对多波束测深问题的模型建立与求解

摘要

多波束测深是在单波束测深的基础上改良而来，能够一次性测量出以测线为轴的全覆盖测量水深条带，具有优越的测深性能。

针对**问题 1**，首先在多波束发射平面上建立**几何模型**，根据坡度与海水深度求解多波束在坡面上的覆盖长度，然后求得其在水平面上的投影即为覆盖宽度。最后利用重叠率公式计算出第一条测线外的测线的重叠率。测线距中心点$800m$内时，重叠率最大为0.37，最小为-0.06。每隔$200m$的距离，覆盖宽度的变化大小为$18.18m$，海水深度的变化大小为$5.24m$。

针对**问题 2**，建立自适应深度与自适应有效坡面的**几何模型**，首先通过测线方向与其水平垂线的方向向量与坡面的法向量，求解出测线与其水平垂线在坡面以及水平面上的投影向量。然后通过**余弦定理**求解出两组投影向量的夹角，先用测线组的夹角求解出海水深度，最后用垂线组的夹角求解出覆盖宽度。最终测量船距海域中心点的距离为 0 及 2.1 海里时，覆盖宽度最大分别为$416.69m, 770.07m$，最小分别为$416.12m, 63.06m$。

针对**问题 3**，首先确定测量船的航线为直线并与海底坡面的等高线平行，然后根据多波束的精确测量范围确定第一条测线的位置，最后通过固定重叠率为10%，利用**贪心思想迭代求解**出后续每一条测线的位置。所得能够覆盖全区域的测线组的总测线长度为 $129640m$。

针对**问题 4**，首先通过可视化分析海底坡面特征，再利用**蒙特卡洛模拟**观察船只的模糊走向，结合特征图与走向图对当前海域进行分区，为每一个海域分区进行初步的测线布设设计，然后使用**最小二乘法**对采样点周围进行海底坡面**拟合**，最后利用**改进的鲸鱼优化算法**求解出测线在每一个分区的具体位置。最终的指标求解包括测线总长度为$\mathbf{43734}\boldsymbol{m}$，漏测海域面积为$\mathbf{0}$，超额重叠率的测线长度为$\mathbf{7408}\boldsymbol{m}$。

关键词：多波束测深 几何原理 迭代求解 蒙特卡洛模拟 鲸鱼优化算法

一、 问题重述

1.1 问题背景

测量海水深度需要利用波束探测。利用声波反射原理，测量声波速度并记录声波传播时间，利用数据求解海水深度。波束测深分为单波束测深和多波束测深。单波束测深所收集数据较为局限，相邻两条测线间并无数据。多波束测深系统针对其缺点进行完善，可测出测线两边一定范围的数据。

1.2 问题重述

问题 1：垂直测线的平面与需要探测的坡面的相交，交线与水平夹角的坡度为 1.5°。换能器可探测角度为 120°，穿过中心点侧线的水深 70 m，建立覆盖宽度、相邻条带间重叠率的数学模型。求解表格内的数据。

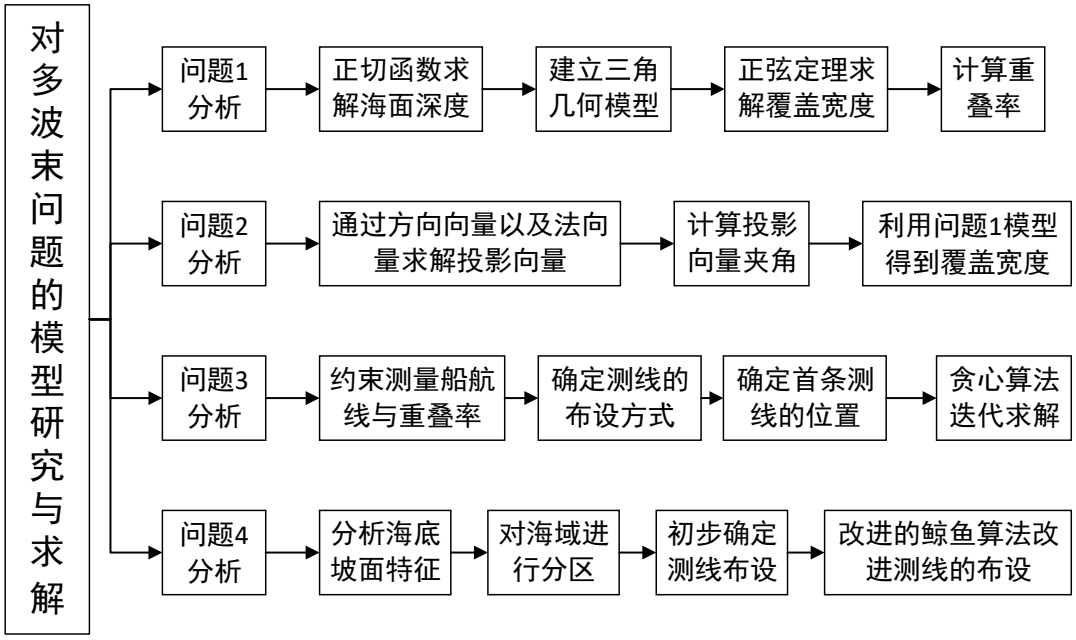
问题 2：水下坡面法向量在水平面上的投影与侧线方向夹角为β，测量海域为矩形，换能器可探测 120°，水下坡面坡度为 1.5°，根据要求建立覆盖宽度的数学模型。

问题 3：矩形海域南北长 2 海里、东西宽 4 海里，水下坡面由西向东逐渐升高，坡度为 1.5°，海域中心点处的水深 110m，换能器探测范围为 120°。根据已知数据设计一组测线，此测线要求矩形海域被完全覆盖且测量长度最短，相邻条带间重叠率须在 10%~20%的范围内。

问题 4：设计一条路线需满足测线的总长尽可能短但条带尽可能覆盖整个海域，并且相邻条带间重叠率尽可能保持在 20%以下。计算这条测线总长、漏测面积与待测海域面积的百分比，以及重叠区域中重叠率超过 20%的测线总长。

二、 问题分析

总体分析大纲框架如下：



2.1 问题 1 的分析

因为问题 1 中海底坡面是一个具有坡度的斜面，因此首先可以通过测线距海域中心点的距离以及正切函数计算出位于每一条测线上的船体下的海水深度，然后根据当下的海面深度以及多波束换能器的开角大小建立三角几何模型，通过正弦定理求解模型得到多波束测深的覆盖宽度W。最后利用相邻的条带覆盖宽度得到相邻条带之间的重叠率。

2.2 问题 2 的分析

问题 2 相比问题 1，由于测线方向的变化，海底坡度在测线垂平面方向以及测线竖直平面方向上所体现出来的两个坡度是首要目标，首先在测线垂平面以及测线竖直平面上取一水平方向向量，分别与海底坡面与水平面上的法向量进行计算，得到方向向量在两个平面上的投影向量，然后计算两个投影向量的夹角即可求出海底坡度在侧线垂平面方向以及测线竖直平面方向上所体现出来的坡度，最后利用问题 1 中的三角几何模型即可求得覆盖宽度。

2.3 问题 3 的分析

问题 3 要求在满足覆盖率和重叠率的条件下，设计一组总测量长度最短的测线，结合现实生活中船的航线应尽量趋于直线以及多波束测量的覆盖宽度随海水深度的加深而增大[1]，因此将航线方向限制为垂直于坡面法向量的方向，重叠率固定为10%，通过迭代求解得到每一条测线的具体位置。

2.4 问题 4 的分析

问题 4 海域的海底坡面是不规则的，因此为了更好地建立模型，首先对海底坡面进行可视化分析，了解该海域的地理特征。然后针对航船的位置进行海底坡面的拟合，利用拟合的海底坡面结合问题 2 建立多波束覆盖宽度的模型，再根据海域的地理特征，对海域进行分区处理，对不同区域实行不同的航线布设。最后设定目标函数为测线总距离最短，约束条件为重叠率以及覆盖率，通过改进的鲸鱼优化算法进行最终结果的求解。

三、 模型假设

（1） 假设测量船扫描的区域不存在盲区。

（2） 假设换能器能够完全接受信号并建立地形图。

（3） 假设测量船能在测量海域自由航行，不存在障碍。

（4） 假设测量船无续航问题。

（5） 假设问题 4 中的海底数据依旧符合现在的海域。

（6） 假设测量船在海面上换向无最小半径限制。

（7） 假设多波束测量的有效距离满足本问测量的距离。

四、 符号说明

符号	符号意义
d	测线距海域中心点的距离
D_d	距离为d时的海水深度
W	覆盖宽度
η	重叠率
θ_1	测线垂面在水平面和坡面投影的夹角
θ_2	测线竖直平面在水平面和坡面投影的夹角
L	测线水平方向上垂线的空间直线方程
L_1	测线垂线在水平面投影的空间直线方程
L_2	测线垂线在坡面投影的空间直线方程
w_j	第j条测线条带的覆盖宽度
$\boldsymbol{\beta}$	待求参数向量
$\boldsymbol{l}$	观测向量
$\boldsymbol{V}$	观测值的改正数向量
$\boldsymbol{Y}$	目标坐标系函数矩阵
$\boldsymbol{B}$	原坐标系函数矩阵
μ	参数尺度
$\boldsymbol{M}$	映射坐标旋转矩阵
$X^*(t)$	当前最优解
pp	自适应p值
d_n	第n条测线在相应轴上与初始点的距离
β	分割海域的直线与边界的夹角
l_n	第n条测线的长度

五、 问题 1 的模型建立与求解

如图 1 所示，在船体正下方海底坡面上的一点作水平线，根据角的关系以及正弦定理求解两部分覆盖宽度，最终求得总覆盖宽度以及重叠率。

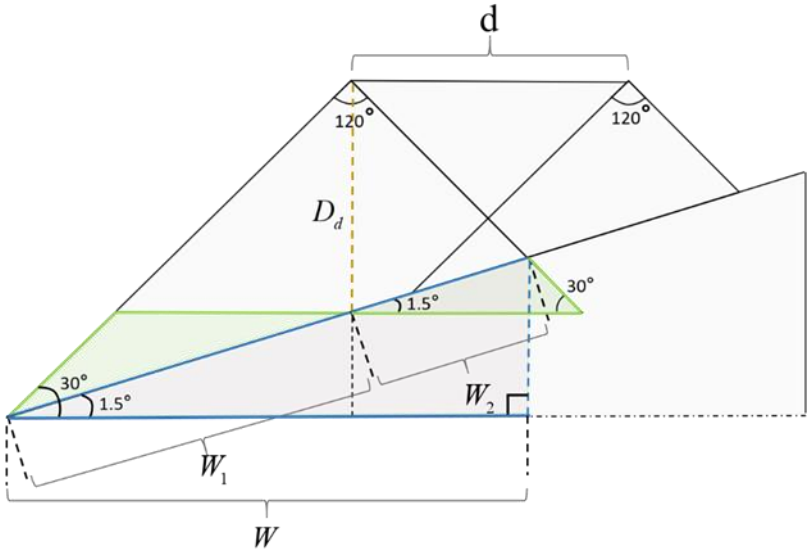


图 1 问题 1 图解

5.1 覆盖宽度

首先通过测线距海域中心点的距离以及正切函数计算出位于每一条测线上的船体下的海面深度，海面深度的计算公式如下：

$$D_d = 70 - d \cdot \tan 1.5° \tag{1}$$

在知道船体下当前的海水深度后，结合海底坡面以及多波束换能器的开角角度即可通过正弦定理求解出总覆盖宽度W，其具体求解式如下：

$$\frac{W_1}{\sin 150°} = \frac{D_d \cdot \tan 60°}{\sin(30° - 1.5°)} \tag{2}$$

$$\frac{W_2}{\sin 30°} = \frac{D_d \cdot \tan 60°}{\sin(30° + 1.5°)} \tag{3}$$

$$W = (W_1 + W_2)\cos 1.5° \tag{4}$$

结合(2), (3), (4)式即可求解出覆盖宽度W的最终表达式，如下所示：

$$W = \left(\frac{\sqrt{3}}{2\sin(28.5°)} + \frac{\sqrt{3}}{2\sin(31.5°)}\right) D_d \cdot \cos 1.5° \tag{5}$$

