 作为高度值  
# fig = plt.figure(figsize=(12, 8), dpi=300)  
# ax = fig.add\_subplot(111, projection='3d')  
#  
# # 绘制概率密度图  
# ax.plot\_surface(X[:,:,0], Y[:,:,0], posterior\_2d, cmap='viridis', alpha=0.7)  
#  
# # 绘制贝叶斯推理结果位置点  
# ax.scatter(result[:, 0], result[:, 1], -result[:, 2], c='red', s=1, label='Position') # Negating z-coordinates  
#  
# ax.set\_xlabel('X')  
# ax.set\_ylabel('Y')  
# ax.set\_zlabel('Z')  
# ax.set\_title('Object Motion with Bayesian Inference')  
#  
# # 显示图形  
# plt.show()  
#  
# # 新添加的代码块  
# # 使用 posterior 作为高度值  
# fig = plt.figure(figsize=(12, 8), dpi=300)  
#  
# # 3D 图  
# ax1 = fig.add\_subplot(121, projection='3d')  
#  
# # 绘制概率密度图  
# ax1.plot\_surface(X[:,:,0], Y[:,:,0], posterior\_2d, cmap='viridis', alpha=0.4)  
#  
# # 绘制贝叶斯推理结果位置点  
# ax1.scatter(result[:, 0], result[:, 1], -result[:, 2], c='blue', s=1, label='Position') # Negating z-coordinates  
#  
# ax1.set\_xlabel('X')  
# ax1.set\_ylabel('Y')  
# ax1.set\_zlabel('Z')  
# ax1.set\_title('Object Motion with Bayesian Inference')  
#  
# # 平面图  
# ax2 = fig.add\_subplot(122)  
#  
# # 设置平面图背景为蓝色  
# # ax2.set\_facecolor('blue')  
#  
# # 绘制概率密度图的等高线，去除连接线  
# contour = ax2.contour(X[:,:,0], Y[:,:,0], posterior\_2d, cmap='viridis', linewidths=1)  
# ax2.clabel(contour, inline=True, fontsize=8)  
#  
# # 在平面图上标注深度和坐标  
# ax2.scatter(result[:, 0], result[:, 1], c='red', s=10, label='Position') # 只标注位置，不显示z坐标  
# for i, txt in enumerate(result[:, 2]):  
# ax2.annotate(f'{txt:.2f}', (result[i, 0], result[i, 1]), fontsize=8, color='white')  
#  
# ax2.set\_xlabel('X')  
# ax2.set\_ylabel('Y')  
# ax2.set\_title('Probability Distribution and Position')  
#  
# # 隐藏坐标轴  
# # ax2.axis('off')  
# # 添加手动标签  
# ax2.text(result[:, 0].min(), result[:, 1].min(), 'X', fontsize=12, color='white', ha='right', va='top')  
# ax2.text(result[:, 0].max(), result[:, 1].min(), 'Y', fontsize=12, color='white', ha='left', va='top')  
# ax2.text(result[:, 0].min(), result[:, 1].max(), 'Z', fontsize=12, color='white', ha='right', va='bottom')  
# ax2.text(result[:, 0].max(), result[:, 1].max(), 'Probability Distribution', fontsize=12, color='white', ha='left', va='bottom')  
# # 显示图形  
# plt.show()