5.2 重叠率

题目给出了在测线相互平行且海底地形平坦时的重叠率计算公式，但在问题一中海底坡面并不水平，因此需要将海底坡面上的覆盖宽度投影到水平面上，该操作在上文计算覆盖宽度时就已实现，因此表达式如题中所示：

$$\eta = 1 - \frac{d}{W} \tag{6}$$

5.3 结果展示

表 1 问题 1 的计算结果

测线距中心点处的距离/m	-800	-600	-400	-200	0	200	400	600	800
海水深度/m	90.95	85.71	80.47	75.24	70	64.76	59.53	54.29	49.05
覆盖宽度/m	315.71	297.53	279.35	261.17	242.99	224.81	206.63	188.45	170.27
与上一条测线的重叠率/%	—	0.37	0.33	0.28	0.23	0.18	0.11	0.03	-0.06

六、 问题 2 的模型建立与求解

6.1 模型建立与求解

问题二的关键在于计算测线的垂面与坡面和水平面交线之间的夹角θ_1，以及测线竖直平面与坡面和水平面交线之间的夹角θ_2，在得出两个夹角后，问题 2 便转化为测线方向坡度以及测线垂面方向坡度随测线方向变化而变化的问题。建立直角坐标系，将所求夹角变为投影在坡面的直线与投影在水平面的直线之间的夹角问题。具体步骤如下：

step1. 建立平面直角坐标系，设测线在水平方向上的垂线的空间直线方程L为：

$$\frac{x - x_0}{X_0} = \frac{y - y_0}{Y_0} = \frac{z - z_0}{Z_0} \tag{7}$$

step2. 测线在水平方向上的垂线在水平面的投影的直线方程L_1为：

$$\frac{x-x_1}{X_1}=\frac{y-y_1}{Y_1}=\frac{z-z_1}{Z_1} \tag{8}$$

step3. 测线在水平方向上的垂线在坡度为 1.5° 的坡面的投影直线方程L_2为：

$$\frac{x-x_2}{X_2}=\frac{y-y_2}{Y_2}=\frac{z-z_2}{Z_2} \tag{9}$$

step4.在直角坐标系里，空间两直线L_1 L_2的夹角余弦为

$$\cos\theta_1=\frac{X_1X_2+Y_1Y_2+Z_1Z_2}{\sqrt{X_1^2+Y_1^2+Z_1^2}\cdot\sqrt{X_2^2+Y_2^2+Z_2^2}} \tag{10}$$

step5.利用反余弦函数得出角

$$arc\cos\left(\frac{X_1X_2+Y_1Y_2+Z_1Z_2}{\sqrt{X_1^2+Y_1^2+Z_1^2}\cdot\sqrt{X_2^2+Y_2^2+Z_2^2}}\right)=\theta_1 \tag{11}$$

在得到测线垂面在海底坡面以及水平面上的投影的夹角θ_1后，即可通过同样的原理求得测线竖直平面与坡面和水平面交线之间的夹角θ_2，然后可通过两夹角求解多波束覆盖宽度的大小，具体如下图所示：

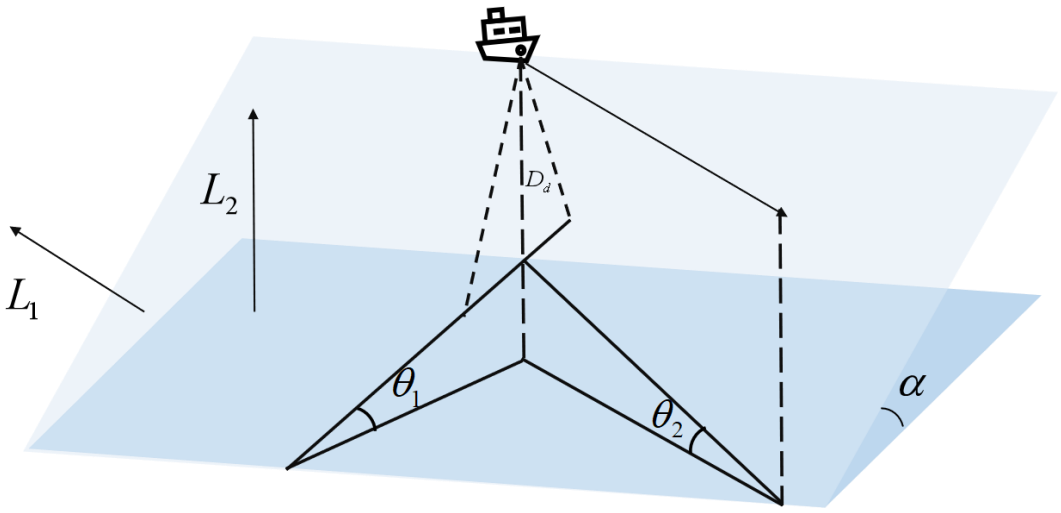


图 2 求解真实坡度图

测量船下的海水深度可通过测线竖直平面与坡面和水平面交线之间的夹角θ_2进行求解，得到关于测量船距离海域中心点距离为d的深度变化公式如下：

$$D_d=70-d\cdot\tan\theta_2 \tag{12}$$

在得到图 2 中的θ_1以及随θ_2变化的海水深度D_d后，即可利用问题 1 中的三角几何模型进行最终覆盖宽度的求解，具体如下图所示：

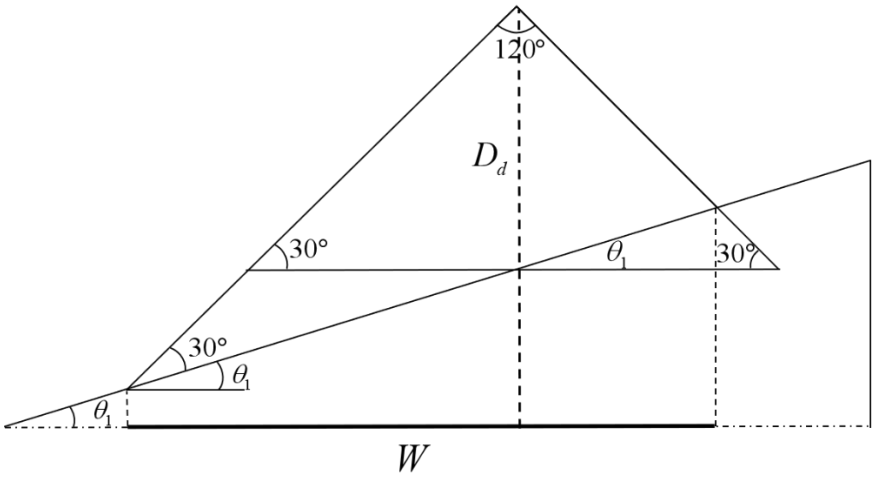


图 3 覆盖宽度求解

然后，结合问题 1 中的(5)式进行关于海水深度以及坡度角θ进行求解，获得最终的覆盖宽度，具体如下：

$$W = \left(\frac{\sqrt{3}}{2\sin(30° - \theta_1)} + \frac{\sqrt{3}}{2\sin(30° + \theta_1)}\right) D_d \cdot \cos\theta_1 \tag{13}$$

联解(11),(13)即可获得最终的模型表达式：

$$W = D_d\left(\frac{\sqrt{3}}{2\sin(30° - \theta_1)} + \frac{\sqrt{3}}{2\sin(30° + \theta_1)}\right)\left(\frac{X_1X_2 + Y_1Y_2 + Z_1Z_2}{\sqrt{X_1^2 + Y_1^2 + Z_1^2} \cdot \sqrt{X_2^2 + Y_2^2 + Z_2^2}}\right) \tag{14}$$

6.2 结果展示

表 2 问题 2 的计算结果

覆盖宽度/m		测量船距海域中心点处的距离/海里							
		0	0.3	0.6	0.9	1.2	1.5	1.8	2.1
测线方向夹角/°	0	416.55	467.05	517.55	568.06	618.56	669.06	719.57	770.07
	45	416.12	451.79	487.46	523.12	558.79	594.46	630.13	665.8
	90	415.69	415.69	415.69	415.69	415.69	415.69	415.69	415.69
	135	416.12	380.45	344.78	309.12	273.45	237.78	202.11	166.44
	180	416.55	366.05	315.54	265.04	214.54	164.04	113.53	63.03
	225	416.12	380.45	344.78	309.12	273.45	237.78	202.11	166.44
	270	415.69	415.69	415.69	415.69	415.69	415.69	415.69	415.69
	315	416.12	451.79	487.46	523.12	558.79	594.46	630.13	665.8

七、 问题 3 的模型建立与求解

由文献【1】可知，当测线走向与测量船所在的海域等深线走向一致时，可以最大程度地增加海底坡面的覆盖率，因此针对问题 3，我们选择自北向南或自南向北的测线方向，保持测线的走向与等深线一致。并且从文献中还了解到相邻条带之间的重叠率被要求在10%以上是为了避免边缘测量的精度问题，因此考虑到精准度，在计算第一条边缘测线位置时考虑测线覆盖范围的长度，将海域边缘也纳入80%的精准测量范围。如下图所示：

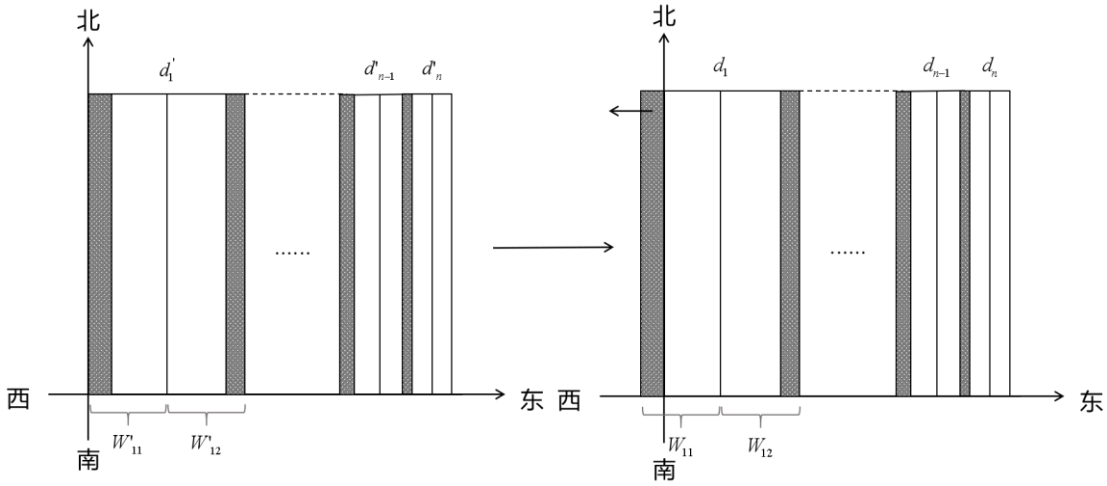


图 4

对于第一条测线，可以根据重叠率的范围建立一元一次方程保证测深的精准度，因为要求测量长度最短，因此将重叠率固定在10%，然后通过方程变换并结合公式(2),(3)得到测线横轴坐标表达式进行求解，方程与表达式如下：

$$w_{11} - d_1 = 0.1(w_{11} + w_{12})\cos 1.5° \tag{15}$$

$$d_1 = (1 - 0.1\cos 1.5°)\frac{\sqrt{3} \times D_d}{2\sin(30° - 1.5°)} - 0.1\frac{\sqrt{3}D_d \cos 1.5°}{2\sin(30° + 1.5°)} \tag{16}$$

在得到第一条测线的位置后，即可进行迭代求解，除去第一条测线，从第二条测线开始，每一条测线的位置都可根据前面的测线的位置求出，递推式如下：

$$d_2 = w_{21} + 0.9w_1$$

$$d_3 = w_{31} + 0.9(w_1 + w_2) \tag{17}$$

$$d_4 = w_{41} + 0.9(w_1 + w_2 + w_3)$$

根据(17)式的规律，可以得到后续每一条相邻的测线位置表达式：

$$d_n = w_{n1} + 0.9\sum_{j=1}^{n-1} w_j \tag{18}$$

最终通过 python 即可求解得出满足条件下所有的测线，测线大致预览图如下图所示，其中蓝线表示测线，红色交界处为重叠处，详细结果见附录。

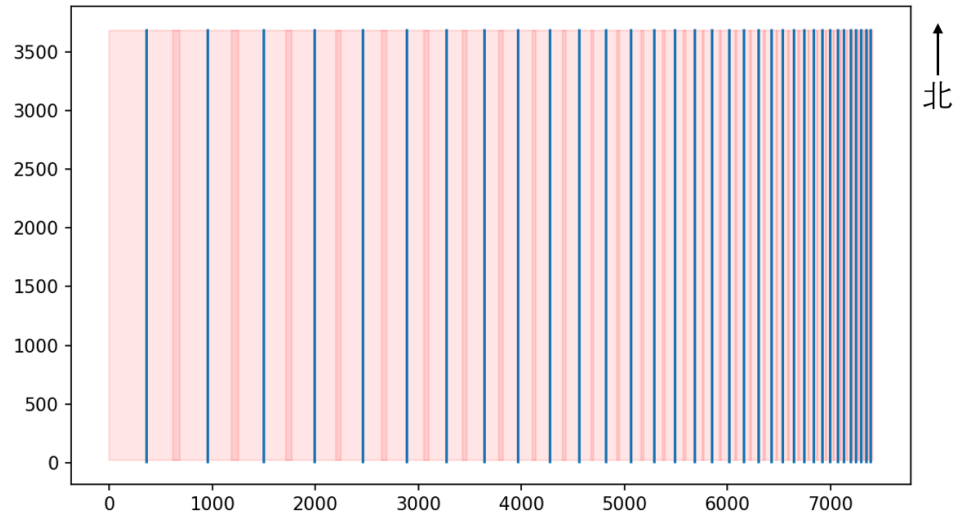


图 5 测线位置预览图

八、 问题 4 的模型建立与求解

在目前测深领域中，较为常见的是平行计划测线以及扇形计划测线[2]，这两种布设方式都与待测海域的海底地形息息相关，类似于平行线的测线布设方式基本都会在等深线的方向上进行布设与测量，而扇形布设方式则多分布在海岛，礁石，河道转弯处等。针对本问的测线布设问题，需要对当前海域的海底地形进行一定的了解，在设计适当的测线布置方式。

8.1 海底坡面的分析

不同于前面的问题，问题 4 的海底坡面并不是一个简单的单方向坡面，因此在建立问题 4 模型前，首先对问题 4 所给数据的海域坡面进行地势和等深线两方面的可视化分析，图像如下：

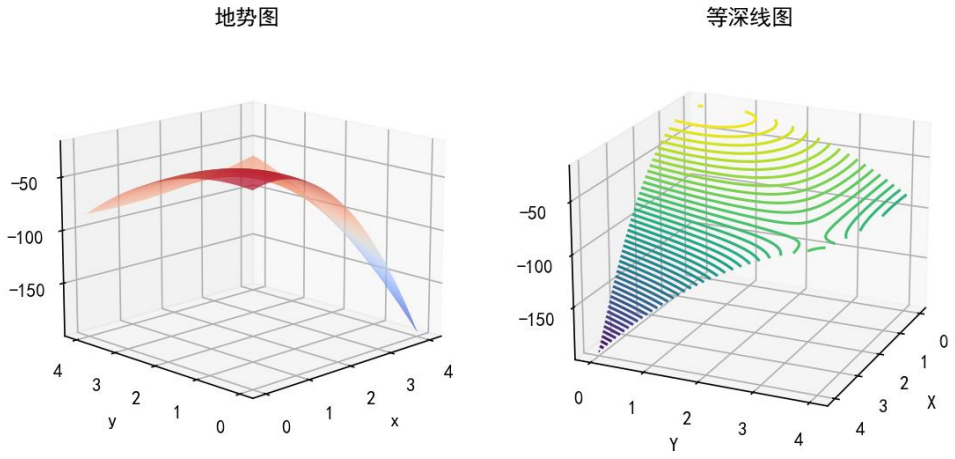


图 6 海底坡面立体图

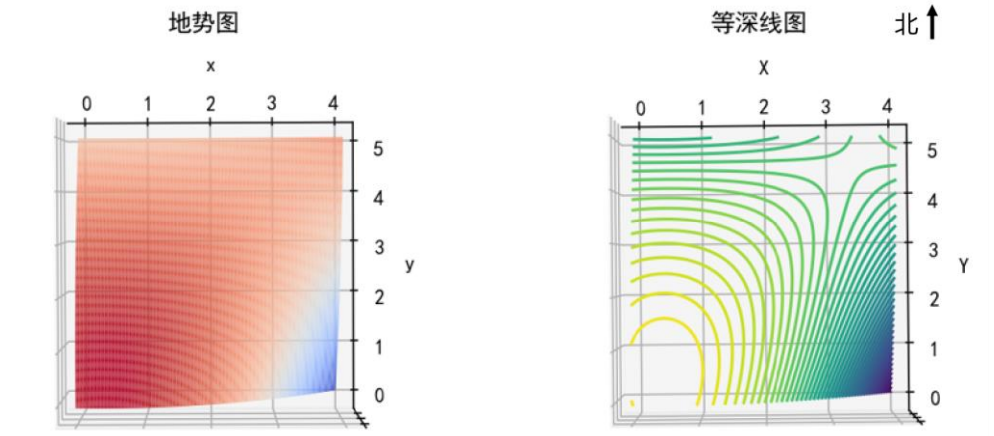


图 7 海底坡面俯视图

可以从图 6 图 7 中很明显地观察到，位于海域东南角的等深线较为密集，地势颜色也较为突出，表现出东南角海域海底深度以及水深的变化率都较大，需要考虑当到坡度大于30°时，不存在覆盖宽度，可以极大程度上节省测线的长度，因此对于该区域需要单独处理。而对于西南角海域，其海底地形较为平缓且海水深度最浅，等深线以声波的形状向其它四个方向辐射。西南角和东北角存在等深线环伺的情况，说明这两处海域存在一定的内凹或外凸特性。

为了更加清楚的了解当前海域的海底坡面细节，本文采取蒙特卡洛模拟进

行可视化分析，通过随机选取海域内的坐标点，通过被选取坐标点周围一定区域内的海域深度采集点数据进行海底坡面的拟合，因为沿着等高线航行有助于增大有效覆盖宽度，因此在每个采集点以坡面上平行于水平面的直线作为航行方向，在计算机中进行1000次蒙特卡洛模拟后，得到如下走向图：

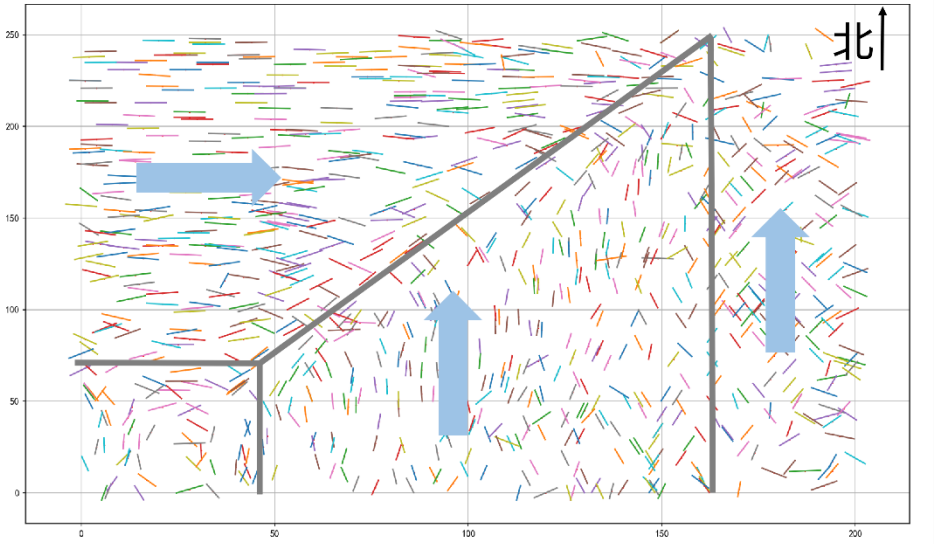


图 8 蒙特卡洛模拟走向图

通过走向图可以看出，在横轴采样点(50,150)区域内的下半部分采样点方向具有非常明显的一致性，而横轴采样点(150,200)区域内的采样点方向较为混杂。剩下的纵轴(100,250)采样点东西走向明显，而西南角内侧较为混杂，适合直线布线。

8.2 测线的初步布设

通过对海底坡面的分析，首先单独考虑东南角部位的海底坡度，通过 python 利用最小二乘法拟合出东南角区域的海底平面图，为了分析的可靠性，首先分析总体，再针对各个小平面进行分析，确保海底坡面在要求区域内的坡度是大于30°的。三维坐标系内对平面的最小二乘法拟合步骤如下：

step1. 将三维坐标系$[x,y,z]$函数映射到目标坐标系统$[x,y,z]$：

$$\boldsymbol{Y}+\boldsymbol{V}_{\boldsymbol{Y}}=(\boldsymbol{B}+\boldsymbol{V}_{\boldsymbol{B}})\mu\boldsymbol{M}+\boldsymbol{l}_n[\Delta x\ \Delta y\ \Delta z] \tag{19}$$

step2. 利用该映射的非线性 GM 模型：

$$\boldsymbol{l}+\boldsymbol{V}=\boldsymbol{f}(\boldsymbol{\beta}) \tag{20}$$

step3. 将问题转化为经典非线性模型最优化问题，求出偏导数的梯度、通过 BFGS 拟合出三维坐标系的最小二乘公式

$$\hat{\boldsymbol{\beta}}^{i+1}=\hat{\boldsymbol{\beta}}^{i}-\boldsymbol{T}^{i}g\left(\hat{\boldsymbol{\beta}}^{i}\right) \tag{21}$$

结合附件所给的数据，发现即便对于坡度最大的东南角也基本无坡度大于30°，因此直接对海域进行分区求解。结合图 8 进行以下初步布设：

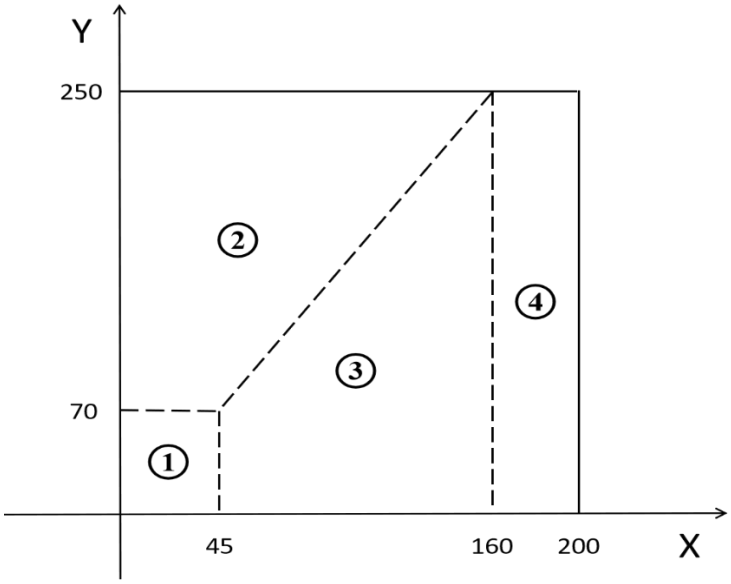


图 9 初步布设测线方向

8.3 改进过的鲸鱼优化算法

鲸鱼优化算法[3]是一种模拟座头鲸狩猎捕食行为的元启发式优化算法，该算法的核心是使用螺旋函数来模拟座头鲸的泡泡网觅食行为，优点具有寻优能力强，参数少等。

相比于原始的鲸鱼优化算法，改进过的鲸鱼优化算法具体在以下几点上进行了改动使其在局部求解上更加优秀：

（1） 为了减少初始随机解对最终结果的影响，加入了猎物逃跑机制。通过选择合适的蒙特卡洛模拟次数来选定最优的位置进行移动，可以很大程度的加大模型的全局寻优能力以及收敛速度。位置更新公式如下：

$$z=max\left(0.1,e^{-\frac{d_{min}}{Mc}}\right) \tag{22}$$

$$(t+1) = X^*(t) + Mc \times z \times m \times \left(1 - \frac{t}{t_{max}}\right) \times r \tag{23}$$

式中，$X^*(t)$是当前得到的最优解的位置，Mc是变量的限制范围长度，m为-1或1的随机变量，r为[0,1]的随机数，d_{min}是所有解中与当前最优解欧式距离最小值，$\overline{Mc}$是所有变量范围长度的平均值。

（2） 使a从2非线性降到0，在前期可以使算法生成较大的A，增加算法的全局寻优能力，中期加快算法的收敛速度，在后期可以使A偏小，增强局部寻优能力。[5]

$$a = \left(2 - \frac{2t}{t_{max}}\right)\left(1 - \left(\frac{t}{t_{max}}\right)^3\right) \tag{24}$$

（3） 加入自适应权重系数。[4]

（4） 从人工蜂群算法中得到启发，向本文的鲸鱼优化算法中引入$limit$阈值思想[4]，当所求解的情况出现大量相同不变时，会进行随机差分变异策略[7]更新当前解：

$$X(t+1) = r_1\left(X^*(t) - X(t)\right) + r_2\left(X_{rand}(t) - X(t)\right) \tag{25}$$

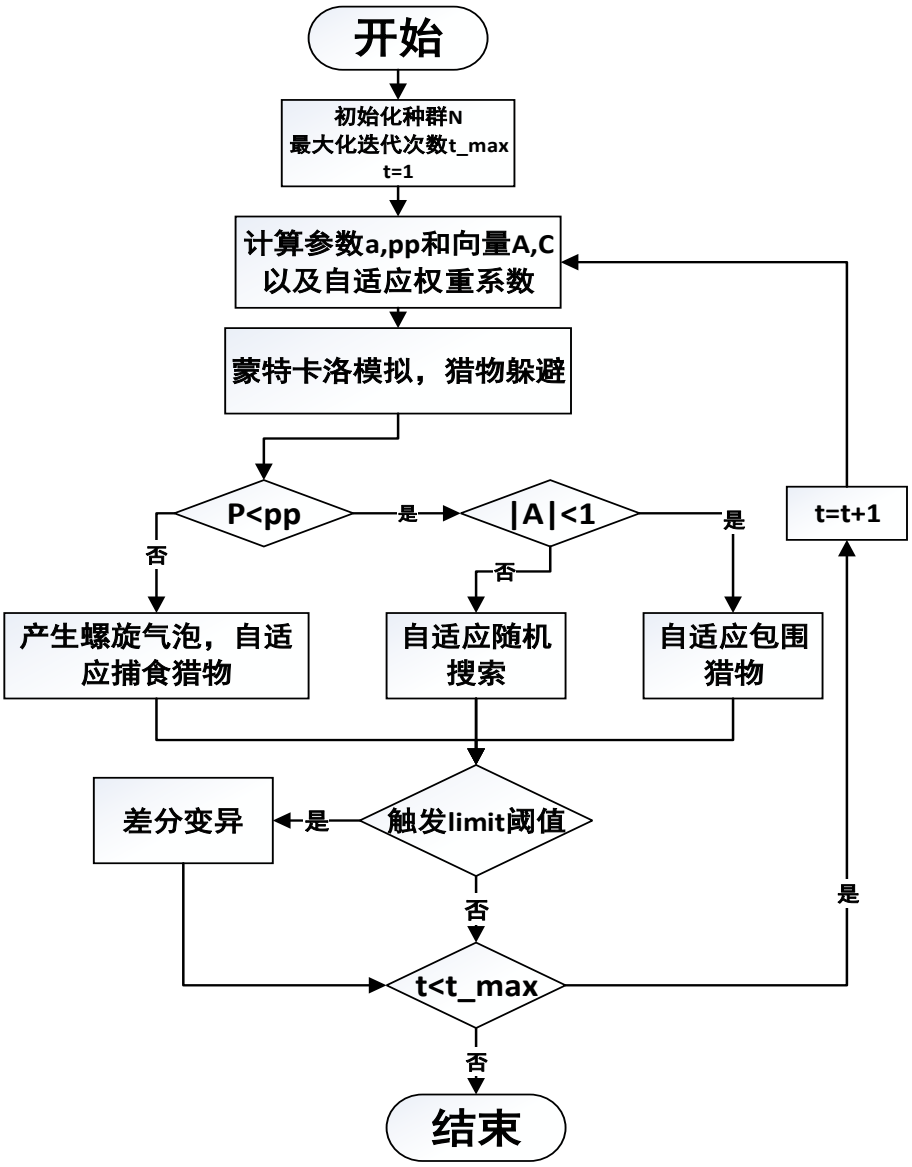
（5）采用了自适应p值[4]，优化了全局搜索能力。

$$pp = \begin{cases} 0.7 & ,t \le 0.5 * t_{max} \\ 0.4 & ,t > 0.5 * t_{max} \end{cases} \tag{26}$$

改进过的鲸鱼优化算法的最终表达式如下，在鲸鱼每次捕食前，猎物都会通过蒙特卡洛模拟进行躲避，增大初始解的优越性，然后根据不同迭代次数以不同的概率进行最优解位置的更新。

$$X(t+1) = \begin{cases} \left(\frac{t}{t_{max}}\right)^3 \cdot X^*(t) - A \times D & |A| < 1, p < pp \\ \left(\frac{t}{t_{max}}\right)^3 \cdot X_{rand}(t) - A \times D_{rand} & |A| \ge 1, p < pp \\ D' \times e^{bl} \times \cos(2\pi l) + \left(1 - \left(\frac{t}{t_{max}}\right)^3\right) \cdot X^*(t) & , p \ge pp \end{cases} \tag{27}$$

算法流程图大致如下：



8.4 确定最终的测线布设

结合 8.1 的海底坡面分析，可以将①③④区域的测线布设为南北方向，将②区域的测线布设为东西方向，然后针对各个区域进行求解。因为测线的方向基本固定，可以将问题 3 模型的迭代思想用于求解测线位置，用改进的鲸鱼算法求解符合约束条件的目标函数的最优值。

8.4.1 针对区域①与④进行求解

首先解决最左边的测线布设，将重叠率，覆盖区域等条件当作默认满足条件，即可通过优化算法解决目标函数的最优值问题，目标函数如下：

$$\min\left|d_1 - \min_w \frac{1}{2} w_{d_1}\right| \tag{28}$$

$$\min\left|d_n - 0.9\sum_{j=1}^{n} w_{d_j} - \min_w \frac{1}{2} w_{d_n}\right| \tag{29}$$

式中，w_{d_j}表示第d_j个测线的总覆盖宽度，$\min_w \frac{1}{2} w_{d_n}$表示第$d_n$个测线上所有位置的测量覆盖宽度中的最小值，这也是改进的鲸鱼算法最主要的目标，因为题目所给的数据为离散的，通过优化算法计算测线在拟合的平面的最小覆盖宽度。

对于这个问题中的重叠率，覆盖区域等约束条件，在目标含数中对于覆盖宽度的的最小值求解即实现了两个约束条件，而测线的总长度要求在分区处理时即有了部分约束。与问题 3 求解思想不同的是，在本文中因为海底坡面的变动性，对于测量船两侧的覆盖宽度不易区别，因此在本问中默认两边覆盖宽度相等，该决策不影响总覆盖宽度，所以对于总体影响不大。

8.4.2 针对区域②与③进行求解

区域②③与区域①④不同之处是前者的海底区域并不是矩形，而是梯形，因此在考虑代码求解时需要为不同的测线位置定制测线长度。本文采用正切函数对不同位置的测线上进行测线的长度求解，如下图所示：

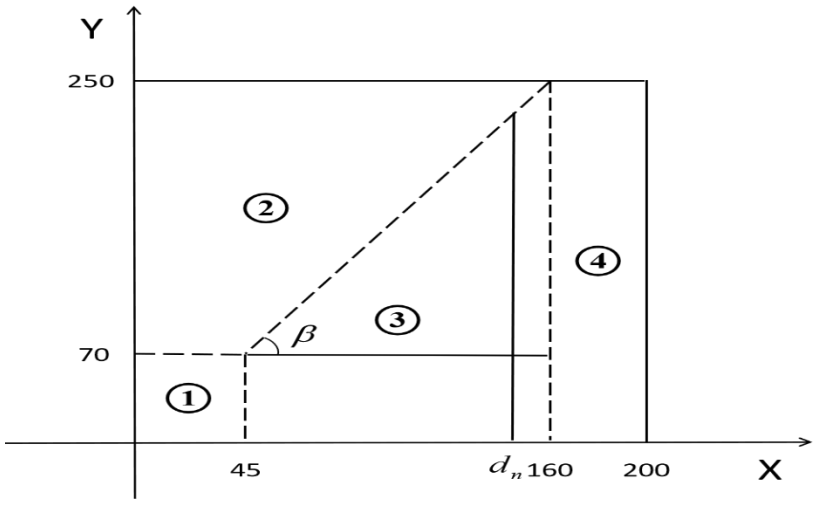


图 10 测线长度变换图

当测线位于d_n的位置时，测线的长度为β与d_n的组合式，如下所示：

$$l_n = 70 + (d_n - 45)\tan\beta \tag{30}$$

8.5 指标的求解

通过分区求解时只对起始边作了边距要求，对于终末区边的位置并没有考虑到另一个相邻分区的侧线布设情况，因此对四个分区的边界进行检查，发现①③分区交界处有一条测线多余，因此将其删去。由于设计测线布设时是严格要求测量覆盖宽度进行设计的，因此不存在漏测海区的情况。详细测线位置见附录，具体指标值如下表所示：

表 3 问题四指标值

指标类型	指标值
测线总长度/m	43734
漏测海区占总海域百分比	0
超额重叠率的测线长度/m	7408

九、 模型的评价及改进方向

9.1 模型的优点

（1） 问题 1，2，3 得到的结果均为精确解，精度及可信度较高。

（2） 问题 3 考虑到海域边缘的测量精度以及航船的航行路线，推广后的实用性较高。

（3） 问题 4 采用大量可视化分析，可信度较高。

9.2 模型的缺点及改进方向

（1） 问题 3 结合文献直接选择平行布线的方式，并未对其它方式做尝试，可能存在丢失最优解的情况，可以结合优化算法进行多方向测线的尝试比较。

（2） 问题 4 并未考虑船体两侧不同的覆盖区域，精度有所缺失。可以对采样点进行测线方向与坡面方向的综合分析，实时计算测线两边的覆盖宽度。

十、 参考文献

[1] 张旭,叶小心,洪德玫.多波束系统在长江航道测量中的测线布设方法研究[J].中国水运.航道科技,2017(01):52-55.DOI:10.19412/j.cnki.42-1395/u.2017.01.011.

[2] 张立华,殷晓冬.水深测量计划测线布设与航迹控制算法[J].海洋测绘,2002(02):33-35.

[3] Mirjalili, S. & Lewis, A. (2016). The Whale Optimization Algorithm. Advances in Engineering Software,95, 51-67.

[4] 何庆, 魏康园, 徐钦帅. 基于混合策略改进的鲸鱼优化算法[J]. 计算机应用研究, 2019, 36(12): 3647-3651,3665.

[5] 魏政磊, 赵辉, 李牧东, 等. 控制参数值非线性调整策略的灰狼优化算法[J]. 空军工程大学学报(自然科学版), 2016, 17(3): 68-72.

[6] 周敏, 李太勇. 粒子群优化算法中的惯性权值非线性调整策略[J]. 计算机工程, 2011, 37(5): 204-206.

[7] 武泽权,牟永敏.一种改进的鲸鱼优化算法[J].计算机应用研究,2020,37(12):3618-3621.

十一、 附录

11.1 问题三的结果展示

本问的结果是以水平面上垂直于坡面法向量的方向向量为 x 轴，距离单位为m。（先从左到右，再从上到小的顺序）

Table 1

290.05	889.04	1436.94	1942.05	2407.71	2837.01	3232.78
3597.64	3934.01	4244.11	4529.99	4793.54	5036.51	5260.51
5467.01	5657.39	5832.9	5994.7	6143.87	6281.39	6408.16
6525.04	6632.79	6732.12	6823.7	6908.12	6985.95	7057.71
7123.86	7184.84	7241.06	7292.89	7340.67	7384.72	7804

11.2 问题四的结果展示

以自西向东方向为 x 轴方向，距离单位为m，①③④区域的测线位置如下：

Table 2

39.97	112.15	183.38	252.7	322.04	388.74	454.88
520.81	583.58	649.14	712.11	774.42	836.83	900.28
962.65	1028.43	1091.83	1156.44	1220.69	1284.41	1351.11
1419.84	1489.62	1559.85	1633.29	1705.09	4842.22	6121.11

以自南向北方向为 y 轴方向，距离单位为m，②区域的测线位置如下：

Table 3

2632.2	7441.6

11.3 求解代码

附录 0：代码运行前库的导入（每串代码前均添加，文件无需操作）

附录一：问题 1 求解代码

附录二：问题 2 求解代码

附录三：问题 3 求解代码

附录四：问题 4 蒙特卡洛模拟

附录五：问题 4 第①区域求解

附录六：问题 4 第②区域求解

附录七：问题 4 第③区域求解

附录八：问题 4 第④区域求解

附录九：问题 4 指标值求解

附录十：问题 4 海底地形分析

附录 0：代码运行前库的导入

```
import numpy as np
import pandas as pd
from numpy import random
from copy import deepcopy
from scipy.optimize import least_squares
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import warnings
warnings.filterwarnings("ignore")
np.set_printoptions(threshold=np.inf) # 不省略输出
np.set_printoptions(suppress=True) #不以科学计数法输出
#显示所有列
pd.set_option('display.max_columns', None)
#显示所有行
pd.set_option('display.max_rows', None)
#设置 value 的显示长度为 100，默认为 50
pd.set_option('max_colwidth',100)
plt.rcParams['axes.unicode_minus'] = False #显示负号
plt.rcParams['font.family'] = ['sans-serif']
plt.rcParams['font.sans-serif'] = ['SimHei']  # 散点图标签可以显示中文
```

附录一：问题 1 求解代码

```
d = -800
for i in range(9):
    D = 70-d*np.tan(np.deg2rad(1.5))
    W = (np.sqrt(3)/(2*np.sin(np.deg2rad(28.5)))+np.sqrt(3)/
        (2*np.sin(np.deg2rad(31.5))))*D*np.cos(np.deg2rad(1.5))
    it = 1-(200/W)
    if i == 0:
        a = 0
    print(f'距离为{d}时，海水深度为：',round(D,2),
          '覆盖宽度为：',round(W,2),'与上一条测线的重叠率为：',round(a,2))
    a = it
    d += 200
```

附录二：问题 2 求解代码

```
theta = 90
podu = []
jiaodu = []
for i in range(8):
    fa1 = np.array([0, -1, 1/np.tan(np.deg2rad(1.5))]) # 海底坡面的法向量
    xian = np.array([np.cos(np.deg2rad(theta)),
                     np.sin(np.deg2rad(theta)), 0])  # 线的方向向量(投影)
    touying1 = (np.dot(xian, fa1) / np.linalg.norm(fa1)) * fa1 # 坡面上的投影
    co_si = (np.dot(touying1, xian) /  # 计算两个投影向量之间的夹角（弧度）
                    (np.linalg.norm(touying1) * np.linalg.norm(xian)))
    angle_r = np.arccos(co_si)
    angle_deg = np.degrees(angle_r)  # 将弧度转换为度数
    print(f"测线夹角为{theta-90}时，侧线平面坡度为：{90-angle_deg}°")
    podu.append(90-angle_deg)
    jiaodu.append(theta-90)
    theta += 45
print('--------侧线平面坡度夹角为侧线旋转 90 度时的侧线垂面坡度-------')

podu += podu
jiaodu += jiaodu

for i in range(8):
    d = 0
    dd = 0.3*1852
    if jiaodu[i]>=90 and jiaodu[i]<=270:
        xi = -1
    else:
        xi = 1
    print(f'----------测线夹角为{jiaodu[i]},测线垂面坡度为{podu[i+2]}时----------')
    print(f'----------测线夹角为{jiaodu[i]},测线竖直平面坡度为{podu[i]}时----------')
    for v in range(8):
        D = 120+d*np.tan(np.deg2rad(podu[i]))*xi
        W = (np.sqrt(3)/(2*np.sin(np.deg2rad(30-podu[i])))+np.sqrt(3)/
            (2*np.sin(np.deg2rad(30+podu[i]))))*D*np.cos(np.deg2rad(podu[i]))
        print(f'距离为{d}时','覆盖宽度为：',round(W,2))
        d += dd
```

附录三：问题 3 求解代码

```
D = 110+3704*np.tan(np.deg2rad(1.5)) # 最深的海水深度
loc = [] # 储存测线的横坐标
loc1 = [] # 储存 w1 的值
loc2 = [] # 储存 w2 的值
WW = [] # 储存已知的覆盖宽度

g3 = np.sqrt(3)
s28 = np.sin(np.deg2rad(28.5))
s31 = np.sin(np.deg2rad(31.5))
t1 = np.tan(np.deg2rad(1.5))
c1 = np.cos(np.deg2rad(1.5))

d = D*g3/(2*s28+t1*g3) # 最开始的测线位置
```

```
w1 = d
w2 = g3/(2*s31)*c1*(D-d*np.tan(np.deg2rad(1.5)))*np.cos(np.deg2rad(1.5))
W = w1+w2
d -= 0.1*W
WW.append(0.9*W) # 去除 10%的边缘不精准
loc.append(d)
loc1.append(w1)
loc2.append(w2)
while True:
    d = (D*g3+1.8*sum(WW)*s28)/(2*s28+t1*g3)
    if d >7408:
        break
    w1 = g3/(2*s28)*c1*(D-d*np.tan(np.deg2rad(1.5)))
    w2 = g3/(2*s31)*c1*(D-d*np.tan(np.deg2rad(1.5)))
    W = w1+w2
    WW.append(W)
    loc.append(d)
    loc1.append(w1)
    loc2.append(w2)

plt.vlines(loc,0,3704)
for i in range(len(loc)):
    plt.axvspan(loc[i]-loc1[i], loc[i]+loc2[i],ymin=0.05, ymax=0.95, alpha=0.1, color='red')
plt.show()

if 7408-loc[-1]>loc2[-1]:
    loc.append(7804)

print(loc) # 测线距离集合
print(loc1) # w1
print(loc2) # w2
print(WW) # W
```

附录四：问题 4 蒙特卡洛模拟

```
df = np.array(pd.read_excel('地理表.xlsx',header = None))
def ju_mian(df):  # 拟合海底坡面
    X = np.linspace(0, 0.02*1852*(len(df[0])-1)),len(df[0]))
    Y = np.linspace(0, 0.02*1852*(len(df[0])),len(df))
    Z = -df
    X, Y = np.meshgrid(X, Y)
    X = X.flatten()
    Y = Y.flatten()
    Z = Z.flatten()
    def nihe_plane_fit(xishuyouxiao, X, Y, Z):
        a, b, c, d = xishuyouxiao
        return a * X + b * Y + c * Z + d
    initial_guess = [1.1, 1.1, 1.1, 1.1]
    a, b, c, d = (least_squares(nihe_plane_fit, initial_guess, args=(X, Y, Z))).x
    return a,b,c,d
def yuansu(df): # 求解覆盖宽度
    a,b,c,d = ju_mian(df)  # 坡面方程的系数
    fa1 = np.array([a,b,c]) # 海底拟合坡面的法向量

    thetaa = np.array([a,b,0])
    co_si = (np.dot([1,0,0], thetaa) /  # 计算两个投影向量之间的夹角（弧度）
             (np.linalg.norm([0,0,1]) * np.linalg.norm(thetaa)))
```

```
    angle_r1 = np.arccos(co_si)
    angle_deg1 = 90-np.degrees(angle_r1)  # 坡面法向量投影于水平面与 x 轴的夹角

    xian = np.array([np.cos(np.deg2rad(angle_deg1+90)),
                     np.sin(np.deg2rad(angle_deg1+90)), 0])  # 测线的方向向量(投影)
    touying1 = (np.dot(xian, fa1) / np.linalg.norm(fa1)) * fa1 # 坡面上的投影
    co_si = (np.dot(touying1, xian) /  # 计算两个投影向量之间的夹角（弧度）
            (np.linalg.norm(touying1) * np.linalg.norm(xian)))
    angle_r = np.arccos(co_si)
    angle_deg2 = 90-np.degrees(angle_r)  # 将弧度转换为度数

    D = np.mean(df)  # 求得深度
    g3 = np.sqrt(3)
    s2 = np.sin(np.deg2rad(30-angle_deg2))
    s3 = np.sin(np.deg2rad(30+angle_deg2))
    c1 = np.cos(np.deg2rad(angle_deg2))
    w1 = g3/(2*s2)*c1*D  # 坡度较大则小于 0
    w2 = g3/(2*s3)*c1*D
    w = w1+w2
    return w,w1,w2,angle_deg1

for i in range(1000):
    xy = [random.randint(0,200),random.randint(0,250)]
    df = np.array([row[max(0, xy[0] - 4) : min(len(df[0]), xy[0] + 4 + 1)]
                   for row in df[max(0, xy[1] - 4) : min(len(df), xy[1] + 4 + 1)]]) # 前 x 后 y
    w,w1,w2,jiao = yuansu(df)
    print(w,w1,w2)
    print(jiao)
    x , y = xy[0],xy[1]
    angle = np.deg2rad(jiao)
    if np.abs(np.cos(angle)) < 1e-6:
        # 如果斜率不存在，绘制垂直于 x 轴的竖直线
        x_r = np.array([x, x])
        y_r = np.array([y - 4, y + 4])
    else:
        a = np.tan(angle)  # 直线斜率
        ii = np.cos(angle)
        x_r = np.linspace(x - 4 * ii, x + 4 * ii, 5)
        # 计算对应的 y 范围
        y_r = a * (x_r - x) + y
    df = np.array(pd.read_excel('地理表.xlsx',header = None))
    # 绘制直线或竖直线
    plt.plot(x_r, y_r)
    plt.scatter(x, y, color='red',s=0.5)
plt.grid(True)

# 显示图形
plt.show()
```

附录五：问题④第一区域求解

```
df = np.array(pd.read_excel('地理表.xlsx',header = None))
def ju_mian(df):  # 拟合海底坡面
    X = np.linspace(0, 0.02*1852*(len(df[0]-1)),len(df[0]))
    Y = np.linspace(0, 0.02*1852*(len(df[0])),len(df))
```

```
    Z = -df
    X, Y = np.meshgrid(X, Y)
    X = X.flatten()
    Y = Y.flatten()
    Z = Z.flatten()
    def nihe_plane_fit(xishuxiaoyong, X, Y, Z):
        a, b, c, d = xishuxiaoyong
        return a * X + b * Y + c * Z + d
    ini_g = [1.1, 1.1, 1.1, 1.1]
    result = least_squares(nihe_plane_fit, ini_g, args=(X, Y, Z))
    a, b, c, d = result.x
    return a,b,c,d
def yuansu(df): # 求解覆盖宽度
    a,b,c,d = ju_mian(df)  # 坡面方程的系数
    fa1 = np.array([a,b,c]) # 海底拟合坡面的法向量

    xian = np.array([0,1,0])  # 测线的方向向量(投影)

    touying1 = (np.dot(xian, fa1) / np.linalg.norm(fa1)) * fa1 # 坡面上的投影
    co_si = (np.dot(touying1, xian) /  # 计算两个投影向量之间的夹角（弧度）
            (np.linalg.norm(touying1) * np.linalg.norm(xian)))
    angle_r = np.arccos(co_si)
    angle_deg2 = 90-np.degrees(angle_r)  # 将弧度转换为度数

    D = np.mean(df)  # 求得深度
    g3 = np.sqrt(3)
    s2 = np.sin(np.deg2rad(30-angle_deg2))
    s3 = np.sin(np.deg2rad(30+angle_deg2))
    c1 = np.cos(np.deg2rad(angle_deg2))
    w1 = g3/(2*s2)*c1*D  # 坡度较大则小于 0
    w2 = g3/(2*s3)*c1*D
    w = w1+w2
    return w,w1,w2


def fun(X,www):  # 目标函数和约束条件
    x = X.flatten() #将 X 变为一维数组
    ww = []
    for i in range(1,6):
        data = np.array([row[int(max(0, x[0]-4)) : int(min(len(df[0]), x[0] +4+1))]
                    for row in df[int(max(0, x[i]-4)) : int(min(len(df), x[i] +4+1))]]) # 前 x 后 y
        w,w1,w2 = yuansu(data)
        ww .append(w)
    re = x[0]*37.04-0.5*np.mean(ww)-0.9*sum(www)
    if re > 0:
        return 10000
    else:
        return abs(re)

s = np.zeros((1,6))
sub = np.array([0,0,0,0,0,0]).ravel()  # 自变量下限
up = np.array([45,70,70,70,70,70]).ravel()  # 自变量上限
type = np.array(s-1).ravel()   #-1 是有理数，0 是整数，1 是 0-1 变量

def dd2(best_x, x):  #欧氏距离
    best_x = np.array(best_x)
    x = np.array(x)
```

```
    c = np.sum(pow(x - best_x, 2), axis=1)   #求方差，在行上的标准差
    d = pow(c, 0.5)  #标准差
    return d
def new_min(arr):  #求最小
    min_data = min(arr)  #找到最小值
    key = np.argmin(arr)  #找到最小值的索引
    return min_data, key
def type_x(xx,type,n):  #变量范围约束
    for v in range(n):
        if type[v] == -1:
            xx[v] = np.maximum(sub[v], xx[v])
            xx[v] = np.minimum(up[v], xx[v])
        elif type[v] == 0:
            xx[v] = np.maximum(sub[v], int(xx[v]))
            xx[v] = np.minimum(up[v], int(xx[v]))
        else:
            xx[v] = np.maximum(sub[v], random.randint(0,2))
            xx[v] = np.minimum(up[v], random.randint(0,2))
    return xx
def woa(sub,up,type,nums,det,www):
    n = len(sub)  # 自变量个数
    num = nums * n  # 种群大小
    x = np.zeros([num, n])  #生成保存解的矩阵
    f = np.zeros(num)  #生成保存值的矩阵
    for s in range(num):     #随机生成初始解
        rand_data = np.random.uniform(0,1)
        x[s, :] = sub + (up - sub) * rand_data
        x[s, :] = type_x(x[s, :],type,n)
        f[s] = fun(x[s, :],www)
    best_f, a = new_min(f)  # 记录历史最优值
    best_x = x[a, :]  # 记录历史最优解
    trace = np.array([deepcopy(best_f)]) #记录初始最优值,以便后期添加最优值画图
    ############################## 改进的鲸鱼算法 ##############################
    xx = np.zeros([num, n])
    ff = np.zeros(num)
    Mc = (up - sub) * 0.1  # 猎物行动最大范围
    for ii in range(det):     #设置迭代次数，进入迭代过程
        # 猎物躲避,蒙特卡洛模拟，并选择最佳的点作为下一逃跑点 #########！！！创新点
        d = dd2(best_x, x)  #记录当前解与最优解的距离
        d.sort()  #从小到大排序,d[0]恒为 0
        z = np.exp(-d[1] / np.mean(Mc))  # 猎物急躁系数
        z = max(z, 0.1)    #决定最终系数
        yx = []  #初始化存储函数值
        dx = []  #初始化存储解
        random_rand = random.random(n) #0-1 的随机数
        for i in range(10):   #蒙特卡洛模拟的次数
            m = [random.choice([-1, 1]) for _ in range(n)] #随机的-1 和 1
            asd = best_x + Mc * z * ((det-ii )/det) * random_rand * m   #最优解更新公式
            xd = type_x(asd,type,n)  #对自变量进行限制
            if i < 1:
                dx = deepcopy(xd)
            else:
                dx = np.vstack((dx,xd))   #存储每一次的解
            yx=np.hstack((yx,fun(xd,www)))   #存储每一次的值
        best_t, a = new_min(yx)  # 选择最佳逃跑点
```

```
        best_c = dx[a, :]  #最佳逃跑点
        if best_t < best_f:  #与鲸鱼算法得到的最优值对比
            best_f = best_t  #更新最优值
            best_x = best_c  #更新最优解
        ##############################鲸鱼追捕##############################
        w = (ii / det)**3  #自适应惯性权重!!!创新点
        a = (2 - 2*ii/det)*(1- w)  #a 随迭代次数从 2 非线性下降至 0！！！创新点
        pp=0.7 if ii <= 0.5*det else 0.4
        for i in range(num):
            r1 = np.random.rand()  # r1 为[0,1]之间的随机数
            r2 = np.random.rand()  # r2 为[0,1]之间的随机数
            A = 2 * a * r1 - a
            C = 2 * r2
            b = 1    #螺旋形状系数
            l = np.random.uniform(-1,1)  #参数 l
            p = np.random.rand()
            if p < pp:
                if abs(A) >= 1:
                    rand_leader = np.random.randint(0, num)
                    X_rand = x[rand_leader, :]
                    D_X_rand = abs(C * X_rand - x[i, :])
                    xx[i, :] = w*X_rand - A * D_X_rand
                    xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
                elif abs(A) < 1:
                    D_Leader = abs(C * best_x - x[i, :])
                    xx[i, :] = w*best_x - A * D_Leader
                    xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
            elif p >= pp:
                D = abs(best_x - x[i, :])
                xx[i, :] = D*np.exp(b*l)*np.cos(2*np.pi*l) + (1-w)*best_x  #完整的气泡网捕食公式
                xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
            ff[i] = fun(xx[i, :],www)
            if len(np.unique(ff[:i]))/(i+1) <= 0.1:    #limit 阈值 + 随机差分变异！！！创新点
                xx[i,:] = (r1*(best_x-xx[i,:]) +
                           r2*(x[np.random.randint(0,num),:] - xx[i,:]))
                xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
                ff[i] = fun(xx[i, :],www)
        #将上一代种群与这一代种群以及最优种群结合，选取排名靠前的个体组成新的种群
        F = np.hstack((np.array([best_f]), f, ff))
        F, b = np.sort(F,axis=-1,kind='stable'), np.argsort(F)#按小到大排序,获得靠前的位置
        X = np.vstack(([best_x], x, xx))[b, :]
        f = F[:num]  #新种群的位置
        x = X[:num, :]  #新种群的位置
        best_f, a = new_min(f)  # 记录历史最优值
        best_x = x[a , :]  # 记录历史最优解
        trace = np.hstack((trace, [best_f]))
        if ii % 10 == 0:
            print('迭代次数：',ii)
    return best_x,best_f,trace

bx = []
bf = []
www = [0]
while True:
    best_x,best_f,trace = woa(sub,up,type,3,30,www)    #种群大小，迭代次数
    if best_x[0] >= 45:
```

```
        break
    print('最优解为：')
    print(best_x[0]*37.04) # 转化为 m
    print('最优值为：')
    print(float(best_f*37.04))
    kuan = 2*(best_f + best_x[0]*37.04 - 0.9*sum(www))
    www.append(kuan)
    print('覆盖宽度：',kuan)
    bx.append(best_x[0]*37.04)
    bf.append(best_f*37.04)

print(bx)
print(bf)
print(www[1:])
```

附录六：问题四第②区域的求解

```
df = np.array(pd.read_excel('地理表.xlsx',header = None))
def ju_mian(df):  # 拟合海底坡面
    X = np.linspace(0, 0.02*1852*(len(df[0])-1)),len(df[0]))
    Y = np.linspace(0, 0.02*1852*(len(df[0])),len(df))
    Z = -df
    X, Y = np.meshgrid(X, Y)
    X = X.flatten()
    Y = Y.flatten()
    Z = Z.flatten()
    def plane_fit(coefficients, X, Y, Z):
        a, b, c, d = coefficients
        return a * X + b * Y + c * Z + d
    initial_guess = [1.0, 1.0, 1.0, 1.0]
    result = least_squares(plane_fit, initial_guess, args=(X, Y, Z))
    a, b, c, d = result.x
    return a,b,c,d
def yuansu(df): # 求解覆盖宽度
    a,b,c,d = ju_mian(df)  # 坡面方程的系数
    fa1 = np.array([a,b,c]) # 海底拟合坡面的法向量

    xian = np.array([0,1,0])  # 测线的方向向量(投影)

    touying1 = (np.dot(xian, fa1) / np.linalg.norm(fa1)) * fa1 # 坡面上的投影
    co_si = (np.dot(touying1, xian) /  # 计算两个投影向量之间的夹角（弧度）
            (np.linalg.norm(touying1) * np.linalg.norm(xian)))
    angle_r = np.arccos(co_si)
    angle_deg2 = 90-np.degrees(angle_r)  # 将弧度转换为度数
    #print(f"测线方向为{theta}度时，侧线垂面坡度为：{round(angle_deg2,4)}°")

    D = np.mean(df)  # 求得深度
    g3 = np.sqrt(3)
    s2 = np.sin(np.deg2rad(30-angle_deg2))
    s3 = np.sin(np.deg2rad(30+angle_deg2))
    c1 = np.cos(np.deg2rad(angle_deg2))
    w1 = g3/(2*s2)*c1*D  # 坡度较大则小于 0
    w2 = g3/(2*s3)*c1*D
    w = w1+w2
    return w,w1,w2
```

```
def fun(X,www):  # 目标函数和约束条件
    x = X.flatten() #将 X 变为一维数组
    ww = []
    shuti = 1 + (23*(x[0]-70))/1620
    for i in range(1,6):
        data = np.array([row[int(max(0, x[i]*shuti-4)) : int(min(len(df[0]), x[i]*shuti+4+1))]
                    for row in df[int(max(0, x[0]-4)) : int(min(len(df), x[0]+4+1))]]) # 前 x 后 y
        w,w1,w2 = yuansu(data)
        ww .append(w)
    re = (x[0]-70)*37.04-0.5*np.mean(ww)-0.9*sum(www)
    if re > 0:
        return 10000
    else:
        return abs(re)

s = np.zeros((1,6))
sub = np.array([70,0,0,0,0,0]).ravel()  # 自变量下限
up = np.array([250,45,45,45,45,45]).ravel()  # 自变量上限
type = np.array(s-1).ravel()   #-1 是有理数，0 是整数，1 是 0-1 变量

def dd2(best_x, x):  #欧氏距离
    best_x = np.array(best_x)
    x = np.array(x)
    c = np.sum(pow(x - best_x, 2), axis=1)   #求方差，在行上的标准差
    d = pow(c, 0.5)   #标准差
    return d
def new_min(arr):  #求最小
    min_data = min(arr)   #找到最小值
    key = np.argmin(arr)  #找到最小值的索引
    return min_data, key
def type_x(xx,type,n):  #变量范围约束
    for v in range(n):
        if type[v] == -1:
            xx[v] = np.maximum(sub[v], xx[v])
            xx[v] = np.minimum(up[v], xx[v])
        elif type[v] == 0:
            xx[v] = np.maximum(sub[v], int(xx[v]))
            xx[v] = np.minimum(up[v], int(xx[v]))
        else:
            xx[v] = np.maximum(sub[v], random.randint(0,2))
            xx[v] = np.minimum(up[v], random.randint(0,2))
    return xx
def woa(sub,up,type,nums,det,www):
    n = len(sub)  # 自变量个数
    num = nums * n  # 种群大小
    x = np.zeros([num, n])  #生成保存解的矩阵
    f = np.zeros(num)   #生成保存值的矩阵
    for s in range(num):     #随机生成初始解
        rand_data = np.random.uniform(0,1)
        x[s, :] = sub + (up - sub) * rand_data
        x[s, :] = type_x(x[s, :],type,n)
        f[s] = fun(x[s, :],www)
    best_f, a = new_min(f)  # 记录历史最优值
    best_x = x[a, :]  # 记录历史最优解
    trace = np.array([deepcopy(best_f)]) #记录初始最优值,以便后期添加最优值画图
    ############################## 改进的鲸鱼算法 ##########################
```

```
    xx = np.zeros([num, n])
    ff = np.zeros(num)
    Mc = (up - sub) * 0.1  # 猎物行动的最大范围
    for ii in range(det):     #设置迭代次数，进入迭代过程
        # 猎物躲避,蒙特卡洛模拟，并选择最佳的点作为下一逃跑点 #########！！！创新点
        d = dd2(best_x, x)  #记录当前解与最优解的距离
        d.sort()  #从小到大排序,d[0]恒为 0
        z = np.exp(-d[1] / np.mean(Mc))  # 猎物急躁系数
        z = max(z, 0.1)     #决定最终系数
        yx = []  #初始化存储函数值
        dx = []  #初始化存储解
        random_rand = random.random(n) #0-1 的随机数
        for i in range(10):    #蒙特卡洛模拟的次数
            m = [random.choice([-1, 1]) for _ in range(n)] #随机的-1 和 1
            asd = best_x + Mc * z * ((det-ii )/det) * random_rand * m   #最优解更新公式
            xd = type_x(asd,type,n)  #对自变量进行限制
            if i < 1:
                dx = deepcopy(xd)
            else:
                dx = np.vstack((dx,xd))   #存储每一次的解
            yx=np.hstack((yx,fun(xd,www)))    #存储每一次的值
        best_t, a = new_min(yx)  # 选择最佳逃跑点
        best_c = dx[a, :]  #最佳逃跑点
        if best_t < best_f:  #与鲸鱼算法得到的最优值对比
            best_f = best_t   #更新最优值
            best_x = best_c   #更新最优解
        ############################### 鲸鱼追捕 ##########################
        w = (ii / det)**3   #自适应惯性权重!!!创新点
        a = (2 - 2*ii/det)*(1- w)  #a 随迭代次数从 2 非线性下降至 0！！！创新点
        pp=0.7 if ii <= 0.5*det else 0.4
        for i in range(num):
            r1 = np.random.rand()  # r1 为[0,1]之间的随机数
            r2 = np.random.rand()  # r2 为[0,1]之间的随机数
            A = 2 * a * r1 - a
            C = 2 * r2
            b = 1     #螺旋形状系数
            l = np.random.uniform(-1,1)  #参数 l
            p = np.random.rand()
            if p < pp:
                if abs(A) >= 1:
                    rand_leader = np.random.randint(0, num)
                    X_rand = x[rand_leader, :]
                    D_X_rand = abs(C * X_rand - x[i, :])
                    xx[i, :] = w*X_rand - A * D_X_rand
                    xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
                elif abs(A) < 1:
                    D_Leader = abs(C * best_x - x[i, :])
                    xx[i, :] = w*best_x - A * D_Leader
                    xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
            elif p >= pp:
                D = abs(best_x - x[i, :])
                xx[i, :] = D*np.exp(b*l)*np.cos(2*np.pi*l) + (1-w)*best_x   #完整的气泡网捕食公式
                xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
            ff[i] = fun(xx[i, :],www)
            if len(np.unique(ff[:i]))/(i+1) <= 0.1:     #limit 阈值 + 随机差分变异！！！创新点
```

```
            xx[i,:] = (r1*(best_x-xx[i,:]) +
                    r2*(x[np.random.randint(0,num),:] - xx[i,:]))
            xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
            ff[i] = fun(xx[i, :],www)
        #将上一代种群与这一代种群以及最优种群结合，选取排名靠前的个体组成新的种群
        F = np.hstack((np.array([best_f]), f, ff))
        F, b = np.sort(F,axis=-1,kind='stable'), np.argsort(F)#按小到大排序,获得靠前的位置
        X = np.vstack(([best_x], x, xx))[b, :]
        f = F[:num]  #新种群的位置
        x = X[:num, :]  #新种群的位置
        best_f, a = new_min(f)  # 记录历史最优值
        best_x = x[a , :]  # 记录历史最优解
        trace = np.hstack((trace, [best_f]))
        if ii % 10 == 0:
            print('迭代次数：',ii)
    return best_x,best_f,trace

bx = []
bf = []
www = [0]
while True:
    best_x,best_f,trace = woa(sub,up,type,3,30,www)    #种群大小，迭代次数
    if best_x[0] >= 250:
        break
    #种群大小可以为自变量个数，迭代次数看情况
    print('最优解为：')
    print(best_x[0]*37.04) # 转化为 m
    print('最优值为：')
    print(float(best_f*37.04))
    kuan = 2*(best_f + best_x[0]*37.04 - 0.9*sum(www))
    www.append(kuan)
    print('覆盖宽度：',kuan)
    bx.append(best_x[0]*37.04)
    bf.append(best_f*37.04)

print(bx)
print(bf)
print(www[1:])
```

附录七：第③区域的求解

```
df = np.array(pd.read_excel('地理表.xlsx',header = None))
def ju_mian(df):  # 拟合海底坡面
    X = np.linspace(0, 0.02*1852*(len(df[0]-1)),len(df[0]))
    Y = np.linspace(0, 0.02*1852*(len(df[0])),len(df))
    Z = -df
    X, Y = np.meshgrid(X, Y)
    X = X.flatten()
    Y = Y.flatten()
    Z = Z.flatten()
    def plane_fit(coefficients, X, Y, Z):
        a, b, c, d = coefficients
        return a * X + b * Y + c * Z + d
    initial_guess = [1.0, 1.0, 1.0, 1.0]
    result = least_squares(plane_fit, initial_guess, args=(X, Y, Z))
    a, b, c, d = result.x
```

```
    return a,b,c,d
def yuansu(df): # 求解覆盖宽度
    a,b,c,d = ju_mian(df)  # 坡面方程的系数
    fa1 = np.array([a,b,c]) # 海底拟合坡面的法向量

    xian = np.array([0,1,0])  # 测线的方向向量(投影)

    touying1 = (np.dot(xian, fa1) / np.linalg.norm(fa1)) * fa1 # 坡面上的投影
    co_si = (np.dot(touying1, xian) /  # 计算两个投影向量之间的夹角（弧度）
            (np.linalg.norm(touying1) * np.linalg.norm(xian)))
    angle_r = np.arccos(co_si)
    angle_deg2 = 90-np.degrees(angle_r)  # 将弧度转换为度数
    #print(f"测线方向为{theta}度时，侧线垂面坡度为：{round(angle_deg2,4)}°")

    D = np.mean(df)  # 求得深度
    g3 = np.sqrt(3)
    s2 = np.sin(np.deg2rad(30-angle_deg2))
    s3 = np.sin(np.deg2rad(30+angle_deg2))
    c1 = np.cos(np.deg2rad(angle_deg2))
    w1 = g3/(2*s2)*c1*D  # 坡度较大则小于 0
    w2 = g3/(2*s3)*c1*D
    w = w1+w2
    return w,w1,w2


def fun(X,www):  # 目标函数和约束条件
    x = X.flatten() #将 X 变为一维数组
    ww = []
    shuti = 1 + (36*(x[0]-45))/1610
    for i in range(1,6):
        data = np.array([row[int(max(0, x[0]-4)) : int(min(len(df[0]), x[0] +4+1))]
                    for row in df[int(max(0, x[i]*shuti-4)) : int(min(len(df), x[i]*shuti +4+1))]]) # 前
x 后 y
        w,w1,w2 = yuansu(data)
        ww .append(w)
    re = (x[0]-45)*37.04-0.5*np.mean(ww)-0.9*sum(www)
    if re > 0:
        return 10000
    else:
        return abs(re)

s = np.zeros((1,6))
sub = np.array([45,0,0,0,0,0]).ravel()  # 自变量下限
up = np.array([160,70,70,70,70,70]).ravel()  # 自变量上限
type = np.array(s-1).ravel()   #-1 是有理数，0 是整数，1 是 0-1 变量

def dd2(best_x, x):  #欧氏距离
    best_x = np.array(best_x)
    x = np.array(x)
    c = np.sum(pow(x - best_x, 2), axis=1)   #求方差，在行上的标准差
    d = pow(c, 0.5)   #标准差
    return d
def new_min(arr):  #求最小
    min_data = min(arr)   #找到最小值
    key = np.argmin(arr)  #找到最小值的索引
    return min_data, key
```

```
def type_x(xx,type,n):  #变量范围约束
    for v in range(n):
        if type[v] == -1:
            xx[v] = np.maximum(sub[v], xx[v])
            xx[v] = np.minimum(up[v], xx[v])
        elif type[v] == 0:
            xx[v] = np.maximum(sub[v], int(xx[v]))
            xx[v] = np.minimum(up[v], int(xx[v]))
        else:
            xx[v] = np.maximum(sub[v], random.randint(0,2))
            xx[v] = np.minimum(up[v], random.randint(0,2))
    return xx
def woa(sub,up,type,nums,det,www):
    n = len(sub)  # 自变量个数
    num = nums * n  # 种群大小
    x = np.zeros([num, n])  #生成保存解的矩阵
    f = np.zeros(num)   #生成保存值的矩阵
    for s in range(num):     #随机生成初始解
        rand_data = np.random.uniform(0,1)
        x[s, :] = sub + (up - sub) * rand_data
        x[s, :] = type_x(x[s, :],type,n)
        f[s] = fun(x[s, :],www)
    best_f, a = new_min(f)  # 记录历史最优值
    best_x = x[a, :]  # 记录历史最优解
    trace = np.array([deepcopy(best_f)]) #记录初始最优值,以便后期添加最优值画图
    ############################## 改进的鲸鱼算法 ############################
    xx = np.zeros([num, n])
    ff = np.zeros(num)
    Mc = (up - sub) * 0.1  # 猎物行动最大范围
    for ii in range(det):     #设置迭代次数，进入迭代过程
        # 猎物躲避,蒙特卡洛模拟，并选择最佳的点作为下一逃跑点 #########！！！创新点
        d = dd2(best_x, x)  #记录当前解与最优解的距离
        d.sort()  #从小到大排序,d[0]恒为 0
        z = np.exp(-d[1] / np.mean(Mc))  # 猎物急躁系数
        z = max(z, 0.1)    #决定最终系数
        yx = []  #初始化存储函数值
        dx = []  #初始化存储解
        random_rand = random.random(n) #0-1 的随机数
        for i in range(10):   #蒙特卡洛模拟的次数
            m = [random.choice([-1, 1]) for _ in range(n)] #随机的-1 和 1
            asd = best_x + Mc * z * ((det-ii )/det) * random_rand * m   #最优解更新公式
            xd = type_x(asd,type,n)  #对自变量进行限制
            if i < 1:
                dx = deepcopy(xd)
            else:
                dx = np.vstack((dx,xd))   #存储每一次的解
            yx=np.hstack((yx,fun(xd,www)))    #存储每一次的值
        best_t, a = new_min(yx)  # 选择最佳逃跑点
        best_c = dx[a, :]   #最佳逃跑点
        if best_t < best_f:   #与鲸鱼算法得到的最优值对比
            best_f = best_t   #更新最优值
            best_x = best_c   #更新最优解
        ############################## 鲸鱼追捕 ############################
        w = (ii / det)**3   #自适应惯性权重!!!创新点
        a = (2 - 2*ii/det)*(1- w)  #a 随迭代次数从 2 非线性下降至 0！！！创新点
```

```
        pp=0.7 if ii <= 0.5*det else 0.4
        for i in range(num):
            r1 = np.random.rand()  # r1 为[0,1]之间的随机数
            r2 = np.random.rand()  # r2 为[0,1]之间的随机数
            A = 2 * a * r1 - a
            C = 2 * r2
            b = 1    #螺旋形状系数
            l = np.random.uniform(-1,1)  #参数 l
            p = np.random.rand()
            if p < pp:
                if abs(A) >= 1:
                    rand_leader = np.random.randint(0, num)
                    X_rand = x[rand_leader, :]
                    D_X_rand = abs(C * X_rand - x[i, :])
                    xx[i, :] = w*X_rand - A * D_X_rand
                    xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
                elif abs(A) < 1:
                    D_Leader = abs(C * best_x - x[i, :])
                    xx[i, :] = w*best_x - A * D_Leader
                    xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
            elif p >= pp:
                D = abs(best_x - x[i, :])
                xx[i, :] = D*np.exp(b*l)*np.cos(2*np.pi*l) + (1-w)*best_x   #完整的气泡网捕食公式
                xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
            ff[i] = fun(xx[i, :],www)
            if len(np.unique(ff[:i]))/(i+1) <= 0.1:    #limit 阈值 + 随机差分变异！！！创新点
                xx[i,:] = (r1*(best_x-xx[i,:]) +
                           r2*(x[np.random.randint(0,num),:] - xx[i,:]))
                xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
                ff[i] = fun(xx[i, :],www)
        #将上一代种群与这一代种群以及最优种群结合，选取排名靠前的个体组成新的种群
        F = np.hstack((np.array([best_f]), f, ff))
        F, b = np.sort(F,axis=-1,kind='stable'), np.argsort(F)#按小到大排序,获得靠前的位置
        X = np.vstack(([best_x], x, xx))[b, :]
        f = F[:num]  #新种群的位置
        x = X[:num, :]  #新种群的位置
        best_f, a = new_min(f)  # 记录历史最优值
        best_x = x[a , :]  # 记录历史最优解
        trace = np.hstack((trace, [best_f]))
        if ii % 10 == 0:
            print('迭代次数：',ii)
    return best_x,best_f,trace

bx = []
bf = []
www = [0]
while True:
    best_x,best_f,trace = woa(sub,up,type,3,30,www)    #种群大小，迭代次数
    if best_x[0] >= 160:
        break
    #种群大小可以为自变量个数，迭代次数看情况
    print('最优解为：')
    print(best_x[0]*37.04) # 转化为 m
    print('最优值为：')
    print(float(best_f*37.04))
    kuan = 2*(best_f + best_x[0]*37.04 - 0.9*sum(www))
    www.append(kuan)
```

```
    print('覆盖宽度：',kuan)
    bx.append(best_x[0]*37.04)
    bf.append(best_f*37.04)

print(bx)
print(bf)
print(www[1:])
```

附录八：问题四第④区域求解

```
df = np.array(pd.read_excel('地理表.xlsx',header = None))
def ju_mian(df):  # 拟合海底坡面
    X = np.linspace(0, 0.02*1852*(len(df[0])-1)),len(df[0]))
    Y = np.linspace(0, 0.02*1852*(len(df[0])),len(df))
    Z = -df
    X, Y = np.meshgrid(X, Y)
    X = X.flatten()
    Y = Y.flatten()
    Z = Z.flatten()
    def plane_fit(coefficients, X, Y, Z):
        a, b, c, d = coefficients
        return a * X + b * Y + c * Z + d
    initial_guess = [1.0, 1.0, 1.0, 1.0]
    result = least_squares(plane_fit, initial_guess, args=(X, Y, Z))
    a, b, c, d = result.x
    return a,b,c,d
def yuansu(df): # 求解覆盖宽度
    a,b,c,d = ju_mian(df)  # 坡面方程的系数
    fa1 = np.array([a,b,c]) # 海底拟合坡面的法向量

    xian = np.array([0,1,0])  # 测线的方向向量(投影)

    touying1 = (np.dot(xian, fa1) / np.linalg.norm(fa1)) * fa1 # 坡面上的投影
    co_si = (np.dot(touying1, xian) /  # 计算两个投影向量之间的夹角（弧度）
            (np.linalg.norm(touying1) * np.linalg.norm(xian)))
    angle_r = np.arccos(co_si)
    angle_deg2 = 90-np.degrees(angle_r)  # 将弧度转换为度数
    #print(f"测线方向为{theta}度时，侧线垂面坡度为：{round(angle_deg2,4)}°")

    D = np.mean(df)  # 求得深度
    g3 = np.sqrt(3)
    s2 = np.sin(np.deg2rad(30-angle_deg2))
    s3 = np.sin(np.deg2rad(30+angle_deg2))
    c1 = np.cos(np.deg2rad(angle_deg2))
    w1 = g3/(2*s2)*c1*D  # 坡度较大则小于 0
    w2 = g3/(2*s3)*c1*D
    w = w1+w2
    return w,w1,w2


def fun(X,www):  # 目标函数和约束条件
    x = X.flatten() #将 X 变为一维数组
    ww = []
    for i in range(1,6):
        data = np.array([row[int(max(0, x[0]-4)) : int(min(len(df[0]), x[0] +4+1))]
                         for row in df[int(max(0, x[i]-4)) : int(min(len(df), x[i] +4+1))]]) # 前 x 后 y
```

```
        w,w1,w2 = yuansu(data)
        ww .append(w)
    re = (x[0]-160)*37.04-0.5*np.mean(ww)-0.9*sum(www)
    if re > 0:
        return 10000
    else:
        return abs(re)

s = np.zeros((1,6))
sub = np.array([160,0,0,0,0,0]).ravel()  # 自变量下限
up = np.array([200,250,250,250,250,250]).ravel()  # 自变量上限
type = np.array(s-1).ravel()   #-1 是有理数，0 是整数，1 是 0-1 变量

def dd2(best_x, x):  #欧氏距离
    best_x = np.array(best_x)
    x = np.array(x)
    c = np.sum(pow(x - best_x, 2), axis=1)   #求方差，在行上的标准差
    d = pow(c, 0.5)  #标准差
    return d
def new_min(arr):  #求最小
    min_data = min(arr)  #找到最小值
    key = np.argmin(arr)  #找到最小值的索引
    return min_data, key
def type_x(xx,type,n):  #变量范围约束
    for v in range(n):
        if type[v] == -1:
            xx[v] = np.maximum(sub[v], xx[v])
            xx[v] = np.minimum(up[v], xx[v])
        elif type[v] == 0:
            xx[v] = np.maximum(sub[v], int(xx[v]))
            xx[v] = np.minimum(up[v], int(xx[v]))
        else:
            xx[v] = np.maximum(sub[v], random.randint(0,2))
            xx[v] = np.minimum(up[v], random.randint(0,2))
    return xx
def woa(sub,up,type,nums,det,www):
    n = len(sub)  # 自变量个数
    num = nums * n  # 种群大小
    x = np.zeros([num, n])  #生成保存解的矩阵
    f = np.zeros(num)  #生成保存值的矩阵
    for s in range(num):     #随机生成初始解
        rand_data = np.random.uniform(0,1)
        x[s, :] = sub + (up - sub) * rand_data
        x[s, :] = type_x(x[s, :],type,n)
        f[s] = fun(x[s, :],www)
    best_f, a = new_min(f)  # 记录历史最优值
    best_x = x[a, :]  # 记录历史最优解
    trace = np.array([deepcopy(best_f)]) #记录初始最优值,以便后期添加最优值画图
    ############################## 改进的鲸鱼算法 ############################
    xx = np.zeros([num, n])
    ff = np.zeros(num)
    Mc = (up - sub) * 0.1  # 猎物行动最大范围
    for ii in range(det):     #设置迭代次数，进入迭代过程
        # 猎物躲避,蒙特卡洛模拟，并选择最佳的点作为下一逃跑点 #########！！！创新点
        d = dd2(best_x, x)  #记录当前解与最优解的距离
        d.sort()  #从小到大排序,d[0]恒为 0
```

```
        z = np.exp(-d[1] / np.mean(Mc))  # 猎物急躁系数
        z = max(z, 0.1)    #决定最终系数
        yx = []  #初始化存储函数值
        dx = []  #初始化存储解
        random_rand = random.random(n) #0-1 的随机数
        for i in range(10):    #蒙特卡洛模拟的次数
            m = [random.choice([-1, 1]) for _ in range(n)] #随机的-1 和 1
            asd = best_x + Mc * z * ((det-ii )/det) * random_rand * m   #最优解更新公式
            xd = type_x(asd,type,n)  #对自变量进行限制
            if i < 1:
                dx = deepcopy(xd)
            else:
                dx = np.vstack((dx,xd))   #存储每一次的解
            yx=np.hstack((yx,fun(xd,www)))    #存储每一次的值
        best_t, a = new_min(yx)  # 选择最佳逃跑点
        best_c = dx[a, :]   #最佳逃跑点
        if best_t < best_f:   #与鲸鱼算法得到的最优值对比
            best_f = best_t   #更新最优值
            best_x = best_c   #更新最优解
        ############################## 鲸鱼追捕 ########################
        w = (ii / det)**3   #自适应惯性权重!!!创新点
        a = (2 - 2*ii/det)*(1- w)  #a 随迭代次数从 2 非线性下降至 0！！！创新点
        pp=0.7 if ii <= 0.5*det else 0.4
        for i in range(num):
            r1 = np.random.rand()  # r1 为[0,1]之间的随机数
            r2 = np.random.rand()  # r2 为[0,1]之间的随机数
            A = 2 * a * r1 - a
            C = 2 * r2
            b = 1    #螺旋形状系数
            l = np.random.uniform(-1,1)  #参数 l
            p = np.random.rand()
            if p < pp:
                if abs(A) >= 1:
                    rand_leader = np.random.randint(0, num)
                    X_rand = x[rand_leader, :]
                    D_X_rand = abs(C * X_rand - x[i, :])
                    xx[i, :] = w*X_rand - A * D_X_rand
                    xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
                elif abs(A) < 1:
                    D_Leader = abs(C * best_x - x[i, :])
                    xx[i, :] = w*best_x - A * D_Leader
                    xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
            elif p >= pp:
                D = abs(best_x - x[i, :])
                xx[i, :] = D*np.exp(b*l)*np.cos(2*np.pi*l) + (1-w)*best_x   #完整的气泡网捕食公式
                xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
            ff[i] = fun(xx[i, :],www)
            if len(np.unique(ff[:i]))/(i+1) <= 0.1:    #limit 阈值 + 随机差分变异！！！创新点
                xx[i,:] = (r1*(best_x-xx[i,:]) +
                           r2*(x[np.random.randint(0,num),:] - xx[i,:]))
                xx[i, :] = type_x(xx[i, :],type,n) #对自变量进行限制
                ff[i] = fun(xx[i, :],www)
        #将上一代种群与这一代种群以及最优种群结合，选取排名靠前的个体组成新的种群
        F = np.hstack((np.array([best_f]), f, ff))
        F, b = np.sort(F,axis=-1,kind='stable'), np.argsort(F)#按小到大排序,获得靠前的位置
```

```
        X = np.vstack(([best_x], x, xx))[b, :]
        f = F[:num]  #新种群的位置
        x = X[:num, :]  #新种群的位置
        best_f, a = new_min(f)  # 记录历史最优值
        best_x = x[a , :]  # 记录历史最优解
        trace = np.hstack((trace, [best_f]))
        if ii % 10 == 0:
            print('迭代次数：',ii)
    return best_x,best_f,trace

bx = []
bf = []
www = [0]
while True:
    best_x,best_f,trace = woa(sub,up,type,4,30,www)    #种群大小，迭代次数
    if best_x[0] >= 200:
        break
    print('最优解为：')
    print(best_x[0]*37.04) # 转化为 m
    print('最优值为：')
    print(float(best_f*37.04))
    kuan = 2*(best_f + best_x[0]*37.04 - 0.9*sum(www))
    www.append(kuan)
    print('覆盖宽度：',kuan)
    bx.append(best_x[0]*37.04)
    bf.append(best_f*37.04)

print(bx)
print(bf)
print(www[1:])
```

附录九：问题四指标值求解

```
y2 = [2632.200918593065, 7441.643831213292]
xf2 = [100.92049487109169, 3.1753085893049136]
kuan2 = [5269.851107967506, 5397.72712103266]

x4 = [6121.107081517494]
xf4 = [11.534919218653867]
kuan4 = [12242.83699884593]

x3 = [1705.088731086144, 4842.21912220321]
xf3 = [28.65037858887025, 40.634963444336684]
kuan3 = [3411.724458856352, 3545.52833096198]

x1 = [39.96596832952728, 112.15161328406698, 183.37751329082073,
252.70234439051754, 322.0420195137932, 388.7361137853222, 454.8751312753448,
520.809219590033, 583.5832090222578, 649.136878744187, 712.1110176570888,
774.422990043548, 836.8277992166737, 900.284891639546, 962.6451808832222,
1028.4327081539202, 1091.8276293949307, 1156.4367171458623,
1220.6897179683178, 1284.4104805358581, 1351.1094387257547,
1419.8398534904281, 1489.6153484546332, 1559.8547854442188,
1633.2938124621155, 1666.6930753057406]
xf1 = [1.2326116986677391, 2.3489591636066187, 17.67238672686084,
48.60754801905055, 22.126506265124494, 40.55986179565195, 31.442817457605415,
1.521534940403203, 68.14108833866196, 5.967231981164787, 18.640431510657418,
```

```
44.790705320337544, 63.1667061539074, 45.63639599169386, 82.51449409688851,
1.0964890463993924, 15.036321993323764, 4.594420908289067, 5.635325439379066,
47.93213271839275, 49.11737432270925, 32.517042520532264, 30.8271197633673,
40.12847922573384, 5.914603552697699, 1423.2156609700617]
kuan1 = [79.99849236632603, 80.43277395689, 78.93297974449945,
77.17364348436587, 75.51057359329593, 73.97505124134386, 72.60571297519652,
72.16798624289072, 71.41075798019438, 70.62161338597184, 70.13528514777636,
69.92771848027405, 69.85966822907153, 70.07988923860967, 70.64792509350559,
70.66049389151112, 71.01413813268482, 71.84304744252768, 71.08776803922001,
72.85515558918041, 75.1777898352475, 76.42225145152815, 78.32194023818329,
78.32355502093378, 82.37180852482197, 77.42921371714237]


ce_long = 0
ce_long += sum(x1[:-1])+sum(x4)+sum(70+(np.array(x3)-
45)*36/23)+sum(45+(np.array(y2)-70)*23/36)
print(ce_long) # 总测线长度

x = np.array(x1[:-1]+x3+x4)
kuan = np.array(kuan1[:-1]+kuan3+kuan4)
ttt = (x[1:]-x[:-1])/kuan[:-1]
print(1-ttt)

tt = (y2[1]-y2[0])/kuan2[0]
print(1-np.array(tt))
```

附录十：问题 4 海底地形分析

```
df = pd.read_excel('地理表.xlsx',header = None)
df = np.array(df)

x0 = np.linspace(0, 0.02*(len(df[0]-1)),len(df[0]))
y0 = np.linspace(0, 0.02*(len(df[0])),len(df))
z0 = -df
x, y = np.meshgrid(x0, y0)

# 绘制结果
fig = plt.figure(figsize=(9, 6))
ax=plt.subplot(1, 2, 1,projection = '3d')
surf = ax.plot_surface(x, y, z0, rstride=1, cstride=1, cmap=cm.coolwarm,linewidth=0.5,
antialiased=True)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f')
plt.title('地势图')
ax=plt.subplot(1, 2, 2,projection = '3d')
plt.contour(x, y, z0, levels=50)  # levels 参数指定等深线的数量
plt.xlabel('X')
plt.ylabel('Y')
plt.title('等深线图')

plt.show()
```